DS&OR Lab
Fachgebiet Wirtschaftsinformatik
Fakultät Wirtschaftswissenschaften
Universität Paderborn

Dissertation

Ontology-Based Representation of Abstract Optimization Models for Model Formulation and System Generation

Der Fakultät Wirtschaftswissenschaften der
Universität Paderborn
zur Erlangung des akademischen Grades
Doktor der Wirtschaftswissenschaften
- Doctor rerum politicarum -
vorgelegte Dissertation
von
Dipl.-Math. Florian Stapel
geboren am 15.10.1984 in
Herford

Gutachter:

1. Prof. Dr. Leena Suhl
2. Prof. Dr. Taïeb Mellouli

2016

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Paderborn, 25.07.2016

(Vorname Name)

Abstract - deutsch

Gegenstand dieser Arbeit ist ein neuer Ansatz zur Entwicklung optimierungs-basierter Entscheidungsunterstützungssysteme, wobei abstrakte Optimierungsmodelle die Kernkomponenten der Systeme darstellen. Um den Entwicklungsprozess und insbesondere den zentralen Schritt der Modell-Formulierung zu unterstützen, wird eine neue Darstellung solcher Modelle vorgeschlagen, die es erlaubt, die Semantik und Struktur von Daten, mathematischen Eigenschaften und den Formulierungs-Elementen der Modelle selbst geeignet auszudrücken. Die Darstellung basiert auf Ontologie-Klassen und assoziierter Funktionalität zur Herleitung algebraischer Statements. Diese Struktur ermöglicht es, Modellierungs-Fragmente in einer wiederverwendbaren Weise zu entwerfen. Formulierungs-Entitäten wie Constraints und Optimierungsziele werden von Datenmodellen und konkreten algebraischen Ausdrücken wie etwa Summationen entkoppelt, sodass eine weiter abstrahierte Sicht auf Optimierungsmodelle entsteht als es in Algebraischen Modellierungssprachen der Fall ist. Die Operationen im Schritt der Modell-Formulierung werden auf einfache Ontologie-Graph-Manipulationen reduziert und es wird die Grundlage für eine Vielzahl relevanter Automatisierungen gelegt. Im Rahmen dieser Arbeit werden verschiedene Konzepte und Technologien zur Unterstützung des Gesamt-Ansatzes präsentiert. Der Erzeugungsprozess des Systems wird erörtert, wobei semantische Modell-Services, die die Instanziierungs-Funktionalität mit Daten anbieten, in ihren Service-Beschreibungen mathematische Eigenschaften der Modelle repräsentieren. Die Services und ihre Beschreibungen können direkt aus der Ontologie-Darstellung generiert werden. Semantische Servicekomposition wird eingesetzt, um ein System welches Modell, Solver und weitere Services enthält, zu erzeugen. Zusammen mit Datentransformationen, die aus Mappings der verwendeten semantischen Typen in Ontologie-Darstellung und Service-Beschreibungen gewonnen werden können, kann ein lauffähiges System entstehen.

Der Entwurf und die Struktur wiederverwendbarer Typen von Modell-Entitäten, sowie die Unterstützung von Operationen in der Modell-Formulierung, bilden den Hauptteil dieser Arbeit. Das Wissen um Vererbungs-Beziehungen und Klassen-Restriktionen, welches in den Ontologien enthalten ist, kann die Automatisierung von Änderungen der Formulierung des Modells, beispielsweise nach einer ersten System-Ausführung, ermöglichen. Beispiele aus den Domänen der Netzwerk-Fluss-Probleme und der Planung von Wasserversorgungssystemen werden zur Evaluation des Ansatzes diskutiert. Dabei werden verschiedene Formalisierungen vorgestellt, wie zum Beispiel die Glättung und Linearisierung nichtlinearer Constraint-Gruppen.

Stichworte: Algebraische Modellierungssprachen, Entscheidungsunterstützungssysteme, Model Formulation, Model Management, Semantic Web Services, Service Composition, Structured Modeling

Abstract - englisch

This dissertation presents a novel approach for developing optimization-based decision support systems in which problem specific abstract optimization models constitute the core components. In order to support the development process and especially the task of model formulation, a novel representation of such models is proposed which allows to suitably express the semantics and structure of data, mathematical properties and the problem formulation elements themselves. The representation is based upon ontology classes and associated derivation functionality for algebraic statements. The latter structure allows to design respective modeling fragments in a reusable way. Formulation entities such as constraints and objectives are decoupled from data models and concrete algebraic terms such as summations, which provides a further abstracted view in comparison to Algebraic Modeling Languages. The operations in the model formulation step are reduced to simple ontology-graph manipulations as well as that a fundament for a variety of meaningful automatizations is laid. Within this thesis, different concepts and technologies to support the overall approach are presented. The generation process of the system will be discussed where semantic model services that provide the functionality to instantiate a model with data represent the mathematical properties of models in their service descriptions. The services and descriptions can be generated straight forward out of the ontology representation. Semantic service composition is exploited to build a system with model, solver and further services. Together with data transformations generated from mappings of the semantic data types included in the ontology representation and service descriptions, a runnable system can be retrieved.

The design and structure of reusable model entity type formulations, as well as the support of operations in the model formulation, constitute the main part of this work. The knowledge about subsumption relations and class restrictions that is represented in the ontologies may allow to automatize changes in the model formulation, e.g., after a first execution of a system. Examples from the domains of network flow problems and water distribution system planning will be discussed to evaluate the approach, thereby providing formalizations of different model formulation fragments and operations such as the smoothing and linearization of nonlinear constraint-groups.

Keywords: Algebraic Modeling Languages, Decision Support Systems, Model Formulation, Model Management, Semantic Web Services, Service Composition, Structured Modeling

Preface

Developing ideas is a fascinating process. You start with a very vague vision or maybe some fragments coming to your mind and end up with a whole book of 250+ pages. In the case of developing an idea to a research fragment, the identification of a research question, the formulation of a goal and the respective exploitation of a research methodology are important for sure. Furthermore, analytical thinking and logical deduction are the typical tools to achieve a concrete result out of a vague idea. But the process of developing an idea itself can not be completely demystified. There always is a black box of creativity and experimentation required to push things forward and explore new directions. I write this with the view of a mathematician and also as a computer scientist with a growing interest in business information systems. But I would strongly believe that the statement remains valid for authors and artists. Following the creative research process can have interesting intermediate results such as a piece of paper being marked all over with diagrams and greek letters or, even more mysterious, images of sheep occuring on whiteboards and further infrastructure of a working group. But in the end, when the work is done, many things occur simple and clear.

When I started my work at DS&OR Lab in early 2011, there was a formal description of the research project C3 "Modeling of Optimization Problems" which was embedded into the visionary Collaborative Research Center 901 "On-The-Fly Computing". To be honest, it really took a comparably long time to identify a core research question and develop new ideas. But with the here achieved results, I would describe the process that took place in the last years as a consequent development of the initial aspects and ideas. For sure, all this would not have been possible without the aid of many people to whom I would like to express my gratitude.

Acknowledgements
At first I would like to thank Prof. Dr. Leena Suhl for supervising this thesis, including several thorough discussions, different opportunities for funding and networking, and finally, motivation. The next big thank goes out to Corinna Hallmann who accompanies me as a collegue and friend for many years, including the studies of mathematics and working at DS&OR. Thanks for making many things possible, such as starting together into the CRC and always organizing a continuous supply of chocolate with expectation value Wednesday.

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing (SFB 901)". I would like to thank my project partners, especially Felix Mohr, Sven Walther, Marie Christin Platenius, Svetlana Arifulina and Alexander Jungmann, for inspiring discussions and the German Research Foundation for funding this project.

I would like to thank Prof. Dr. Taïeb Mellouli and his working group for taking the time for a thorough discussion on the research work of this thesis. I also thank Prof.

Mellouli for being the second referee. Special thanks go out to Dr. Michael Römer for arranging the meeting at Halle and providing many ideas in the discussion. In this context, I also would like to thank Hendrik Nieß for many ideas and the thorough discussion at Halle.

I would like to thank Dr. Kostja Siefen for his encouragement in discussing a requirements engineering ansatz for the development of optimization models and systems. The latter ansatz also had a strong emphasis on optimization models in the development of dss and as such the discussion was fundamental to me for starting to evolve the idea of a semantic optimization model representation. Further thanks in this context also go out to Lars Beckmann, whose concept of an "Optimization Framework" always showed me further important aspects of the big picture of optimization system development.

Big thanks go out to my family and friends for supporting me over the last five years. I thank Alexandra Bäcker for assisting me over all the time and being at my side. Furthermore, I especially would like to thank my parents for always being there. A big thank goes out to Lieselotte Bäcker for proofreading major parts of this document. I also would like to thank the whole family Bäcker for supporting me over the last years.

Paderborn, July 2016

Contents

List of Figures

1. Introduction

Decision support systems (dss) and optimization technology provide powerful tools for decision makers. The ongoing development in solver technology allows for the solution of more and more practical problems in acceptable time. In addition to that, tools for formulating and managing optimization models provide expressive languages as well as supporting functionality to modelers and system developers. Together with the ongoing research in the field of operations research, decision support can be provided for a wide and growing range of application areas such as supply chain management, home health care or public water supply.

With the growth of application areas and methods, suited support for the development of optimization-based decision support systems becomes more and more important. Concerning approaches and frameworks for the formulation and solution of optimization models, areas of interest cover the suited representation of models, the integration of data-sources, as well as means to generate complete prototype systems in an integrated development environment (, see, e.g., AIMMS [RB13]). Besides the monolithical approach of algebraic modeling systems such as AMPL, GAMS, AIMMS, LINGO or FICO Xpress Optimization Suite [FGK03, McC14, RB13, LIN07, FIC16], frameworks such as the Python Optimization Modeling Objects (Pyomo) [HWW11, HLWW12] provide novel development capabilities for formulating, analyzing and integrating optimization models into decision support applications. Furthermore, service-oriented architectures provide an emerging paradigm for conceptualizing software systems based on reuse. Concerning the latter, web service approaches for distributed optimization, e.g., allow to invoke solver functionality remotely over the internet, where the binding to a service can be changed dynamically. Exemplary web service frameworks are, e.g., given by the two projects Optimization Services (OS) [FMM10b] and Open Optimization Framework (OOF) [EM03].

The central role of decision models in the development of dss is reflected by the research field of model management. Model management covers different questions concerned with decision models in a general modeling lifecycle. Besides others, suited representations and the provision of means to formulate, integrate and reuse decision models, are of interest. The framework of Structured Modeling (SM, see, e.g., [Geo87]) and its extensions besides others allow to reuse modules of model formulations, integrate models into new models, as well as to perform semantic search for and reasoning on models by respective representational extensions.

Outside the SM framework, a work by Binbasioglu [Bin96] discusses configurable types of linear model formulation elements, providing a first effort to not only reuse and integrate greater blocks of model definitions, but to also base the model representation upon reusable formulation conceptualizations. Unfortunately, the latter approach is only rarely formalized and restricted to linear problems that are typically of the class LP. Together with the missing semantics of an underlying language framework, the possibilities for support and automatization in the model formulation of general mathematical programming models seem to be limited. Furthermore, the portability of specifications seems hard to achieve. In addition to that, the requirements on a suited formulation type and model representation approach can also be extended by a development aspect: A widely automatized generation of systems by means of service composition, where the model specification builds a core, is desireable. Altogether, a research gap that is identified within this thesis can be described as follows: Current optimization model representations cannot cover suited model formulation support, reusage and the requirements for service composition, in a holistic approach. In order to deal with the respective issues, this thesis provides a novel framework and representation.

1.1. Approach within this Thesis

This thesis is concerned with a novel optimization model representation and system generation approach. Abstract optimization models will be represented semantically, within ontologies. The ontology representation serves as a layer inbetween algebraic modeling languages (AML), and the layer of software services in which special services provide the functionality to instantiate models with data. On the service level, the models can be composed with further services, e.g., for solvers, as well as that data transformations might be generated in an automated fashion based upon mappings of semantic types. A runnable system can originate out of a model with only a few specificational steps. In order to generate semantic service descriptions, methods to analyze ontology-represented models will be presented. The ontology representation itself exploits the aforementioned concept of reusable types for goal and constraint formulations. Furthermore, also types for data entities will be introduced. The formulation types are abstracted from data models and allow to reuse blocks of AML statements in different model contexts. Means for deriving AML-represented models out of ontology-represented ones will be provided, as well as that the general design of formulation types will be discussed and exemplified by two elaborate case studies and further examples.

The reusable types formalize definitions of model entities such as goals, constraints, variables, parameters and sets. Model entities are represented by ontology individuals and typed in ontology classes which respectively represent the types. A whole abstract optimization model is described in terms of ontology axioms that correspond to the class restrictions of the respective types. The axioms represent the relations between entities, such as a parameter being defined over an index-set or a variable occuring in the definition of a constraint group. By providing vocabularies for domain data models, the classes for constraint and goal formulations can be abstracted from the data types. The organization of formulation types exploits concepts such as subsumption, in order to define a type as an extension of another one, and allows for simple operations in the model formulation. As standardized ontology languages such as OWL 2 [W$^+$12] are designed for portability and an enhanced Web architecture, the usage of ontologies for the optimization model formalization furthermore enables different scenarios for automated reasoning by agents, as well as the usage with software services.

This thesis presents different technologies towards the realization of the model formulation and system generation approach that was outlined above. The transformations inbetween the three layers as well as the vocabularies to define reusable formulation types constitute central components. Exemplary formalizations for case study models from the domains of network flow problems and water distribution systems will be given to demonstrate the definition and organization of respective type hierarchies, as well as to illustrate the different application scenarios in model formulation. Different design considerations will be discussed and a demonstrator application for the derivation of AML models out of ontology-represented ones is provided with the digital material accompanying this thesis. Numerical results for a novel optimization model dealing with the pipe renewal planning of water distribution systems round up the considerations of this thesis. The results provided within this thesis can be seen as a contribution to a greater vision. In this vision, different models and semantic services can be composed into systems in an automated fashion. In the latter point, the vision extends the restricted service architecture of this thesis to a wider range of models, software components and execution processes. Also, some technologies provided to make the findings of this thesis applicable in practice are only outlined within this thesis. This, e.g., covers the exemplary description of methods to automatically recommend formulation types to a modeler in a given model context. To that end, a broad perspective for future research comes along with this thesis.

The efforts towards a highly automated composition of software systems, e.g., for optimization, are also reflected by a research project at the University of Paderborn in which the author of this thesis was engaged. The CRC 901 - On-The-Fly Computing [CRC11] - is a current research project that is concerned with the opportunities of providing general and individual IT-services by semantic service compositions, where services are available on global markets. As a part of this vision, the application scenario of optimization systems has also been investigated within a collaboration during the first period of the research project. As such, the author of this thesis was part of this collaboriation in parallel to the work on this thesis. The collaborative work especially investigated the more top-down approach of composing optimization systems without the here proposed ontology representation, ontology model transformation and analysis methods, and architecture. As such, the findings of this thesis can be seen as a novel contribution to the "On-The-Fly Computing" vision, especially concerning architectures and application scenarios.

1.2. Outline and Methodology

This thesis is structured according to the design science methodology presented by Peffers et al. in [PTRC07]. The description of the six process model steps *problem identification and motivation, objectives of a solution, design and development, demonstration, evaluation* and *communication* is contained in that order within the chapters, sections and subsections of this document. Before the *problem identification and motivation* can be described, a broad literature survey is given in Chapters 2 and 3 which reviews the state of the art in the different relevant fields, as well as that fundamental notions for the concepts introduced in this thesis will be presented. Whilst Chapter 2 deals with software services and ontologies in general, Chapter 3 reviews the state of the art for managing decision models and developing software systems for optimization and decision support. Chapter 4 consists of two consecutive sections that conclude the *problem identification and motivation* and describe the *objectives of a solution* in terms of requirements. The problem identification and motivation part also contains the descriptions of some use cases. The *design and development* part is mainly dealt with in Chapter 5 which presents the framework of this thesis in a top-down manner. The most important part is given by the definition of the ontology representation of abstract optimization models. Practical and important design considerations on a more specific level will also be part of Chapter 6. Chapter 6 in general provides means for *demonstration* and *evaluation*, as different type formalizations will be demonstrated and further discussed in the context of some case study models. The core topic of Chapter 6 is the discussion of reusability and model formulation in the ontology representation. Chapter 7 presents a practical optimization model for the renewal planning in water distribution systems with some numerical results. This extends and underlines a preceding case study of Chapter 6. Though the main character of Chapter 7 is of practical optimization nature, some application scenarios for the concepts of the preceding chapters will be discussed. In Chapter 8, the architecture and usage of a demonstrator application for deriving AML models and statements out of ontology-represented models will be described, thereby further extending the considerations for *demonstration* and *evaluation.* The Conclusion and Outlook in Chapter 9 is separated into the two naming parts and therefore provides means to recapitulate the results of this thesis and watch them as part of a greater vision towards which further research and work should take place.

2. Software Services and Ontologies

In this chapter, important techniques from the fields of semantic software services and ontologies will be discussed. This shall provide the background for the general approach and the design of the ontology representation of abstract optimization models which will be introduced in Chapter 5. This chapter starts with a short recap on component-based software engineering and then introduces software services and web services as basic concepts. Furthermore, ontologies and related technologies will be introduced as a core concept for the approach of this thesis. Ontologies also provide a basis for introducing semantic software services, their composition and means to provide automated data mediation in service workflows at the end of this chapter.

2.1. Component-Based Software Engineering

Component-Based Software Engineering (CBSE) is a methodology for developing software which is based on reuse. The concept of a *software component* forms its central building block where a component-based software development process for new systems is based on integrating existing components by means of *component composition.* To that purpose, components are described structurally and semantically by so-called *interfaces, contracts* and *extrafunctional properties* (see [CHJK02]). This makes the provided functionalities and possible dependencies explicit such that components can be deployed independently and integrated into applications, e.g., as binaries or services, without accessing their source-code. In order to standartize the necessary descriptions of software components and allow for a successfull composition, *component models* serve as the basis for component-based software engineering. With implementations of component models that can be referred to as *component frameworks*, further supporting middleware services are typically supplied. By this, a developer can focus on the main tasks of integrating components into new applications.

The purpose of this section is to give a short introduction to some of the major concepts of component-based software engineering as there are *interface definitions, component composition* as well as *pre- and postconditions.* The concepts introduced here are important for understanding software services which can be seen as a specialized form of components. To that end, this section includes basic concepts for presenting the approach of this thesis. This subsection is not meant as a state-of-the art review or a complete depiction of CBSE. Further relevant topics not treated here cover *development processes, contracts, the specification of extrafunctional properties* or special *component frameworks*. The interested reader is refered to the literature which will be presented now:

Two good introductions on CBSE which also served as a basis for this section can be found in the book of Sommerville [Som12] and the article of Crnkovic et al. [CHJK02]. Szyperski [Szy02] presents a standard book on CBSE which includes a common definition of components. This section refers to the 2002 version of the book. A newer version from 2011 is also available. Finally, for a classification and discussion of component models, the reader is referred to the article of Lau and Wang [LW07] from 2007.

There are different definitions of components and even broader is what component frameworks implement. This thesis refers itself to the definition of Szyperski [Szy02] who defines a software component as follows: "A software component is a *unit of composition* with contractually specified *interfaces* and explicit *context dependencies* only. A software component can be deployed independently and is subject to composition by third parties,". A short recap on common definitions including the latter can be found in the article [LW07].

The definition above highlights a software as an independent unit that can be combined with others such that a new system originates. It does not incorporate the notion of component models which are included in other definitions and provide the basis for any practical implementations.

Software component models "*define standards for implementation, documentation and deployment of components*" [Som12]. Upon these standards the definition and implementation as well as the composition of components into a running application can take place. Following Lau et al [LW07] and Sommerville [Som12], the most important component models seem to be Web Services, Enterprise-Java-Beans and .NET. Implementations of component models may provide some standartized services to the components that are being used in a concrete software project. These services may roughly be classified as *platform services* and *helper services* [Som12]. Platform services allow for communication and interoperation in distributed environments whereas helper services may provide more concrete functionality that is being used in applications with a high possibility. Examples of the latter include *resource management* or *authentication*.

The most important part of a component specification is the definitions of its *interface*. Interfaces in CBSE are separated in *provided* and *required* interfaces. A provided interface defines the methods that users of a component can call whereas the required interface defines methods which are required from other components for the component itself to work. Though components are designated to be independend building blocks, the requiered interface dependencies are allowed in CBSE and represent a major difference of components and *software services*. The most important thing about the interface definitions are that they have to be made explicit and should be complete in order to reuse a component as a black-box in different contexts.

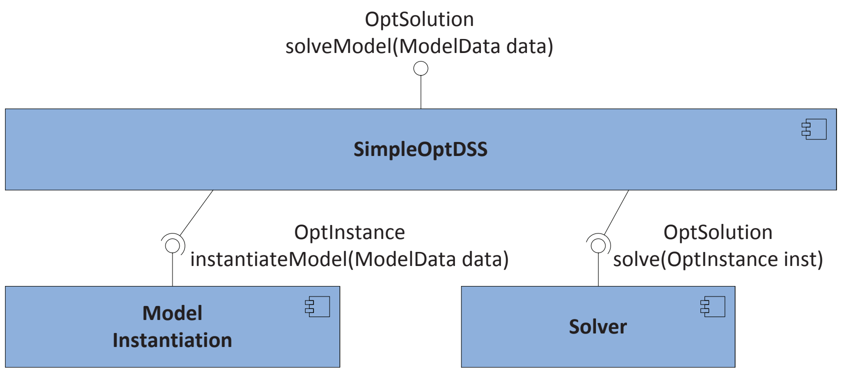


Figure 2.1.: Sequential composition of a model and a solver

A simple example of components and parts of their interface definitions is visualized in Figure 2.1. The drawing will also serve as a basis for explaining *sequential component composition* in what follows. The image shows three components. The component `SimpleOptDSS` provides the solution of an *abstract optimization model* for some scenario data by its provided interface's method `solveModel(...)`. Readers which are not familiar with optimization software and optimization models are refered to Section 3.1 which presents the basic concepts in the fields related to the latter notions. For the moment it should be sufficient to know that an abstract optimization model provides a mathematical description of a computational problem where the word abstract is used in order to clarify that the optimization model itself defines some set-and parameter-based data model that can be instantiated with concrete instance data.

On can see in Figure 2.1, that the provided interface of a component is visualized by a circle which is annotated with some methods and their signatures. In the example, a description of a single method `solveModel(...)` is given. In order to provide the functionality of this method, `SimpleOptDSS` has to invoke two methods of other components called `Model Instantiation` and `Solver` in that order. The complete method signatures with some descriptions are given as follows:

- `public OptSolution solveModel(ModelData data)`: This method computes a decision proposal in form of an optimization model solution for some instance data. To that end it instantiates a model with the data (component `Model Instantiation`) and calls a solver (component `Solver`) with the retrieved instance.

- `public OptInstance instantiateModel(ModelData data)`: This method instantiates a fixed optimization model with instance data.

- `public OptSolution solve(OptInstance inst)`: This method solves an optimization problem which is given as an instance (with concrete data) by some implemented optimization algorithm.

The three components form an example of a *sequential composition*. There are three kinds of *component composition* in CBSE which will be introduced now:

Composition of Components
When components are integrated into new systems, three kinds of composition can be distinguished. Following [Som12] there are:

1. *Sequential Composition*: By sequential composition new components or systems are generated from existing components where the latter are executed in sequence. The example from Figure 2.1 provides an example for a sequential composition where the consecutive execution of provided interface's methods from the `Model Instantiation` and `Solver` components yield a new functionality that directly solves a model for given input data. Sequential composition is also possible for services. It is important to notice that one needs additional code to perform the calls to the individual components in the composed component. In addition to that, mediation code to transform the output of a preceeding service into an input of the next service to call may be required for the composition.

2. *Hierarchical Composition*: This kind of composition is used when a component directly calls other components and all components shall be integrated. In a situation with a component A calling methods from a component B, the required interface of A should be matched by the provided interface of B. If the interfaces are not directly compatible, an adaptor component has to be inserted to adjust the data types of inputs and results and hence provide a matching of parameters. Figure 2.2 provides an illustration how provided and required interfaces are combined in hierarchical composition.

3. *Additive Composition*: The additive composition is used to combine the provided functionality of two ore more components. Since mutual components can be integrated by this kind of composition, mediation code is both needed for managing the calls and inputs to the individual components over a combined provided interface, as well as for performing the calls to further components that were first required by the individual components and are now required within a unified required interface of the composition. The whole situation is visualized in Figure 2.3. This kind of composition can also be used for services.

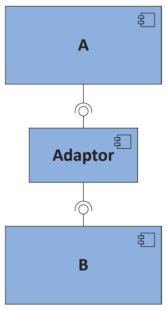


Figure 2.2.: Hierarchial composition with an adaptor. Extended version of an image from [Som12]

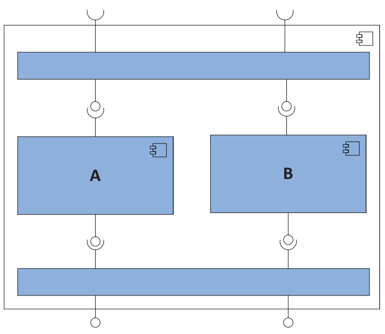


Figure 2.3.: Additive composition with glue code. Image in similarity to [Som12]

Pre- and Postconditions

Method names and signatures specify the semantics of a method only to a limited degree. To that end, a certain form of documentation is required. In CBSE, the usage of Object Constraint Language (OCL) [WK03] is a common and formal approach for documenting interface methods (see also [Som12]). In OCL, so-called *pre-* and *postconditions* of methods can be specified.

The following example refers to the components in Figure 2.1 where the the method `solve(...)` from the provided interface of the component `Solver` is described:

```
context  solve
  pre:   Solver.getSolution(inst) = null
 post:   Solver.isOptimal(Solver.getSolution(inst)) = true
```
(2.1)

The precondition says that a solution for the passed instance should not exist before

the execution. To that end another method `public OptSolution getSolution(OptInstance inst)` from the provided interface of the component `Solver` is used. After the execution of `solve(...)`, the solution can be obtained by a call to `getSolution(...)`. Using the method is not a requirement since a reference to the optimal solution can also be obtained directly as a return value from a call to `solve(...)`. But one can use `getSolution(...)` as a building block to formulate the postcondition. The method `public Boolean isOptimal(OptSolution sol)` returns true iff the solution object passed to the the method represents an optimal solution.
Pre- and postconditions will be considered again in the context of semantic service descriptions in Section 2.6.

2.2. Services, SaaS and SOA

This section will introduce the notion of *software services*. The concept of *Software as a Service* (SaaS) will be presented and compared to *Service-oriented Architectures* (SOA). A special focus is laid on service concepts for optimization software.

Refering to Turner et al. [TBB03] "SaaS focuses on separating the *possession* and *ownership* of software from its *use* [TBB03]". This stresses that SaaS is a methodology of making software accessible instead of an architecture or design philosophy. By SaaS software is typically being made available over the internet and can be accessed by browsers [Som12]. The provision and maintainance of the service is up to a service provider which can be different to the developer of a software. By standard internet protocols, clients of a *software service* can access remote funcionality on the basis of messages. The customer that uses a software service does not have to care about deployment issues, hardware ressources or the management of updates.

Lovelock and Wirtz [LPW14] define general *services* in their book on services marketing and compare the notion of a service to that of manufacturing. Besides others they highlight the following definition of a service: "A service is any act, performance or experience that one party can offer to another and that is essentially intangible and does not result in the ownership of anything, but nonetheless creates value for the recipient. Its production may or may not be tied to a physical product."

Software services are covered by this definition. The requester of a service that invokes a service over the internet won't own a copy of the software but rather just get a response to his specific request that might be of some value for him.

There are different paradigms for the provision of commercial software services in SaaS. Valente and Mitra [VM07] highlight the difference between *Application Sevice Provider* (ASP) and the general *e-Service* paradigm for e-commerce. In ASP, a service provider "employs the personnel needed to install and maintain the application and the servers" [VM07] where the focus lies on the technical outsourcing of service provision from the developer of a software. In contrast to that, the e-services paradigm presented in [RK03] lays a focus on the customer, e.g., by providing customer care: "In the downstream channel, e-service subsumes concepts such as customer/citizen-relationship management (CRM), relationship marketing, one-to-one marketing, and customer care [RK03]".

In SaaS, software services are services for a customer, where the users use the services of a service provider technically as a client. By this model, technical standards or restrictions are not directly imposed but one may rather highlight a standard use case where the client accesses the service via a browser. Examples are, e.g., given by Google Docs or the NEOS Server [CMM98] for numerical optimization where the customer, besides others, can get results for browser submitted jobs via email. Interactions with software services provided by SaaS are typically long (Google Docs) and are manually induced by a human user (Google Docs, NEOS Server).

When custom applications invoke a remote service directly, message based communication can be used. The *Web Services* approach [HB04] provides a standard for this form of machine to machine interaction where especially message formats and interface descriptions are defined. Furthermore, the specifications in the context of web services include means for the discovery of services, the definition of workflows that invoke multiple services, security issues and more. Web services will be further discussed in the next section. By the provided standards and technologies, flexible connections between servers of different enterprises can be established [Som12]. Applications that make use of multiple extenal web services provide an important example for the realization of a *Service-oriented Architecture* (SOA) [Erl05, OAS12]. Different definitions of SOA exist. Following [The15], the notions *SOA* and *service-orientation*, as well as another definition of a *(software) service* in the context of SOA are to be distinguished:
"*Service-Oriented Architecture* (SOA) is an *architectural style* that supports *service-orientation. Service-orientation* is a way of thinking in terms of services and service-based development and the outcome of services. A service:

- is a logical representation of a repeatable business activity that has a specified outcome (e.g. check customer credit, provide weather data, consolidate drilling reports)
- is self-contained
- *May* be composed of other services
- is a "black-box" to consumers of a service

[The15]"

By this definition no technical restriction, e.g., on using the web service standards are made. SOA rather highlights important features such as the perspectives for integrating *business activities* by the means of *workflows* where services provide reusable functionality to be used by different acteurs. Another important property is that software services are *loosely coupled*, i.e., the services that are incorporated in a system can be exchanged, e.g., by a newer version, during the execution. Compared to SaaS, SOA is an architectural design that can be used with SaaS.

Valente and Mitra [VM07] discuss the evolution of web based optimization in terms of a shift from ASP to e-Services. An important project in the scope of this article is the WEBOPT project which implements a system that provides different optimization functionality in a SOA by using web services. With WEBOPT a fundament for composing different services to new subsystems in a wokflow is laid: "The infrastructure provided by WEBOPT enables the creation of distributed applications which may take advantage of multiple specialist knowledge" [VM07]. WEBOPT is an example for the synergy of e-Services and the web service technology and extends the Optimization Service Provider project (OSP) which was based on older technology and the ASP model.

Technological standards for realizing optimization solvers and model related functionality (see Chapter 3) as web services have been defined in the projects *Optimization Services* (OS, Fourer and Ma [FMM10b]) and *Open Optimization Framework* (OOF, Ezechukwu and Maros [EM03]). Whilst OS has a solver-centric view and defines XML-languages for optimization related data, communication, discovery and registration, OOF also lays a focus on *Algebraic Modeling Languages* (AML) by providing an XML-based algebraic modeling language also called AML.

Optimization software, architectures and the technology behind will be further discussed in Chapter 3. In the next section, the discussion of software services will be continued by introducing web services in more detail.

2.3. Web Services

The goal of this section is to introduce important concepts related to the notion of web services. Web services provide a common standard for the description of software services. Therefore, they also serve as a basis for the concept of *semantic services* which will be of further importance in the remainder of this thesis.

Following Haas and Brown ([HB04]), a web service can be defined as follows: "A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner described by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [HB04]".

The definition above highlights the standars used for the *description* of and *communication* with web services in order to provide a platform independent machine-to-machine interaction. Let us now shortly characterize the relavant aspects of these standards:

WSDL:
A web service is to be described by a service description in Web Service Description Language (WSDL) format. The current W3C recommendation WSDL 2.0 [SAJR07] requires service descriptions to be built as XML documents with respectively structured information under the tags `types, interface, binding` and `service`. The first two tags define the *abstract section* of the service description and are used to define the logical interfaces of a service, i.e., the provided operations with their signatures, respective messages (implicitely in WSDL 2.0 as XML schema types) and types. The last two tags build the so-called *concrete section* and describe relevant information about the communication with the service such as the bindings to concrete message protocols, message exchange patterns, or the addresses under which the service can be contacted.

SOAP:
Refering to the W3C recommendation [JNM+07], SOAP ".. is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment [JNM+07]". SOAP is based on XML for representing data and can be embedded into standard internet protocols such as HTTP from the application layer. The `Envelope` tag in a SOAP-message is used to capture header and body information. When applied for communication with a web service, the specifications under a `Body` tag can, e.g., be used to describe method calls and their inputs.

UDDI:
Universal Description, Discovery and Integration (UDDI) [CHRR04] denotes a standartized registry service in the context of web service specifications. It is an approved OASIS standard for the publishing and discovery of (web) services as well as that it defines "a set of services supporting the description and discovery of (1) businesses, organizations, and other Web service providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services [CHRR04]".

Following [Som12], different registries have been implemented by major software vendors such as Microsoft at the beginning of the 21. century. But due to an improved search engine technology all these registries have been deactivated by now. The standard way for discrovering services may now be described as using standard search engines that look for respectively commented WSDL descriptions (see, [Som12]).

The standards introduced the last can be used for the *publication, discovery* and *understanding* of, as well as the *communication* with web services. But besides WSDL their usage is no requirement. The *Web Services Architecture* [BHM+04] that is visualized in Figure 2.4 formaly specifies the different roles involved in the publication and usage of a web service:

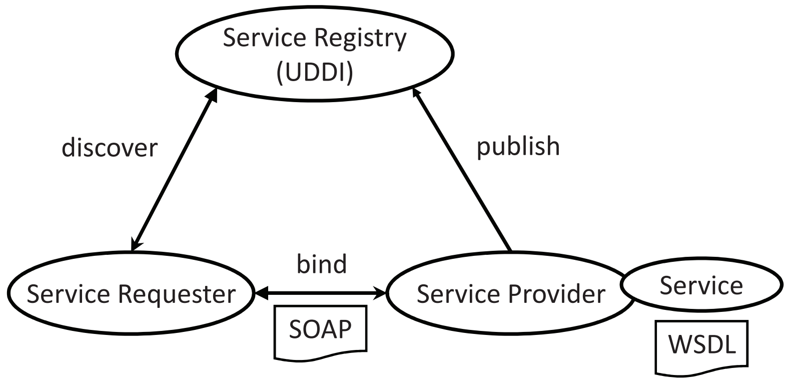


Figure 2.4.: Web Services Architecture. Image in similarity to [Som12]

The *service provider* implements a web service and generates his *service description* in WSDL. He makes the service available and, in order to be found, publishes the web service, e.g., in an UDDI registry. A *service requester* can then discover the web service and contact the *service* under the given address. By retrieving the WSDL he can decide wether the web service fulfills his requirements or not. Finally, by binding to a concrete protocol such as SOAP, the service requester can invoke the service.

Further standards for web services have been developed until the current date. The specifications which are referred to as "WS-*" are not owned or standartized by a single organization. Important examples include security specifications such as *Web Services Security* (WS-Security) or specifications for business processes such as the *Web Services Business Process Execution Language* (WS-BPEL). Both the latter have been published by OASIS.

Different apsects of the web service standards have been critizised. The lack of semantics that may go along with specifying "only" method signatures and types will be a topic in the later sections of this chapter. The complexity of the various "WS-*" specifications has been criticised ([Bra12]) as well as that an alternative for simple publication of and communication with web services, that are not required to publish their interface in form of a WSDL-description, e.g., because they are somehow "enterprise internal", has been proposed:

RESTful services [RR08] have been introduced as a simple altenative which employs the *Representational State Transfer* (REST) programming paradigm for distributed systems. Communication with a RESTful service for the requester is to be done by the HTTP methods GET, PUT, POST and DELETE and a requester will be delivered the representation of a resource (to be identified by an URI) for an HTTP request. The operations are idempotent such that multiple request at different points in time yield the same result. Following [Som12], Restful services are considered to have a smaller overhead than web services and find applications in many enterprises that implement service-based systems, where the services are "non-external".

2.4. The Visions of Semantic Web and OTF Computing

In the section, two visionary projects whose ideas are relevant for this thesis will be introduced. The first one will be the vision of the *Semantic Web*. The semantic web vision contains standards for ontology and query languages which will be introduced in the next section and are important for the approach of this thesis. The second vision will provide important concepts for semantic service composition.

The Semantic Web can be described as an initiative to "lead the use of the Web as an exchange medium for data as well as for documents [W3C15a]". Its initial idea has been described in the 2001's article [BLHL01] by Tim Berners-Lee, James Hendler and Ora Lassila. Besides others, the article presents the story of a use case where different agents arrange appointments for a therapy. The therapy involves different specialists at different locations and the agents who arrange the appointments make use of some expressed information that extends the structural information given with actual web technologies by semantics. This extended web is described as follows: "The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [BLHL01]".

In order to realize the vision, different technologies are mentioned as a requirement. The knowledge to be formalized needs to be represented with techniques from the field of *knowledge representation* where not only web pages are extended with information, but the information adheres to a certain vocabulary that is being defined in the form of *ontologies.* The formal semantics of *ontologies* together with *rule knowledge* serve as a basis for deducting new knowledge or integrating knowledge from different sources in a consistent way. To that end, software *agents* may crawl the web for information and invoke other resources or agents in order to answer complex *semantic queries.*

The notions introduced above will be further explained in the following sections. This section now continues with a short description of current research activities related to the initial vision of the "Semantic Web".

Today, activities such as *Linked (Open) Data* [BL15] and the W3C *Data Activity* [W3C15a] are related to the primary vision. In *Linked Data*, structured data is published in form of URIs that can be resolved on a network by performing requests based on the HTTP protocol. The underlying data structure is given by the

Resource Description Framework (RDF) [W3C15b] which allows to store the data in an extensible graph that is build of so-called *triples* which relates a *resource* node with another *resource* or a *literal* node by a *property.* The resolving of uris by HTTP requests can iteratively retrieve new RDF-Data and by this extend the RDF graph. The extension of knowledge associated with these activities may allow for the answering of more and more complex queries. Tim Berners-Lee names four design principles in [BL15] for the publishing of linked data that enforce the usage of (HTTP)-URIs, RDF* and SPARQL (see Subsection 2.5.3). These are added by a fifth one about open content in the remainder of the same note in order to yield *Linked Open Data.* A famous example of linked open data is given by the DBPedia project [DBP15], an initiative that allows to extract structured information from Wikipedia by the means of semantic queries.

The W3C Data Activity Group [W3C15a] is an initiative that "merges and builds upon the eGovernment and Semantic Web Activities [W3C15a]". Its aim has been described to "make data publication less of a specialist activity and ensure that the excellent work done by portals does not lead to *de facto* data silos [W3C15a]". This goal can be seen as motivated by the fact that the hughe amounts of data published today are not published on web sites as in the primary semantic web vision, but rather "through portals that act on behalf of multiple agencies [W3C15a]".

Software services and especially web services were introduced in the preceeding sections. Software services of a certain form also play an important role in the vision of the semantic web. The need for semantically described web services that allow, e.g., for a better discovery and matching of capabilities is highlighted in the article [BLHL01] as well as the book [BKM08]. But though *Semantic (Web) Services* are highly motivated by the vision of semantic web, the research on semantic service descriptions, matching and semantic service composition forms own fields of research. One encompassing concept for these techniques is that of *Semantic Service Provisioning* [KTSW08]. The latter techniques and notions will be further introduced and discussed in Section 2.6.
An important approach for semantic service provisioning in the actual development of the semantic web vision is the Semantic Automated Discovery and Integration (SADI) Web service Design Pattern proposed by Wilkinson et al. in [WVM+11]. SADI services are atomic and stateless. They simply process individuals of owl classes (see Section 2.5.1) by annotating them with new information on the basis of rdf-triples. By doing this, the concept of linked open data which was introduced above can be realized in a simple manner. SADI services have a very simple interface description in terms of OWL classes. The restrictions of these classes describe the input and output data semantically. The authors describe different plug-ins and environments that allow the simple discovery and pipelining of SADI services. The application of SADI services to bioinformatics is highlighted.

Broadening the scope from the context of the semantic web, semantic service provisioning can be motivated as an improvement in the development of service-oriented systems in general. Kuropka et al. [KTSW08] have named the following initial limitations (2004) of service-oriented systems in the introduction to their book that features the results of the *Adaptive Services Grid* (ASG) project: *Static Discovery and Binding, Fixed Service Landscape, Static Service Composition* and *Poor Service Level Agreement Specification.* These limitations are identified as mainly due to the missing of semantically rich service specifications that allow for an automated discovery and integration of services even at runtime. In the book they present an architecture for a semantic service provisioning platform. With the experiences on service descriptions, automated composition and a use case the authors come to the conclusion, that future (advanced) research on service engineering should besides others focus on real-world use cases and the level of detail for service specifications.

The ongoing research on semantic service provisioning encompasses different fields as, e.g., *service matching* and *service composition.* The CRC 901 - On-The-Fly Computing - is a current research project at the University of Paderborn that is concerned with the opportunities of providing general and individual IT-services by semantic service compositions where services are available on global markets. The wide scope vision also includes economic market and security aspects as well as the concept of so-called "On-The-Fly" Compute Centers that are concerned with the provision of hardware services for the performant execution of resource demanding applications as, e.g., in mathematical optimization.

The CRC 901 - On-The-Fly Computing states its objective and scope as follows: "The objective of CRC 901 - On-The-Fly Computing (OTF Computing) - is to develop techniques and processes for automatic on-the-fly configuration and provision of individual IT services out of base services that are available on world-wide markets. In addition to the configuration by special OTF service providers and the provision by what are called OTF Compute Centers, this involves developing methods for quality assurance and the protection of participating clients and providers, methods for the target-oriented further development of markets, and methods to support the interaction of the participants in dynamically changing markets [CRC11]".

The CRC 901 is an ongoing project. Within the project techniques for *service description, service matching* and *service composition* are developed. The research within the project further spins the progress that has been made in Semantic Web activities or the ASG project towards a vision of automatically configured and provided IT services, where the global market perspective becomes a novel focus, and further requirements such as those for security and hardware aspects, as well as economic questions, have been identified.

With the consideration of research activities on semantic data and software services in this section a rough overview on techniques for discovering, searching and integrating data and services into new applications was given. The following sections of this chapter will further introduce the relevat techniques, especially ontologies in the next section and semantic (web-) services in Section 2.6.

2.5. Ontologies

This section introduces ontologies and related technologies such as representation and query languages. The last section has already motivated using ontologies for the description of semantic software services and in the vision of the semantic web. The term ontology originates from philosophy, where it forms a certain branch, but has become a novel meaning from the computer science research in the last decades.

Ontologies in the computer sciences are used to represent knowledge about excerpts of the real world, called domains, in a well formalized way. The formalization takes place in concepts and their interrelationships. Besides others, it has the goal to generate a vocabulary that can be used to share data between different applications or systems in a system independent way. An ontology is hence designed for a certain purpose, but will be used by multiple acteurs. An often quoted description of ontologies is given by Gruber [Gru93] as an "explicit specification of a conceptualization".

An important aspect of ontologies is that the ontology languages such as the Web Ontology Language (OWL, see 2.5.1) in its *DL-Profile* are related to a subset of *First Order Logic* (FOL), namely an extension of the description logic $\mathcal{SROIQ}$. The OWL 2 Direct Semantics [BPB12] specify the semantics for the respective OWL profile in a model-theoretic way. With *logic reasoning* new inferences can be gained based on the specified semantics, i.e., new ontology knowledge can be generated out of existing one based on ontology reasoning. This aspect will be further discussed in Subsection 2.5.3. As it will be explained there, the close connection to description logics also provides a basis for reasoner implementations. Another aspect of the logic grounding is that the reasoning procedures can also validate ontology definitions for freeness of contradictions.

Ontologies also provide extensible formalizations. This means that the closed-world semantics usually being incorporated in databases are changed to an open-world assumption where new facts can be added or concluded incrementally. Complex reasoning tasks with the aid of additional rules provide the possibility to retrieve important information for automatization tasks and especially other software components. An application of the latter principle will, e.g., be presented in this thesis when analyzing optimization models in Subsection 5.6.1.

As already mentioned, ontologies play an important role in the description of semantic software services and in the vision of the semantic web. The role of ontologies in the description of semantic software services will be discussed in Section 2.6.

Within this thesis ontologies will primarily be used as a representation mechanism for abstract optimization models that extend the established structural and syntactic representations by semantics. The ontology representation which will be introduced in Chapter 5 captures semantics of data, model constituents and their mathematical properties as well as that it allows for the generation of a semantic model service.

In the next subsection ontology languages, especially OWL, will be introduced. Followed by this is the introduction of a graphical representation of ontology excerpts in Subsection 2.5.2 as well as an introduction and review on reasoning, rules and query languages. The last part of this section is devoted to a discussion of ontology mapping and merging.

2.5.1. Ontology Languages and Semantics

Ontologies are to be specified in a certain ontology language and every such language has its own semantics. Different ontology languages exist at current date. In what follows, the two closely related languages RDF(S) and OWL will be introduced.

RDF(S):
The *Resource Description Framework* (RDF) [DMR14] as well as its data-modelling vocabulary *RDF Schema* (RDFS) [BG14] provide an important ontology language for metadata annotation and the vision of the semantic web. As it was already mentioned in Section 2.4, RDF-data is stored in form of triples containing so-called *subjects, predicates* and *objects* such that multiple triples together form a directed RDF-graph. An example for such an RDF-triple would be `ex:florian ex:worksAt ex:DSOR .`, stating that the resource representing the individual `ex:florian` is employed at an institution that is identified by the resource `ex:DSOR`. It is mentioned, that everything contained in this triple is a resource, i.e., even the predicate `ex:worksAt` can be seen as such a resource, and that the resources are identified by IRIs which were abbreviated with the usage of the prefix `@prefix ex :<http://examplenamespace.org/>`. IRI is the abbreviation of "Internationalized Resource Identifier" that extends the character set of the well known concept of URI towards unicode. Besides the usage of resources that are identified by an IRI, RDF also allows the usage of *literals* and *blank nodes*.

By the graph-structure of RDF-data, a high extensibility is given which is a key-feature for its usage in the semantic web. Services, e.g., such ones that are conformant to SADI (see Section 2.4) may answer to HTTP-GET messages with further semantic data in RDF-format that extends a prior knowledge graph. Through the usage of URIs or according to the novel standards better say IRIs, global consistent names and linking can be guaranteed.

RDF itself can be described in different syntaxes, where the standard recommended *RDF 1.1 XML Syntax* [FG14] and the triple-near *Turtle* notation [GE14] are mentioned here.

With RDFS and the vocabulary introduced it is possible to describe schematic ontological knowledge of RDF-data and perform *reasoning* based on the specified semantics of the constructs. Constructs introduced by RDFS include type-and subclass-specifiers, domains, ranges and subproperties. The semantics of RDF and RDFS are specified in the RDF 1.1 Semantics [PP14]. Since RDF is fundamental for OWL which will be introduced the next, the important RDF(S) constructs will be futher explained in the context of OWL.

OWL:
In the remainder of this thesis the *OWL 2 Web Ontology Language* (OWL) [W$^+$12] will be used, where the version number two will often be omitted and the common abbreviation OWL will be used. In OWL, a strong distinction between *individuals, classes and properties* that intuitively specify concrete objects, classes and relationships, is made. Besides these three, OWL contains further constructs, e.g., for *data(type) properties, data values* and *annotations.*

As with resources in RDF, OWL 2 makes use of IRIs for the globaly unique naming of *entities.* Since OWL is named as an ontology language of the web, this point is of crucial importance both for conformity with the URI/IRI standard for the identification of web resources as well as the consistency of globally distributed resources. IRIs can be abbreviated via the use of *prefixes.* According to the OWL 2 structural specification [PBB12] entities are "*classes, datatypes, object properties, data properties, annotation properties* and *named individuals*".

The main part of an OWL ontology document is a set of *axioms* that make statements about the included *classes, properties, individuals* and *data values* which are asserted to be true. Furthermore, OWL ontologies can carry annotations, which have no defined semantics but can be used by tools, as well as that ontologies can import other ontologies. Every ontology may have an ontology IRI as well as that there is the possibility to reference different versions of an ontology by version IRIs.

OWL - Syntax:
There are also different syntaxes available for OWL 2. The only syntax that is required to be supported by all tools is RDF/XML [FG14]. By this syntax OWL ontologies are described and stored as RDF documents where the embedding of OWL constructs is done by an imported namespace. To that end this syntax is also often referred to as the *OWL-RDF syntax.* Other syntaxes are OWL/XML [MPPS09] which is a direct serialization of OWL in XML that can be mapped to RDF/XML, and the OWL functional syntax [PBB12] which is mainly used for the structural specification of OWL. A practical syntax for describing OWL documents and parts of it in tools and text is the Manchester Syntax [MP12] for OWL 2, also known in its prior version "Manchester-DL Syntax" for OWL 1. Within this dissertation, the Manchester Syntax will typically be used in combination with a graphical representation of OWL ontology excerpts. The graphical representation of ontologies in OWL will be presented in the next subsection.

OWL - Language Constructs:
Let us now introduce the constructs which are important for this thesis with an intuitive description of their semantics.

As it has already been mentioned, OWL distinguishes *individuals, classes* and *properties.* The separation between individuals and classes can be seen as a separation between the types of instance and class knowledge. Individuals can be used to describe instance knowledge where especially a type in form of a class can be specified. The respective property to be used in an axiom is called `rdf:type` and stems from RDF. Classes can be defined to be subclasses of other classes by the property `rdfs:subClassOf` from RDFS. With the aid of `rdfs:subClassOf` whole subclass hierarchies can be built. The distinction between instance and class level is manifested in the notions of *assertional* and *terminological* knowledge.

Besides the language built-in properties a user can define properties on his own. *Object Properties* as well as *Data Properties* (or *data type* properties) can be defined on the terminological level and used to relate individuals with other individuals or data on the assertional level. The relation given with properties is binary. Properties can have a *domain* and a *range* specified on the terminological level that specifies the allowed types / data types of the related individuals. Further knowledge on class level can be used to define properties of the relations such as symmetry or reflexivity.

Classes can be described in different ways. The most common one is to specify necessary conditions on the members of a class. Such a necessary condition can be given by a *restriction.* Restrictions describe the individuals that participate in a relationship. By this they can describe an *anonymous class* of which a concrete class to be restricted can be a subclass. In this case one speaks of an *object-(property-)* or *class restriction.* Multiple such restrictions can be specified for a class. The restrictions are distinguished as *quantifier restrictions, cardinality restrictions* and *hasValue Restrictions* (see [Hor11b]). The representation approach of this thesis will make extensive use of cardinality restrictions in what follows. For a given property and a class to which the restriction applies, a cardinality restrictions imposes a condition on the number of distinct individuals to be related by the aid of the construchts `min, max, exactly` and a respective nonnegative integral argument.

Let us now introduce a first ontology described in Manchester Syntax to illustrate the constructs mentioned above. The first example ontology provides a vocabulary for network data, i.e., instance data that consists of network arcs and nodes which are related to each other, as well as some annotated parameter values. The ontology provides a vocabulary which makes use of *classes, object properties, data properties* and *datatypes.* Also some restrictions are imposed. The ontology contains no individuals and the definition can be found in Figures 2.5 and 2.6. The specifications in Manchester Syntax introduce the prefix `cdo-network` in order to denote the ontology concepts fore simply. This prefix will also be used in the following textual

```
Prefix: dc: <http://purl.org/dc/elements/1.1/>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf−schema#>
Prefix: cdo−network: <http://www.semanticweb.org/florianstapel/ontologies/2015/5/
    CDO−Network#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2015/5/CDO−Network
    >
Datatype: xsd:double
Datatype: xsd:integer
ObjectProperty: cdo−network:LastNode
    Domain:
        cdo−network:Arc
    Range:
        cdo−network:Node
ObjectProperty: cdo−network:hasSingleCommodityFlow
ObjectProperty: cdo−network:hasSingleCommodityUpperCapacity
ObjectProperty: cdo−network:FirstNode
    Domain:
        cdo−network:Arc
    Range:
        cdo−network:Node
ObjectProperty: cdo−network:hasSingleCommodityCost
DataProperty: cdo−network:FlowValue
    Domain:
        cdo−network:SingleCommodityFlow
    Range:
        xsd:double
```

Figure 2.5.: A network data ontology in Manchester Syntax - Part 1

description, where the separation of prefix and suffix is done by the symbol `"#"` instead of the colon `":"` used in the Manchester Syntax.

A visualisation for OWL ontologies will be discussed in the next subsection. Therefore, a visual representation of the latter ontology can be found in Figure 2.7.

Concrete network arcs and nodes are to be related by the object properties `cdo-network#FirstNode` and `cdo-network#LastNode`. More specific, the domains and ranges of the properties are defined as the classes `cdo-network#Arc` and `cdo-network#Node`. The class `cdo-network#Arc` furthermore carries two cardinality restrictions that intuitively state that every network arc has exactly one first and one

```
DataProperty: cdo−network:SupplyValue
    Domain:
        cdo−network:SingleCommoditySupply
    Range:
        xsd:double
DataProperty: cdo−network:CapacityValue
    Domain:
        cdo−network:SingleCommodityUpperCapacity
    Range:
        xsd:double
DataProperty: cdo−network:CostValue
    Domain:
        cdo−network:SingleCommodityCost
    Range:
        xsd:double
Class: cdo−network:SingleCommoditySupply
    SubClassOf:
        cdo−network:SupplyValue exactly 1 xsd:double
Class: cdo−network:SingleCommodityFlow
    SubClassOf:
        cdo−network:FlowValue exactly 1 xsd:double[>= 0.0]
Class: cdo−network:SingleCommodityCost
    SubClassOf:
        cdo−network:CostValue exactly 1 xsd:double[>= 0.0]
Class: cdo−network:Node
Class: cdo−network:Arc
    SubClassOf:
        cdo−network:hasSingleCommodityCost max 1 cdo−network:
            SingleCommodityCost,
        cdo−network:LastNode exactly 1 cdo−network:Node,
        cdo−network:FirstNode exactly 1 cdo−network:Node,
        cdo−network:hasSingleCommodityUpperCapacity max 1 cdo−network:
            SingleCommodityUpperCapacity,
        cdo−network:hasSingleCommodityFlow max 1 cdo−network:
            SingleCommodityFlow
Class: cdo−network:SingleCommodityUpperCapacity
    SubClassOf:
        cdo−network:CapacityValue exactly 1 xsd:double[>= 0.0]
```

Figure 2.6.: A network data ontology in Manchester Syntax - Part 2

last node.

Further object properties, classes and restrictions in the ontology describe the typical network data conceptualizations such as costs, supplies, capacities and flow values. The respective IRIs carry very specialized names, e.g., `cdo-network#SingleCommodityUpperCapacity`. The principles of modeling in domains such as networks will be further discussed in Chapter 6.

Data properties and respective restictions are also included in the example. The `cdo-network#CapacityValue` at first is a data property that relates individuals of type `cdo-network#SingleCommodityUpperCapacity` to data values of type `xsd:double`. By a restriction on `cdo-network#SingleCommodityUpperCapacity` the value is further forced to be unique and nonnegative.

This subsection gave a short first overview on the language constructs of OWL that are of great importance for the ontologies developed in this thesis. Further constructs such as means to specify classes to be disjoint or equivalent will be introduced in the context of the respective sections making use of them.

OWL - Semantics:

The OWL 2 semantics are specified in two forms. At first, the OWL 2 RDF-Based Semantics [Mic12] provide a complete definition of OWL 2 semantics that is compatible with the semantics of RDF(S). For the sublanguage OWL 2 DL (see next paragraph) there are the OWL 2 Direct Semantics defined in [BPB12]. The direct semantics are defined in a model-theoretical way and extend the semantics of the description logic $\mathcal{SROIQ}$. OWL 2 DL ontologies can be mapped to rdf-graphs and the *Correspondece theorem* [Mic12] states that conclusions made using the DL semantics will stay valid for the rdf-based semantics.

The complete OWL 2 specification can be restricted to sublanguages, e.g., for the purpose of handling the computation complexity of reasoning or keeping implementations simple. Common sublanguages of OWL 2 are called *profiles* and have been defined in the *OWL 2 Web Ontology Language Profiles* [BBI+12]. The profiles from OWL 2 have to be distinguished from the ones named for OWL 1. In OWL 2 the three profiles OWL 2 EL, OWL 2 QL and OWL 2 RL were introduced as different sublanguages that are all more restrictive than the OWL DL profile. As already mentioned above, OWL DL itself is the profile that corresponds to the sublanguage of OWL whose semantics can be seen as an extension of $\mathcal{SROIQ}$ and which are specified in the OWL 2 Direct Semantics. OWL DL is of great importance since it is *decideable* in contrast to OWL 2 Full which corresponds to OWL 2 ontologies that have been mapped to rdf and are interpreted using the rdf-based semantics.

OWL - Software and APIs:

Finally, different tools and APIs are in use for tasks related to the management and manipulation of ontologies. For the OWL ontologies within this thesis, Protégé version 4.3 [Hor11b] is used as an ontology editor. Protégé is a famous open-source ontology editor and provides further support for reasoners, queries, rules and visualization via plugins. Protégé in the versions 4 upwards uses OWL API [Hor11a]. OWL API provides parsers for different OWL syntaxes as well as that it provides different functionality associated with the management and manipulation of OWL ontologies. OWL API was used for the implementation of a demonstrator tool that is presented in Chapter 8.

2.5.2. Visualization of Ontologies within this Thesis

Within this thesis, a visual representation of OWL ontologies and ontology excerpts will be used. As to the best of the author's knowledge there is no recommended standard for representing OWL ontologies visually, and in addition to that, a consistent and simple visual representation of all OWL 2 constructs seems hard to realize, an own visualiation based on a relevant subset of OWL 2 will be used for the examples of this thesis. This visualization will be introduced within this subsection.

The chosen representation is similar to the common visualizations for RDF-graphs (see, e.g., [HKRS08]) with the key difference that the here presented approach will distinguish individuals and classes visually as there also is the clear distincition between classes and individuals in OWL. Classes will be visualized as ellipsoids without a filling but the abbreviated IRI of the class. Individuals in contrast to this will be visualized with filled circles. The latter idea is inspired by an ontology visualization under the reference [Wik16]. Data types will in contrast to classes be visualized by rectangles which will be filled in case of concrete values. The whole structure of the ontology definitions will then be represented as a graph, where the directed arrows represent the properties and a labeling will carry further information.

We will distinguish full and stroke arrows, where the full arrows are reserved for the RDF(S) properties `rdf:type` and `rdfs:subClassOf`. The usage of type- and subclass- properties for the full arrows will become clear from the context of classes and individuals. Stroke arrows will be used for properties that are defined in the respective new or imported vocabularies. In order to describe the respective property that is used, or even to describe more expressive constructs such as restrictions, the stroke arrows will be labeled.

Before the labeling of arrows will be further introduced, a first example of a visual ontology representation will be given. The network data ontology which has been introduced in Manchetster Syntax in Subsection 2.5.1 is now being visualized in Figure 2.7. One can see the typical visualizations of classes, data types, properties and respective restrictions. The visualization omits the IRI prefix `cdo-network` since there is only one ontology in this example. For a visualization of individuals, type specifications and subclass relations in the context of multiple ontologies the reader is referred to the examples given in later chapters and sections, e.g., an ontology excerpt in Figure 5.5 of Section 5.3.

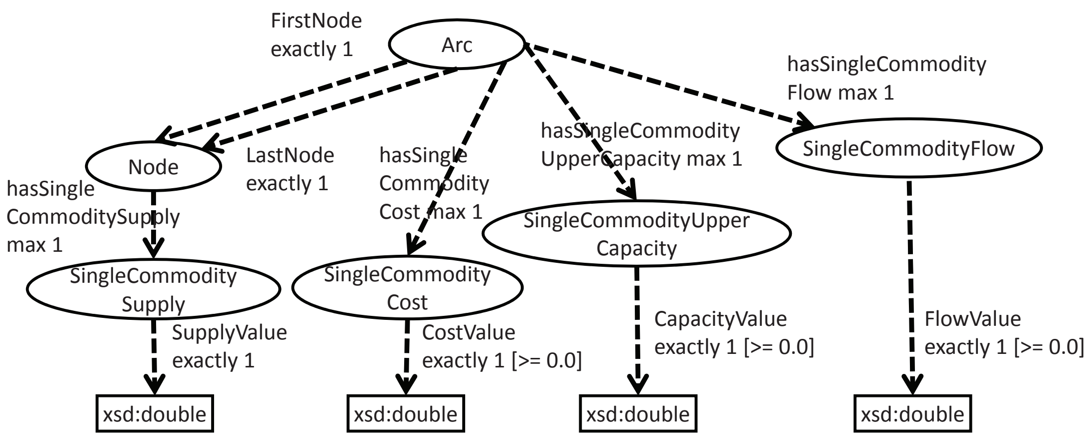

FirstNode exactly 1
Arc
hasSingleCommodity Flow max 1
Node
LastNode exactly 1
hasSingle Commodity Cost max 1
hasSingleCommodity UpperCapacity max 1
SingleCommodityFlow
hasSingle CommoditySupply max 1
SingleCommodityUpper Capacity
SingleCommodity Supply
SingleCommodity Cost
SupplyValue exactly 1
CostValue exactly 1 [>= 0.0]
CapacityValue exactly 1 [>= 0.0]
FlowValue exactly 1 [>= 0.0]
xsd:double
xsd:double
xsd:double
xsd:double

Figure 2.7.: Visualization of the network data ontology

The labeling of a stroke arrow in its most simple form includes an abbreviated IRI for denoting a property resource. In the case of an arrow connecting constructs on the terminological level, e.g., classes and data types, the respective arc can be seen as defining a property with its domain and range. This is not completely formal as a visualization might like to highlight that a property is available between certain classes without refering to the most general superclasses for which the property is defined. To that end, the arrow only specifies entities that are in the case of classes subclasses of the real domains and ranges and the meaning of the arrow and label in this case refers to an object property restriction for the range of values in axioms with the respective subclass as a domain. For the correct specifications the reader is referred to the appendix of this thesis or the accompanying digital material, where the ontologies in use are given with a complete and formal specification in Manchester Syntax. Coming from the property definitions to the assertional level, arcs can be read as triples where the respective axiom for an arc labeled `p` that connects nodes `a` and `b` reads `a p b` in triple-notation.

Restrictions are also to be visualized. Basically, the Manchester Syntax is used to represent restrictions such as cardinality restrictions of the form `p keyword number b` for a class `a` where `keyword` is one of `min, max, exactly` and `number` is a nonnegative integer. The class IRIs `a,b` will be excluded from the statement and be represented by the start and endnodes of the corresponding arrow. An example of such a cardinality restriciton is given for the class `cdo-network:Arc` which reads `cdo-network:LastNode exactly 1 cdo-network:Node` in the Figure 2.7. Data restrictions are also described by excerpts in Manchester Syntax where an example is given in the figure for the class `cdo-network:SingleCommodityUpperCapacity`

which reads `cdo-network:CapacityValue exactly 1 xsd:double[>=0]` .
Further constructs of our OWL ontology visualizations such as, e.g., disjoint classes or subproperties will be introduced when they are used.

2.5.3. Reasoning, Queries and Rules

The development and maintenance of ontologies, as well as their usage in applications, require methods to query and analyze ontologies. In this section, the approaches to *query languages* and *ontology reasoning* that are relevant for this thesis will be discussed. *Ontology reasoning* describes the process of logically deducting new knowledge out of the existing one. OWL(-DL) reasoners can, e.g., be used to check for contradictions both locally and globally, or to compute subclass-hierarchies and direct types of instances. Ontology reasoning is also an important method for debugging ontologies. Query engines can invoke reasoning to answer complex queries in respective query languages.

Reasoning:
Reasoning for OWL ontologies is usually being based on Description Logics (DL) [Baa03]. Different approaches for ontology reasoning exist. Both *Tableau-Based Reasoning* [MH09] or *Resolution-Based Reasoning* [Mot09] are typically introduced for description logics such as $\mathcal{ALC}$ and $\mathcal{ALCHI}$ and then extensions towards the full spectrum of OWL 2 constructs, reflected by the DL $\mathcal{SROIQ}(\mathcal{D})$ take place. Resolution in general is a typical first order theorem proving approach. The tableau-(calculi-) based reasoning currently seems to prevail over the resolution-based reasoning for DL ontologies, at least in the practical reasoner implementations. Both approaches will be shortly discussed in what follows. A comparison of different reasoners for ontologies in the OWL 2 EL profile can be found in [DCT$^+$11].

In order to understand reasoning procedures and the corresponding problems one needs to understand how syntax and semantics of description logics can be defined. A simple description logic that can later be extended towards more powerfull languages that capture the full capabilities of OWL 2 ontologies is the above mentioned *Attributive Language with Full Complement* $\mathcal{ALC}$. In $\mathcal{ALC}$ *concepts* and *roles* are introduced as unary and binary predicates and *individuals* are used to represent specific objects of a domain. Allowed descriptions of *complex concepts* can be built by using *conjunctions, disjunction, negation, existential quantification* and *value restriction.*

The semantics can be defined with the aid of *interpretations* in a *model theoretic* way or alternatively by a translation to first-order logic. An example for the latter approach can, e.g., be looked-up in [Mot09] where knowledge bases in the description logic $\mathcal{ALCHI}$ are mapped to formulas in FOL. Independent of that, in DL one defines the notions *ABoxes* and *TBoxes* that intuitively refer to the assertional and the terminological knowledge in a DL-knowledge base, i.e., the knowledge and axioms about individuals and classes. Different decision problems in reasoning can be defined both for the tableau-calculi and the resolution based approach. All these problems that, e.g., deal with the satisfaction of Tboxes, the deduction of subclass relations (in ontology speaking) and the retrieval of instances can be reduced to a certain form of satisfiability problem. For the case of Tableau-based reasoning the solution of the so-called *Consistency Problem* for $\mathcal{ALC}$ Aboxes is a core component of reasoners whose results are to be used for other reasoning tasks. In the case of resolution-based reasoning one considers the problem of "checking satisfiability of K-, [Mot09]" as a core problem. The latter problem is concerned with the satisfiability of an $\mathcal{ALCHI}$ knowledge-base as a first-order formula.

Reasoners such as FaCT++, HermiT and Pellet [TH06, SMH08, SPG+07] are based on DL tableau/hypertaubleau -reasoning and introduce different optimizations to increase performance as well as extensions to allow for more complex queries. They are currently the most common tools for reasoning with OWL 2 ontologies. Another important reasoner usually refered to as RacerPro is included in the RacerPro Knowledge Representation and Reasoning System [HHMW12]. All the latter reasoners can, e.g., be used with the Protégé ontology editor via plugins.

Resolution-based reasoners seem to be beneficial for the case of large Aboxes (see, e.g., [Mot09]). A reasoner implementation is given within the ontology managing infrastructure KAON2 [Mot06]. The KAON2 reasoner is no longer maintained.

Ontology Queries:
Ontology query languages are typically classified as either RDF-based or DL-based. A recommendation for the case of RDF(S) ontologies is given by SPARQL [W+13], whereas DL queries can, e.g., be performed by ASK queries in the DIG protocol [BMC03]. An interesting approach for DL reasoning that also allows for the mixing of queries on assertional and terminological knowledge is given by SPARQL-DL [SP07]. SPARQL and the distinct subset SPARQL-DL will now be introduced. SPARQL-DL will be used in the design presented within this thesis.

SPARQL [W+13] denotes a recommendation for querying RDF-graphs. A SPARQL query is constructed of so-called *triple patterns* which are rdf-triples in which zero or more variables occur. Multiple triple patterns together yield a *Basic Graph Pattern* (BGP) and finally operators such as `OPTIONAL, UNION` and `FILTER` can be used to construct more complex SPARQL-queries. Answers to SPARQL queries are then variable allocations that in the end yield a subgraph of the queried RDF-graph. An implementation of a SPARQL query engine is, e.g., given by the ARQ engine of the Apache Jena framework [Apa15]. Results of SPARQL (and also SPARQL-DL) queries can be obtained in different formats, where the SPARQL Query Results XML Format [JD13] will be used in what follows.

Typical DL Queries can be used to ask for either direct or general super- and subclasses, equivalent classes and instances of classes. But the main restriction is that the queries are atomic and are either Tbox, Rbox or Abox queries (see [SP07]).

I.e., you may ask for all subclasses of a certain class or all instances of a class separately but not mix queries with both individual and class variables. A flexible approach to overcome this limitation is the above mentioned SPARQL-DL [SP07]. SPARQL-DL has been designed as a distinct subset of SPARQL that allows for the combined treatment of ABox, RBox and TBox queries. Different query atoms such as `Type(a,C)`, `PropertyValue(a,p,v)` or `SubClassOf(C,D)` can be used in conjuntion with a SPARQL-like syntax to formulate queries. An API including a query processor that is based upon OWL API is also available (see `http://www.derivo.de/ressourcen/sparql-dl-api.html` for more information). SPARQL-DL queries will be used and extensive examples of such queries will be given in Chapter 5 of this thesis.

Rules

Logical knowledge that goes beyond the expressive power of the ontology language OWL 2 can be specified in form of rules which are then to be considered in reasoning procedures. A simple and common approach to formulate rules directly upon OWL ontologies is given by the *Semantic Web Rule Language* (SWRL) [HPSB+04]. SWRL provides Horn-like rules that can be added to OWL-DL ontologies. OWL syntaxes such as RDF/XML are extended for specifying OWL ontologies with SWRL rules. The semantics of OWL ontologies including SWRL rules are defined in a model-theoretic way.

SWRL rules consist of an *antecedent* and a *consequent* that can consist of multiple *atoms*. Multiple atoms are then treated as a *conjunction*. Following the definition [HPSB+04], atoms are of the form `C(x)`, `P(x,y)`, `sameAs(x,y)`, `differentFrom(x,y)` or so-called builtIns are used. A "?" character typically marks a variable, e.g., `?x` and variables are treated as universally quantified having their scope limited to a single rule. Different language built-ins can be used, e.g., to perform simple computations with numbers in a rule.

A first example of a SWRL rule also mentioned in the member submission [HPSB+04] is given as follows where a human readable syntax with the usual logic operators is used. This syntax is reflected by implementations, e.g., for the specification of rules in Protégé:

$$parent(?x, ?y) \wedge brother(?y, ?z) \Rightarrow uncle(?x, ?z).$$

The rule intuitively states that a person `?z` that is a the brother of a person `?y`, which is a parent of a child `?x`, is an uncle for `?x`.

Further examples and uses of SWRL rules will be presented in Section 5.6.

By the condition of only using variables in a consequent that are used in the antecedent, that is typically refered to as "safety", the decidability of SWRL extended OWL ontologies can be guaranteed [MSS05].

Reasoner implementations that support the reasoning with SWRL rules are given by

HermiT, Pellet and RacerPro.

2.5.4. Ontology Imports and Ontology Mapping

In typical applications, not only a single ontology is used, but multiple ontologies come together. Different scenarios can be distinguished such as the ontology import, where concepts and axioms from another ontology are loaded into an actual ontology project. The new "importing" ontology can make use of the knowledge contained in the imported ontology in order to formulate new axioms. It is important to notice that ontologies are only referenced and loaded by this step and that a working ontology import requires that no conflicts in the vocabularies being brought together occur. Besides the ontology import, different application scenarios such as the integration of data that is annotated with ontology metadata or the automated composition of semantic software services require different ontologies to be *semantically integrated*. In a typical situation, where two ontologies provide conceptualizations for overlapping excerpts of a domain, differences can occur that, e.g., result from the different views, modeling principles or granularity requirements of the individual ontologies. To solve the heterogeneity problem, the ontologies need to be aligned. *Ontology alignment* can be referred to as the task to both provide an *ontology matching*, i.e., to find similarities in the concepts of ontologies, and to provide an *ontology mapping* that concretely relates the two ontologies. The mapping can consist of simple relations, i.e., subclasses, equivalent classes, disjointness, or also complex transformations. The task of *ontology mapping* provides the basis to automatically integrate or transform data. The terms *ontology alignment, ontology matching* and *ontology mapping* are not clearly disambiguated in the literature. To that end, the latter separation of the notions *ontology matching* and *ontology mapping* will be dropped and the use of the term *ontology mapping* will be favored as the task to find and specify mappings. This notion follows Noy [Noy09]. Other authors prefer the notion of ontology matching (see, e.g., [SE13]).

Ontology Import:
Ontology imports can be either direct or indirect and are typically handled automatically by tools such as Protégé. Ontology `A` imports ontology `B` directly if a respective import is specified for `A`. For a concrete syntax this could, e.g., mean that the ontology IRI of `B` is specified within an `<Import>` tag in the OWL/XML serialization of `A`. If then `B` imports another ontology `C` that is not directly imported by `A`, then `C` is an indirect import of `A`. By following all referenced ontologies recursively, the *imports-closure* of an ontology can be build. Ontology imports are important for the representation approach of this thesis. Different ontologies will be required to specify and process an abstract optimization model and ontologies concerned with the entities of optimization models will use other vocabularies provided by imported ontologies to formulate their axioms.

Ontology Mapping:
As mentioned above, ontology mapping or the associated task of finding an ontology alignment is of great importance for data/ontology integration and emerging applications such as semantic service composition or the answering of queries in the semantic web.

A definition of an ontology alignment can, e.g., be found in Shvaiko and Euzenat [SE13]: "An *alignment* is a set of correspondences between entities belonging to the matched ontologies. Alignments can be of various cardinalities: 1:1 (one-to-one), 1:m (one-to-many), n:1 (many-to-one) or n:m (many-to-many). Given two ontologies, a *correspondence* is a 4-tuple: $< id, e_1, e_2, r >$, such that

- *id* is an identifier for the given correspondence;
- e_1 and e_2 are entities, e.g., classes and properties of the first and the second ontology, respectively;
- r is a relation, e.g., equivalence ($\equiv$), more general ($\sqsupseteq$), disjointness ($\perp$), holding between e_1 and e_2.

[SE13] ". In addition to that, the correspondences can have annotated metadata, especially a *confidence* value that represents the likelihood that a relation holds in a $[0, 1]$-range.

As mentioned above, allowed mapping constructs are, e.g., equivalence relations, super-class and sub-class relations and disjointness. Following Noy [Noy09], also more complex transformation rules are allowed and the possible mapping constructs depend on the chosen representation of mappings.

Methods to discover mappings cover manual, semi-automated and automated approaches. Besides the completely manual approach, Noy [Noy09] distinguishes two basic cases. In the first case mappings are found using a *shared ontology* and in the second case ontologies are compared directly with the aid of different techniques that are, e.g., based on machine learning or graph comparison.

Following Noy [Noy09], shared ontologies can be used in different ways for integrating or mapping ontologies. In the most simple case, semantic interoperability is given directly by reusing so-called *upper ontologies* for ontology development. Examples are given by SUMO [NP01] or DOLCE [GGMO03]. In a more general case, common shared vocabularies are reused with additional infrastructure to construct mappings between two ontologies directly out of mappings to the intermediary and standartized ontologies. An example is given by the *Process Specification Lanugage* (PSL) [Gru03] and the methods proposed by Gruninger and Kopena in [GK05].

Shared ontologies to which the ontologies to be matched conform or refer might not always be available. Different automated approaches that exploit and compare the structure of ontologies exist. Approaches cover a detailed comparison of entity definitions in an OWL subprofile by a similarity measure [EV$^+$04] and approaches such as GLUE [DMDH02] that use multiple learning strategies to especially exploit the information on the instances of ontologies or the taxonomic structure for finding most similar concepts. A recent survey on matching approaches and a discussion of the state of the field can be found in [SE13]. Another standard text is given by [Noy09].

2.6. Semantic Services

Structural specifications of service interfaces such as gained with WSDL basically require a human to read the service specification and further documentations in order to understand what a service does and how it can be integrated with other services. When applications are created by the means of *service composition*, the services are to be selected in a first step and then to be bounded in a static way at design time, i.e., an automated recomposition with new services at runtime can not take place. In addition to that, errors in the execution of the final system due to changed services might not be dealt with automatically. The development of composed services in this context is often characterized as time-consuming and error-prone compared to the possibilities that are envisioned by *semantic services* which are going to be discussed in this section.

In order to deal with the latter issues in terms of automatization, the structural specifications of software services are to be augmented by semantics. Semantic services provide descriptions by which "properties, capabilities, interfaces, and effects are encoded in an unambiguous, machine-understandable form [MSZ01]." With the means to provide these rich descriptions, the vision of generating service compositions automatically out of *semantic service requests* that generally describe "what shall be achieved and not which concrete service has to be executed [KTSW08]" can be treated as an AI Planning task. To conclude, semantic services are introduced to simplify the *discovery, selection, execution* and *composition* of web services.

In order to pronounce their relation to web services and the vision of the semantic web, the notion of semantic web services was established, see, e.g., [MSZ01]. In the semantic web, automated interaction of web services and agents can be seen as an important feature, e.g., to allow for the answering of semantic queries. Today, different visions and usage scenarios for semantically described services exists and therefore the shorter notion of semantic services will be used in what follows. Semantic services, e.g., occur in the context of a semantic service provisioning platform described by Kuropka et al. [KTSW08] or in the recent vision of On-The-Fly Computing which has been introduced in Section 2.4.

An early approach to the semantic markup of web services and the enforcement of automated web service discovery, execution, composition and interoperation was, e.g., given by McIlraith et al. in 2001 [MSZ01]. This approach will be discussed together with novel service composition approaches in Subsection 2.6.2.

In their book on the Semantic Web, Bussler, Kashyap and Moran [BKM08] describe further aspects and usage-scenarios of semantic services. They highlight the role of data, process and selection semantics for semantic web service descriptions and elaborate a clinical use-case. Payne and Lassila [PL04] also highlight the role of semantic web services in the context of the vision of the Semantic Web. The prior vision "to increase automation in processing Web-based information and to improve the interoperability of Web-based information systems [PL04]" may only be achieved by tackling behavioral issues such as the "unarchitected, unanticipated encounters of agents on the Web [PL04]".

The semantic service provisioning approach by Kuropka et al. [KTSW08] that has already been mentioned above further tackles important issues for turning semantic service visions into practice. The authors highlight the aspects of run-time composition and service quality. They review the literature at date (2008) and present own approaches to service matching and automated composition, especially enforcing the late binding and automated re-planning of compositions after failures. Usage scenarios are elaborated and relevant aspects for providing semantic services over a platform, such as service enabling, are discussed.

A very recent approach towards semantic services is followed in the "On-The-Fly Computing" project which has been introduced in Section 2.4. The vision includes many facets in order to realize the provision of individual IT-services out of base-services on global markets, see, e.g., [CRC11]. Semantic service descriptions, service matching and composition play a major role in this approach as well as means for validation and verification. The technical report [APG$^+$14] describes the configuration of a specification language and matchers in the OTF Computing vision, thereby highlighting the broad range of service description aspects for different applications and providers of services. A novel approach to service descriptions, the *Service Specification Language* (SSL), goes along with this.

In the remainder of this section, the issues concerned with semantic service description languages, service composition and matching, as well as mediation will be discussed.

2.6.1. Semantic Service Description Languages

In this subsection, different approaches to the semantic markup of software services will be presented. The view on which descriptional elements may be contained under which terms may differ from approach to approach. Therefore, the major descriptional elements will now be characterized in an abstract fashion.

In similarity to web services, every service that is considered in this section will provide callable operations with certain inputs and outputs. Within the approach of

this dissertation, these services will also be limited to only provide a single operation and not have any effect on a world state. Nonetheless, the latter restrictions would be too restrictive for a review of semantic service description approaches.
Thereforem the possibly multiple operations a service provides are first of all to be described by signature descriptions. The input and output data of service operations is to be augmented by *data semantics*, e.g., by directly using ontological types from or making reference to so called *domain ontologies*. Kuropka et al. [KTSW08] define the latter as follows: "Domain ontologies serve the purpose of defining how a domain is formalized and what situations are hold regarding the content that is to be communicated in a network of web services [KTSW08]."

Method specifications with ontological types can be seen as a part of a service's *interface description*. But an interface description may generally contain more information, e.g., on the order of callable operations and the communication with the service. This is reflected by the following citation from [KTSW08] dealing with the interface concept in WSMO (to be explained in the remainder of this subsection): "A service interface consists of a choreography which describes the interface for the client-service interaction required for service consumption, and an orchestration which describes how the functionality of a Web Service is achieved by aggregating other Web Services [KTSW08]".

The operations a service exhibits have a meaning that is to be defined in a manner more explicit than only providing a signature. To that end, a service may have a *precondition* on its execution, a *postcondition* to specify what holds after the execution of the service, and an *effect* that typically describes global changes in the world's state after the execution of the service. Effects are often exemplified with physical effects, i.e., after the purchase of a book in an online shop the real physical good will be shipped. Preconditions, postconditions and effects are to be formulated in logic. For that purpose, the elements of the ontologies underlying the service descriptions can be treated as logical predicates and then new logical expressions can be formulated. The preconditions, postconditions and effects of services describe capabilities that are to be matched with goal descriptions for a required service in a service composition approach. Here, an overall goal might be reached after multiple service invocations and the capabilities of every individual service influence the possible services to be executed in advance. Pre- and postconditions were exemplified in the context of CBSE in Section 2.1.

The latter descriptional elements correspond to functional properties of the software behind the semantic services. Nonfunctional properties such as costs should in addition to that also be included into semantic service specifications. In the context of this thesis, interesting nonfunctional properties would include the mentioned specifications of usage costs, performance aspects as well as futher issues concerned with the *quality of service*, e.g., response times. Nonetheless, as this would go beyond the scope of this thesis, it was decided to leave nonfunctional properties out of the consideration.

Let us now shortly introduce three relevant approaches to semantic service descriptions. For further approached the reader is referred to the literature, e.g., [KTSW08, BKM08], for a description and discussion of approaches such as OWL-S or SWSF.

WSMO/WSML:
Following de Bruijn et al. [BLPF06], "The Web Service Modeling Language (WSML) is a language for the specification of different aspects of semantic web services. It provides a formal language for the Web Service Modeling Ontology WSMO which is based on well-known logical formalisms, specifying one coherent language framework for the semantic description of Web Services, ... [BLPF06]".

WSMO distinguishes the four top-level elements *Ontologies, Goals, Web Service Descriptions* and *Mediators* and therefore includes further descriptional elements for service discovery, composition and execution than the ones that were mentioned in the introduction above. There are different language profiles of WSML which will not be described here. Ontologies in WSML are described by *concepts, relations, instances, relation instances* and *axioms* plus some allowed nonfunctional properties, e.g., for annotations. Web services in WSML have *capabilites*, i.e., *preconditions, assumptions, postconditions* and *effects* as well as *interfaces*. In addition to that, *nonfunctional properties* can be specified for web services. The authors in [BLPF06] highlight that "WSML has a normative human readable syntax that makes a separation between conceptual and logical syntax, thereby enabling conceptual modeling from the user point-of-view according to a language-independent meta-model (WSMO), while not restricting the expressiveness of the language for the expert user [BLPF06]".

SAWSDL:
Semantic Annotations for WSDL and XML Schema (SAWSDL) [LF07] provides a lightweight approach to semantically describe web services based on WSDL (see Subsection 2.3) and XML Schema. SAWSDL has the status of a W3C recommendation. It is independent of ontology languages, though OWL and RDF are mentioned as typical examples. SAWSDL provides means to augment WSDL description elements by references to semantic concepts from ontologies. To that end, a so-called `modelReference` attribute is defined for the WSDL and XML Schema elements. Two further constructs, namely `liftingSchemaMapping` and `loweringSchemaMapping` are provided in order to specify complex mappings between semantic data and XML. SAWSDL does not include any means to formalize preconditions or postconditions.

SSL:
The *Service Specification Language* (SSL) [APG$^+$14] refers to a novel semantic service approach in the context of the OTF Computing vision. Service specifications in SSL may in principle contain various descriptional elements, where the basic idea

is to provide a core language for service decription that can be adapted for individual service providers in respective domains by means of configuration and to which transformations from the existing service specification languages exist. Key features included in SSL service specifications typically encompass *signatures* with types, inputs, outputs and references to ontologies, *preconditions, postconditions* and *protocols*. Nonfunctional properties and means to describe *hardware services* are further important features.

Service specifications in this thesis:
Within this thesis, only semantic services that provide a single operation will be defined. These services will further be of the type input-output with no effect. I.e., data is passed into the service as an input and a respective computational result for which certain postconditions hold will be returned as an output. To that end, the key descriptional elements for the services are signatures with ontological types, preconditions and postconditions. The services will mainly be brought into sequential execution orders and therefore will comprise very simple workflows with limited branching capabilities. Errors are to be dealt with to a limited amount. This kind of mostly sequential workflow with no "physical effects" may become clear when considering that the focussed application, namely mathematical optimization, is of computational nature. Since the descriptions will be kept that simple, no concrete specification language from the literature needs to be chosen, but rather will visual representations in combination with signatures and logical expressions for pre- and postconditions be given. Nonfunctional properties will not be considered for the service specifications. An example will, e.g., be presented in Chapter 5, Subsection 5.6.2.

2.6.2. Semantic Service Composition

Service compositions are services which have been integrated into a new service with a new functionality. In contrast to component-composition in CBSE, which was discussed in Section 2.1, it is important that the newly generated functionality is provided as a service for the possible users. This means that it should be independent and loosely coupled. Nonetheless, the notions of *composed* services and *atomic* services out of which the first ones are to be generated can be distinguished. Service composition also decribes the task to generate a composition that fulfills certain requirements. Composition approaches can be characterized by the degree of automation and the point in time at which real services are bound. The basic distinction for the latter issue is *design-time* versus *run-time composition*.

A service composition can be seen as a workflow of concretely bound services and related tasks to find concrete services can be considered under the roof of service composition. Nonetheless, in recent approaches, a composition is first of all described and generated in an abstract way. This refers to an abstract description of control-

and dataflow for certain activities. The computational activities considered here can be described semantically by preconditions, postconditions and effects as well as signatures with ontological types. The word abstract then means that the activities do not represent concrete services but rather some placeholders under which different concrete services that *match the description* might be inserted. Service composition from this view is the planning of activities and their ordering such that a goal is reached, an initial state is respected and the plan is valid according to the individual requirements of the activities (or operations), i.e., preconditions, postconditions and signatures. Kuropka et al [KTSW08] describe these compositions as follows: "The elements of a composition are activities that perform a task. The only activities are service interactions. Besides invoking services, a composition can itself be invoked as a service. The resulting composition includes service interactions only on the specification level without a binding to a concrete implementation [KTSW08] ".

Right before or even during execution, service compositions will always be bound to some conrete services. The following citation motivates abstract service compositions and the requirement of automated service composition techniques for the case of dynamic service landscapes, in which services can occur, vanish or change, and where especially new services that are aggregates of others may become available: "If a service changes it might no longer be suitable for the compositions in which it was used previously. Of course, if a service is removed, all service compositions using this service will fail. If service types exist and some form of dynamic binding (without semantics) is in place some of these problems can be solved. However this rarely works with existing Web services based on simple WSDL descriptions because the probability for a full overlapping of functionality and data structures is rather low, [KTSW08]".

By shifting the service composition and the binding of concrete services from design-time to run-time by means of automatization, the actual services landscape can be considered to satisfy a formal request. As already mentioned above, before an abstract composition can be executed, the binding to concrete services has to take place in form of a selection. Besides others, this selection has to respect possible differences in the service desciptions of the service requested by the composition and the services actually provided. The differences may contain signature differences such as parameter numbers, types and operation names as well as differences in preconditions, postconditions, nonfunctional properties and further. If signatures and logical conditions match, nonfunctional properties can be seen as important factors to choose the best implementation. The whole topic of comparing service descriptions of required and provided services is called *service matching* and plays an important role for the enactment of abstract compositions as well as the classical case where a single service (atomic or composed) is searched.

Service matching (or matchmaking) and requests have also been defined in the literature: "Service matchmaking is the task of finding services for a given *user* *request* based on their semantic service specification. A user request describes the requirements of the user toward the service in a similar fashion as the semantic service specification describes the functionality of a service [KW08]".

Service Composition Approaches:
An early approach to semantic services and semantic service composition was given by McIlraith et al. in 2001 [MSZ01]. The authors present a semantic markup based on the DAML family with a focus on the following descriptional elements: "Fundamental to having computer programs or agents implement reliable, large-scale interoperation of Web services is the need to make such services computer interpretable - to create a Semantic Web of services whose properties, capabilities, interfaces, and effects are encoded in an unambiguous, machine-understandable form [MSZ01]". The authors present a composition approach by model-based programming using the situation calculus and GonGolog, a variant of Golog. *Generic procedures* that are abstracted from the concrete services are instantiated to concrete sequences of service calls by means of logical deduction. The decuction process is based on the knowledge of the user constraints and services as well as further parameters and hence leads to different answers for individual user requests.

The customization of generic procedures (or templates) according to user constraints and available services requires a suited generic procedure to be available and selected for an individual user request. In the context of their holistic semantic service provisioning approach, Kuropka et al. [KTSW08], like other authors at that date treat service composition as a planning problem without the need for this kind of guidance for finding a control and data flow as well as the services to invoke. To that end, the generation of abstract service compositions as well as the consideration of run-time composition are central features of the approach. The authors treat the following four requirements on automated service composition not being referenced in the service composition literature at that date: *Parallel control flow, uncertainty in initial state and service effects, alternative control flow, creation of new variables.* The first requirement relaxes the assumption of total ordering of service invocations to the possibilities of a parallel workflow and the second requirement allows a more realistic treatment of services with multiple possible effects. Based on the uncertain states, different paths of service executions might be required which lead to the third requirement and then requirement four deals with the problem that in planning it is often assumed that all variables are defined in advance. As the authors mention, this requires knowledge of the service landscape and so they try to deal with the issue in their approach. For the compositions an extended version of Enforced Hill Climbing is used.

In [MB13] Mohr and Kleine Büning also tackle the problem of alternative control flows for efficient composition in practice by complementing existing composition approaches with the possibility to include composite components that can be generated before the composition step as a preprocessing task.

Service Composition in this thesis:
The framework presented within this thesis incorporates service composition by the aid of templates. Templates can be compared to the generic procedures that were mentioned above and provide placeholders for services and a rough concept of a control and data-flow including xor-branches. The composition with these templates then reduces to the customization and instantiation of the template according to the given knowledge of user constraints and available services. As template composition includes instantiation, the result of this composition task will contain concretely bound services. The instantiation of a template in the case of this thesis will also be guided by fixing some of the services, e.g., a model instantiation service in a first step and allowing to leave service placeholders for data mediation uninstantiated in the actual composition step. The data transformations are to be implemented before the execution of the system instead of being part of a search procedure. Means to create such transformations for data semi-automatically will be described in the next subsection.

2.6.3. Data Mediation in Sequential Workflows

In this subsection, a special issue concerned with inconsistencies in semantic service descriptions will be discussed. Inconsistencies can occur in different forms such as varying operation names or granularities of service operations, e.g., when two operations of one service might be combined by a sequential execution to yield the functionality of a single operation of another service. Of special interest for the services that are applied in this thesis are parameter inconsistencies, where two comparable operations with the same name have differences in the number and types of parameters. These can, e.g., occur, when the outputs and inputs of two services to be executed in sequence do not match. This setting will further be referred to as *data inconsistencies* where the following assumptions are made: It is assumed that inputs and outputs of services are separated into so-called *ports*, i.e., a port produces or consumes data of a certain structural type, e.g., an XML-message, and two ports of two services to be executed in sequence are connected in the sense that the port-output of the one service is to be passed as an input for the input port of the other service. In this case the problem of inconsistent data may occur and the issue to deal with is the automated generation of a transformation program.

The task described above is known under the general term *mediation*. As descried above, mediation for services might deal with various forms of inconsistencies as there are, e.g., differences in data, processes and protocols (see, e.g., [BKM08]). A similar concept in object-oriented programming is the one of an *adapter* (cf. *adapter pattern*) which also occured in the context of CBSE in Section 2.1. The framework presented within this thesis will make use of data mediation for services and respective programs performing such data transformations will be referred to as *transformation services*.

In [BL04], Bowers and Ludäscher decribe an approach to generate data transformations in scientific workflows semi-automatically out of ontological type definitions for service inputs and outputs. The authors present a framework in which structural types are related to ontology types by *registration mappings* and the specified semantic concepts in the ontologies are mapped by so-called *contextual subpaths*. Together with the subpath-relation and the registration mappings, *correspondence mappings* are defined as a *semantic join* of two registration mapping rules. A correspondence mapping consists of mappings of substructures of the structural types, where the substructures are described by queries. Out of multiple correspondence mappings, the automated generation of data transformation programs is possible under respective assumptions. The authors define general assumptions on the registration mappings, namely *well-formedness, consistency with respect to cardinality constraints* and *partial completeness*, which are at first independent of concrete languages. Consistency in the latter context states that cardinality constraints of structural types imply semantic-type cardinality constraints. Partial completeness according to the registration mappings firstly enforces the registration mapping of a service's output port to be complete in terms of semantic types, i.e., the required concepts for some cardinality constraints of a semantic type have respective structural type instances. Furthermore, the partial completeness requires the data items of the structural types of an input port all to be registered with semantic types in terms of contextual paths. A concrete example using XML data is elaborated and refinements of the assumptions are discussed. The authors mention that partially complete registration mappings that also fulfill respective XML-specific requirements such as a compatibility of cardinality restrictions should allow to generate unique data-transformations. A problem to deal with in practice might be the violation of the partial completeness assumption that leads to *underspecified* correspondence mappings.

3. Model Management and Optimization Software

The aim of this chapter is to introduce the basic notions concerned with optimization models and optimization software in the decision support system context. Literature and technologies will be reviewed and a special focus is laid on the field of model management. The next section starts with a short introduction of basic notions.

3.1. Basic Notions

Mathematical optimization techniques are nowadays being used for a huge variety of applications in science and industry. A common approach is based upon formulating a decision situation at hand as an *optimization model*, where typically abstractions from concrete data are being made in order to allow for the execution of the model with different data-sets. Such an *abstract optimization model* defines *goals and constraints* of the problem in terms of arithmetic expressions and inequalities. *Sets* and *indexing* can be used to separate the model logic from instance data.

Techniques and formalisms within this thesis are often optimization specific. Nonetheless, the more general notion of *decision models* provides the accurate background for discussing the related research and state of the art. Krishnan and Chari [KC00] define a *decision model* as a "formal abstract representation of reality" that "constitutes an important component of decision support systems (DSS) [KC00]". Furthermore, "Such models can be instantiated with data to create *model instances* that represent specific problem situations. Model instances are solved by executable programs known as *solvers* to obtain *model solutions* [KC00]". This definition captures the aspect of abstraction or, say, a schema when a model can be instantiated with different data-sets to obtain computational instances. It also highlights the model as a central component to support *decision making.* Decision models, their instantiation and solution are therefore typically core components of decision support systems. When the model component is restricted to *optimization models*, the term *optimization system* will be used within this thesis. General decision models can, e.g., be mathematical models for simulation, forecasting or optimization, whilst optimization models will be restricted to a definition given below. Within this thesis, the term model, if not stated otherwise, will always correspond to an abstract model without concrete instance data and is hence to be distinguished from a concrete *model instance.*

Optimization models constitute a special form of decision models which consist of *objectives (or goals) and constraints.* The goal is to find a feasible solution to the set of constraints that minimizes or maximizes the *objective*. The scope of optimization in this thesis corresponds to the notion of *mathematical programming* which usually deals with a finite set of real variables to be determined in a solution. Constraints are understood as (in-)equalities of algebraic expressions in the model parameters and variables. Optimization problems are to be *solved* by specialized algorithms. Implementations of such algorithms are usually contained in software components called *solvers* (this is the same as for the general decision model context above).

The basic workflow in optimization can be described as formulating an abstract optimization model based upon a problem description, implementing the abstract model and instantiating it with data, solving the instance and interpreting the results. The process may typically be reiterated where model and data are adapted and further steps such as validation and selection and integration of components such as solvers take place in order to obtain suited decision support for a real world decision situation. The research discipline of *model management* tries to answer different questions concerned with decision models in the different steps of a general *modeling lifecycle*. Therefore, model management deals with different questions of creation, description, analysis, reuse and integration. It will be the topic of the next section. For optimization specific purposes, software solutions to support the modeling lifecycle exist in different forms. A common approach is to formulate an abstract optimization model in a symbolic, algebraic notation in a so-called *Algebraic Modeling Language* (AML). AMLs are typically a part of *Algebraic Modeling Environments*. The latter languages and systems will be a topic of Subsection 3.3.2.

3.2. Model Management

Current and past research on decision models in software systems for decision support can be viewed as part of the field of *model management*. An overview article is, e.g., given by Krishnan and Chari [KC00]. The following subsections will introduce basic concepts and describe the state-of-the art for central areas such as SOA for model management, model representations and model integration.

3.2.1. Modeling Lifecycle and Model Formulation

This subsection is concerned with the *modeling lifecycle* of decision models and the central step of *model creation*. Krishnan and Chari [KC00] define the modeling lifecycle as a process of eight tasks, namely *problem identification, model creation, model implementation, model validation, model solution, model interpretation, model maintenance* and *model versions/security*. Each of the latter tasks has a certain goal and can be achieved by different *mechanisms*. *Problem identification* is the first step that has the goal of determining a "clear, precise problem statement [KC00]" that can be translated into a formal mathematical model in the *model creation* step.

Model creation can be achieved by different mechanisms where the authors define the four mechanisms *formulation, integration, selection and modification* as well as *composition. Model implementation* has the goal of creating a computer executable statement of the model. This can contain both the formulation of an abstract optimization model, e.g., in an AML, as well as its instantiation with data. After *model implementation*, the model can be validated by *model validation* as well as that a computational instance for the *model solution* step shall exist. *Model interpretation* denotes the interpretation and analysis of results. With the last step, a first *decision proposal* for the practical *decision problem* can be obtained. *Model maintenance* denotes the step to revise the problem statement and/or model and reiterate the latter steps. *Model versions/security* is a task to be performed parallel to the latter loop and has the goal to "maintain correct and consistent versions of models [KC00]" and "ensure authority to access [KC00]".

As mentioned above, *model creation* can be achieved by the different mechanisms model *formulation, integration, selection and modification* as well as *model composition.* A commonly known mechanism that is not focussed on reusing existing models or parts is *model formulation.* According to Krishnan and Chari [KC00], "model formulation is the task of converting a precise, problem description into a mathematical model. It is different from the other mechanisms used to create models in its focus on constructing the mathematical structure of the model [KC00]". According to the same reference, questions concerned with model formulation cover the "characterization of the modeling process", "represenational issues" as well as "Reasoning Issues". The modeling process can both be investigated under the process perspective of "determine where model management technology can be of the most use [KC00]", as well as on the sources of knowledge employed by *modelers.* Representational issues do not only deal with representations of model and data, e.g., whether they should be domain specific in problem objects or "in terms of object such as resources and activities that underlie modeling paradigms [KC00]", but also how knowledge about modeling paradigms and solution technology should be represented. Reasoning issues are concerned with the question of "what is the reasoning process used to formulate a model from a qualitative problem description [KC00]".

In [MSM92], Murphy, Stohr and Ma provide composition rules to generate linear programming models (LPs) out of abstract *algebraic pieces* such as terms with summations and right-hand side coefficients with equality/inequality relations. They define the three tasks of constructing models from a complete collection of pieces, the inference of summations from terms and the inference of right-hand side pieces from a complete collection of left-hand side pieces. Rules are proposed for these tasks and an algorithm for composing LPs from pieces and terms is described. The index structure of pieces is identified as a central ingredient for the construction rules and general problems of the algebraic piece construction are solved by extending the indexing by information about form, time and place. In [MMS96], the implementation of the model formulation system LPFORM is presented which implements some of the methods from [MSM92]. The system is graphically oriented and lets the user enter model entities such as sets and parameters as well as the algebraic pieces in window dialogues. The composition techniques from [MSM92] can be applied to a so-called scenario and afterwards a GAMS model ([McC14], see Subsection 3.3.2) can be derived. A graphical interface also allows for the definition of network-based models by entering and connecting entities such as pumps, wells and refineries for problems of the oil industry. The graphical editing of the network generates constrainst such as mass balance equations automatically and allows for *model integration.* The approach is not restricted to network based problems but to the class LP, as the techniques from [MSM92] are LP-restricted.

The approach of Murphy et al. [MSM92] can be characterized as bottom-up in contrast to the approach presented by Binbasioglu in [Bin96]: Abstract representations of equations / constraints are stepwise refined into linear constraints by rule-knowledge that formulates on a so-called *action-resource* view. Following the author, "the action-resource view captures the semantics of linear programming model formulation since it formalizes the choice of decision variables, coefficients, constants and their appropriate configuration [Bin96]". In this form, the idea has some similarities to the abstraction of formulation types and their derivation as it will be presented in Chapter 5. Nonetheless, the reasoning techniques in [Bin96] are restricted to linear models. Furthermore, the approach in [Bin96] configures a small set of equation types into flexible statements such that, e.g., certain coefficients can be used or left-out, whilst the approach presented in this thesis will follow a different paradigm by providing many different types of less flexible syntactic structure that are organized in taxonomies and can be compared by ontology matching. Operations in the formulation of a model are then performed, e.g., by changing the formulation type to a subclass and adapting the related data specifications. By this, a complex and step-wise configuration of a constraint type is avoided.

3.2.2. Structured Modeling

Under the different model representations and frameworks that have been proposed in the model management literature, *Structured Modeling* (SM) [Geo87] has gained a lot of attention until today. Krishnan and Chari [KC00] describe Structured Modeling as "a formally specified notational framework for modeling that was developed to address a variety of model development problems [KC00]." More concrete, Geoffrion states that "Structured Modeling aims to provide a formal mathematical framework and computer-based environment for conceiving, representing, and manipulating a wide variety of models [Geo87]". This subsection aims at giving a short overview on Structured Modeling and its extensions. This will also be of importance for the model management approaches presented in the following subsections.

Structured Modeling (SM) is based upon formulating general decision models in terms of five elements and their interrelationships. These five elements are *primitive and compound entities, attributes, test elements* and *functions*. Whilst the entities and attributes relate to the representation of data (structure), the test and function elements can represent things like mathematical constraints or formulas for computations. The elements and their relationships are to be defined in multiple structures on multiple levels of aggregation. Namely, the *elemental structure*, the *generic structure* and the *modular structure* constitute three levels of representation that are required to specify model instances or represent "abstract" models in form of a model schema. Concrete representation languages that are capable of representing the latter structures will be discussed below. By allowing to specify a broad range of decision models such as probabilistic, simulation or optimization models, the approach is generally applicable in the dss context.

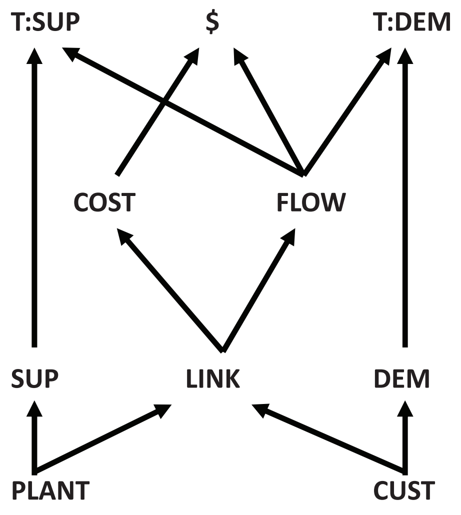


Figure 3.1.: Genus graph of a transportation model. Reproduction of an image from [Geo87]

Figure 3.1 gives an example of a genus graph that can be used to visualize a generic structure. The graph represents definitions for a transportation model. The primitive entities `PLANT` and `CUST` represent plants and customers. The entities can be compared to sets in the AML view (see Subsection 3.3.2) as the elements do not carry any values. The same goes for the compound entity `LINK` representing the links in-between plants and customers. The definitional dependencies in Structured Modeling are visualized in graph or tree views which, besides others, allows for automated model integration approaches that are a topic of Subsection 3.2.3. The latter aspect has also been described by Krishnan and Chari [KC00]: "Another useful feature of structured modeling is its explicit representation of the inter-relationships between model elements [KC00]". Attributes in the example are `SUP,DEM,COST` and

`FLOW` which in order represent the values for supply and demands at the respective plant and customer nodes, as well as the unit-transportation cost parameters and unit flow values for the links. The flow values are variables of the model. This fact can be represented in SM by a *variable attribute* which is a specialization of the usual *attribute*. The model contains an objective and two constraints (`T:SUP` and `T:DEM`) that can be represented by a function and two test elements.

Different languages have been introduced that formalize and extend the Structured Modeling framework introduced by Geoffrion in 1987. Geoffrion proposes SML [Geo92b, Geo92c], a language formalization that is grouped into four levels of complexity. Geoffrion [Geo89] and Tsai [Tsa98] study model integration (Subsection 3.2.3) on SML represented models. Tsai [Tsa98] especially highlights the operations on SML represented models as manipulations to be performed in a text-processing software such as WORD. SML itself is extended towards the Structured Modeling Markup Language SMML for better sharing and reuse in [EGT07]. SMML as an XML language is compliant with current web standards and is used in the model management SOA [EGD13] to be discussed in Subsection 3.2.4. Different languages for SM are typically developed towards a certain application scenario. BLOOMS [GS97] and ASCEND [PMW92] provide two object-oriented approaches. In BLOOMS [GS97], it is possible to define classes for SM elements and use OO-principles such as inheritance for reuse. The system itself is capable of some model integration functionality, see, e.g., [GS95]. Besides the SM-focussed approach of BLOOMS, Piela et al. [PMW92] present the ASCEND language and interactive environment in which five operators for inheritance, typing, refinement, merging and grouping of objects and their members can be used to enforce, e.g., reusage and type checking. They also provide means for checking dimensional consistency. Though object-orientation is exploited by the latter approaches for reuse, there is no broad discussion of design patterns, e.g, for optimization models. As OO focusses its design towards the reuse of functionality it also seems hard to reuse single "constraints" in the latter approaches but rather are bigger software components for models envisioned. The desig approach of this thesis in contrast is intended to reuse detailed type-conceptualizations of model entities in a platform-independent way by exploiting ontologies.

Logics and logical reasoning are central aspects of ontologies but have also been exploited in the past SM literature. In [CK90], Chari and Krishnan present the logic based executable modeling language LSM. LSM represents structured models as a set of FOL-formulas by treating individuals of models as objects that occur in predicates and functions representing SM-concepts. Following an approach from [BK93], the predicates, functions and constants that represent SM constructs such as primitive entities or genus-graph relations are then embedded into the language $L\uparrow$ [BK93] as constants. This then allows for a reasoning on models to conclude or state relations such as a formulation being a linar approximation of another or the evaluation of functions. Checking the integrity of a model by rules or validating units of measurement are further important applications. The inference of mathematical properties such as linearity of a model, or using predefined formulations to perform a linearization, are not a topic of the latter approach but should be mentioned at this point as features of the approach presented within this thesis.

3.2.3. Model Integration and Composition

This subsection is concerned with the two mechansims of *model integration* and *model composition*. Both mechanisms can be used to support the *model creation* step and aim at reusing models or model parts. They are distinguished by the fact that model integration typically tries to create a single consistent model out of parts, whilst model composition, in similarity to the composition of software services (recap. Subsection 2.6.2), tries to bring existing models into an integrated solution process to iteratively solve a bigger problem. Confusion may result from different usage of the terms integration and composition in the literature. Historically, both model integration and model composition are covered under the name model integration ([Geo89]), where the model integration as defined above is specialized to *deep model integration* and model composition is denoted as *functional integration*. Geoffrion defines the term *deep model integration* as the integration task for models in the same representation language with the further requirement of generating a valid model in that language. With the distinction of model integration and model composition this thesis sticks to the view of Krishnan and Chari [KC00]: "Model integration, like model composition, also leverages previously developed models. However, in model integration, the models being integrated are modified [KC00]". Furthermore, the concept of model integration can be refined to schema integration and process (solver) integration: "Schema integration is the task of merging the internal structure of two models to create a new model. Process integration is similar to solver integration in the context of model composition [KC00]". This distinction highlights that integration, in similarity to composition, can generate a process. But for process (solver) integration "the solution process of two or more models may have to be interwoven [KC00]". This subsection will treat model composition as well as model schema integration, where model schema integration is set equivalent to deep model integration or shortly model integration.

Geoffrion has investigated model integration for SM with a manual procedure [Geo89] that is based on the SML language for Structured Modeling described in [Geo92b] and [Geo92a] (recap. subsection 3.2.2). Model integration has been further studied in the model management literature for languages that are based on the Structured Modeling framework. Gagliardi and Spera [GS95] provide procedures and formal results based on genus graphs. Some of the procedures have also been implemented in the scope of their object-oriented SM language BLOOMS [GS97]. The authors define three levels of automation for model integration. Level one integrates two input models (or selections of their genera) fully automatically. Level two, in contrast, requires user-entered dependency replacements ("order of integration among the genera [GS95]"). Level three provides an interface for manual model integration. Results for levels one and two are presented in the reference, where example models are integrated in such a way that previously user-entered parameters are now computed by the resulting model. In case of a transportation model being integrated with an exponential smoothing demand forecast model, a fully automated level one integration can take place, whilst for a four model example the user has to replace dependencies. The example models are typical case-studies in the model integration literature, see, e.g., [Geo89]. An article by Tsai [Tsa98] investigates operations for model integration in the SM framework directly on the SML language provided by Geoffrion. Operations to support model integration are introduced in terms of tree-editing operations for the modular, generic and elemental structures. The author, amongst others, defines operations for projection, concatenation and joining of models as well as that some formal results on their validity are proven. A hierarchy relationship on genus types is defined that explains which genus types can be defined on others. The relation is also used in the join operation to decide which one of two compatible genera should be deleted when merging them. Compatibilty of genera is a further definition given in the article. An illustrative example for a transportation model and a multi-item EOQ model is given. Bhargava et al. [BKK91] introduce a certain aspect into the general framework of model integration by focussing on the question whether object names in a model refer to the same element. To that purpose they postulate the specification of four types of information that especially include dimensional information and a definition of the intended interpretation called *quiddity*.

Model composition denotes a task that uses previously developed models to solve a greater problem, e.g., by selecting and bringing them into an execution order. It contains the task of "linking together independent models such that the output of one model becomes an input of another [KC00]", as well as that it can use techniques from model selection "when no one model meets the requirements of a problem [KC00]". To that end, the task can be seen as similar to (semantic) service composition (recap. Subsection 2.6.2) for general software services, since automated procedures are exploited to determine components, control and data flow to solve an overall task. A special issue for model composition is solver integration since the pure determination of composition of models by their inputs and outputs might not be executable due to different data formats or the missing of a suited solver component that is capable of solving a model.

A literature review on model composition will not be provided at this point, but rather a will a single publication, which treats model composition as a general service composition task and highlights the aspects of distributed model management, be described . Madhusudan [Mad07] introduces a web service framework for distributed model management where the execution of serveral models can be brought into a service plan by AI Planning methods ([MU06]). Models are not only composed due to their inputs and outputs but rather are model services integrated with solution methods and further services, e.g., for querying data. The case-study example highlights an application from engineering design where, e.g., linear programming services and a finite element analysis are parts of the resulting process. As mentioned before, data-services that provide access to external data repositories are also part of the composition. As the models are implemented as web services and a framework architecture with different service providers and repositories is instantiated, model composition for distributed model management is envisioned by the approach. Domain knowledge is modeled in ontologies which are being exploited by the semantic service composition based on HTN planning.

3.2.4. Distributed Model Management and SOA

To a relevant part, model management is concerned with the reuse of models. Mechanisms such as model composition, model integration or model selection are based upon existing models to perform the task of model creation. When model resources are to be shared within organizations or over the internet, architectural considerations and the support for steps such as a semantic search become important. To that end, distributed model management, as it was also leveraged by the model composition approach [Mad07] presented in the last subsection, has gained greater attention in the model management community in the last years.

In [EGD13], such a semantic service-oriented architecture for distributed model management systems that exploits the semantic annotation of decision models is presented. The authors aim "to address problems encountered in sharing and reusing models in a distributed setting [EGD13]". Besides that, the support for model discovery and composition is presented. The suited formalization and processing of semantic information about decision models is provided, where semantic annotations of XML Schema documents, as proposed in the SAWSDL extension of the Web Service Description Language (WSDL) [LF07], come into use. The authors provide a classification of services that are concerned with decision models and model management functionality. Different scenarios, from simply solving a solver instance that is obtained from a repository, to retrieving, manipulating, instantiating and then solving abstract models are possible under the architecture and proposed services to be integrated into a workflow. Parts of the classification can be quoted as follows: "Model execution services are responsible for solving a model. This involves providing the model with a compatible model instance (data), and identifying and invoking the appropriate solver service. The mechanism will differ depending on the underlying model representation. For example, in the case of binary executable models, a model execution service simply runs the model and returns the results of the execution of the model. In the case where proxy model services are involved, the role of the model execution service extends to orchestrating the translation of a model (in case of SMML models) into an executable model format and invocation of a compatible model solver service [EGD13]". The latter classification can also be seen as usefull for concretizing and formalizing the example-based considerations on model services for model and service composition in [Mad07].

But as the view on model semantics and model instantiation functionality in this thesis is fixed to an instantiation and solution workflow, own terms and a workflow will be coined in Section 5.1. This is due to the fact that the term model execution service does not exactly fit the target service architecture of this thesis. For the description and annotation of decision models in [EGD13], the XML-based Structured Modeling Markup Language (SMML) [EGT07] is recommended (recap. Subsection 3.2.2). SMML itself exploits and adapts the Structured Modeling principle introduced by Geoffrion. Besides SMML, further modeling paradigms and representations are possible. In that case, the respective model services' WSDLs are to be semantically described by SAWSDL.

As the last publication highlights, the role of model and model metadata representations is important for sharing and reuse in a distributed environment. In order to find and select models, model discovery and semantic search are further important tasks to be investigated and supported by suited representation languages. Bhrammanee and Wuwongse [BW08] present the ODDM framework for modelbases which uses ontology languages and a formalism called OWL Declarative Description (ODD) to represent and query model semantics in modelbases. A model is being represented in multiple ontologies. The *model ontology* encapsulates the important meta-information for model discovery such as problem classes for, e.g., optimization, important domain concepts included in a model and certain properties such as a restricted graph structure for transportation models. The model itself is represented as an instance of a model schema which again provides knowledge for the model ontology.

The model representation approach in this thesis will differ from the latter representation by a further abstracted structure that allows to efficiently manage model formulation as well as to reuse constraint formulations with different data models. Here, the expression structure of the statements will not have to be represented within the ontology but can be derived from the specifications by predefined reference implementations. Nonetheless, the importance of semantic model representations should have become more clear at this point.

3.3. Optimization Software

The preceding section has presented model management as a research discipline in the field of decision support systems that is concerned with general decision models.

This section shall highlight research and technologies that are more specific to optimization software.

3.3.1. Solver Software

A broad scale of commerical and academic software implementations of algorithms for mathematical programming has been developed until today. The software can be used in different forms such as executable binaries offering a console environment, via an API, or in the cloud. This corresponds to the deployment and reuse of software components as well as the idea to use solver software as a service (, see, also Subsection 3.3.3). Solver software can be classified by the *problem classes* it is capable to solve, as well as the methodology exploited in the algorithms. Solvers for Mixed Integer Programming (MIP) are based upon MIP-technology. They are widely used and provide powerfull tools for solving large scale problems where the perfomance has been tuned over years. Also some extensions for nonlinear problems are typically available for MIP solvers. The relatively new field of Mixed Integer Nonlinear Programming (MINLP) has also produced academic and commercial codes. A novel approach to mathematical programming solvers is given by the LocalSolver [Inn07] that claims to be capable of solving also "highly nonlinear problems [Inn07]" by a hybrid approach that is besides others, based upon a combination of local search and MIP-techniques. Until now, meta-heuristics were typically exploited in problem specific solutions and not in general purpose solvers. Furthermore, the SCIP Optimization Suite [Zus07] provides a branch-and-cut-and-price framework that integrates MIP-techniques and techniques from Constraint Programming (CP) [ABKW08] for solving Constraint Integer Programs (CIP) that can also contain nonlinearities with a restriction on the problem being linear after fixing the integer variables.

Gurobi Optimizer [Gur15] and the solver contained in IBM ILOG CPLEX Optimization Studio [IBM15] are the commmercially most successfull MIP solvers. Both solvers can be used in a service fashion in the cloud. FICO Xpress Optimization Suite [FIC16] contains the FICO Xpress-Optimizer, a further established MIP solver formally known under the name Xpress-MP.

BARON [N. 15] is a MINLP solver that is capable of solving also non-convex problems provably, globally optimal. It is distributed with typical AML Environments (, see, Subsection 3.3.2) and uses a polyhedral branch-and-cut approach to global optimization [MN05]. Bonmin [PLA$^+$08] is capable of solving convex MINLPs to global optimality. Lindo [LIN07] provides a global solver for nonconvex MINLP that is, besides other possibilities, distributed with the modeling language LINGO and an algebraic modeling environment. AIMMS [RB13] provides an outer approximation algorithm implementation to solve convex MINLP.

The LocalSolver [Inn07] that was introduced in the introductory text above provides a novelty by integrating local search techniques on a general purpose level. The capability to apply this to mathematical programming models represented in the proprietary algebraic modeling language LSP on this level of abstraction is of importance for this thesis as it demonstrates the possibility to include different solution approaches into a system architecture that is based upon representing mathematical programming models in AMLs, instantiating them with data and later calling a solver with that instance. Althoug meta-heuristic solution approaches are not explicitely part of the design presented in Chapter 5, these methods could be integrated by obtaining transformations or, as in the case of the LocalSolver, by letting a suited solver do the transformation.

3.3.2. Algebraic Modeling Systems

The task of implementing and instantiating abstract optimization models, with a special focus on data management, is typically performed in and supported by algebraic modeling aystems such as AMPL, GAMS, AIMMS or LINGO [FGK03, McC14, RB13, LIN07]. These systems provide *Algebraic Modeling Languages* (AML) in which models can be formulated in an instance data independent way. The languages are typically declarative, though procedural elements such as loops for iteratively adding a group of constraints to the model have entered to some of them, see, e.g., LINGO [LIN07]. The systems are typically statically or dynamically linked to solver software. The whole modeling system is then distributed as a single product. Valente and Mitra highlight the language approach with the capability to instantiate abstract, symbolic representations with data [VM07]: "These modeling systems, based on Algebraic Modeling Languages (AMLs), enable the definition of models via symbolic algebraic expressions. Algebraic modeling systems interpret the algebraic model and use a given set of data to create model instances in MPS format or equivalent [VM07]". Furthermore, single optimization solvers are sometimes distributed within whole algebraic modeling systems, as it is the case for the FICO Xpress Optimization Suite [FIC16] containing the AML Xpress-Mosel. The language and system LINGO that was already mentioned above is also typically distributed with a proprietary solver.

Algebraic modeling languages have an underlying paradigm that views abstract optimization models as a union of certain constituents (or *modeling components*, see, e.g., [RB13]). The basic constituents are

- Sets: Mathematical sets are used for representing a collection of elements and can be used for indexing purposes. References to concrete elements are avoided by this.
- Parameters: Parameters provide the data-input to the model and can be defined on sets, i.e., indexed by the elements of a set.

- Variables: Variables represent the decisions of a model and can also be indexed by set elements.
- Objectives (or goals): The objective(s) of an abstract optimization model can be represented by algebraic expressions in combination with an indicator of the direction of optimization, e.g., minimization or maximization. The expressions can be symbolic in the way that they use indexing for sets, parameters and variables in order to be independent of concrete instance data values. A typical example is the summation over all elements of a set in a special operator for summation.
- Constraints: The constraints of an abstract optimization model can also be represented in a symbolic fashion as it is the case for the objective(s). By indexing a constraint entity in an AML model, the symbolic formalism can be used to represent a group or set of similar constraints that are translated into a set of rows in the instance-matrix representation of a model instance.

AMPL [FGK03] is an example of a language that clearly adopts this classification by providing keywords for each of the latter five. AIMMS [RB13] allows for the definition of variables to be defined by an expression (i.e. a constraint) that later can be defined as an objective to the model. This fits the classification but may allow for a further decoupled and modular structure that enforces reuse in the model formulation. To that end it should be noted that AIMMS is an example of a modeling environment in which model formulation is not due to purely writing statements in a text editor, but due to a GUI-guided process in which the modeling components are created, manipulated and added iteratively by the user via window dialogues.

As already mentioned above, further typical parts of AML models can be identified, as there can be self-defined functions or procedures as well as expressions for computed parameters, i.e., computations to be performed a priori the solution of an instance. The a priori principle also applies to checks on instance data that can, e.g., in the case of AMPL, be specified in a constraint-like syntax.

The AML modeling paradigm in opposite to Structured Modeling does not foreground the model representation as graphs and trees but rather as a declarative specification, e.g., in form of syntactic statements. Nonetheless, AML environments such as AIMMS also exploit similarities such as the possibility to put the modeling components of a model into modules, cf., the "modular structure" in SM (Subsection 3.2.2). Generally speaking, one can say that the approach of AMLs is more taylored towards mathematical programming models than the general decision model approach of SM.

Further specialized constituents of AMLs are, e.g, initial primal or dual values of variables and constraints. In general, features that have established themselves as usefull for solver implementations are reflected by AML environments and languages that, as already mentioned above, provide a high level access to solvers as integrated components.

Further AML constructs and declarations that are important in the remainder of this thesis will be discussed at the example of AMPL [FGK03]. The usage of AMPL for the examples and demonstrative implementations within this thesis is motivated by the fact that AMPL provides a clear declarative structure that is very near to mathematical notations. In addition to that, it is wiedely used in the academic area. The language authors describe some aspects of AMPL as follows [FGK03]: "AMPL is notable for the similarity of its arithmetic expressions to customary algebraic notation, and for the generality and power of its set and subscripting expressions [FGK03]". Sets can be declared in AMPL by the keyword `set` and a respective *set-expression*. There are different constructs to define, e.g., *ordered sets* or *sets of sets* which will not be explained here. The most important feature of sets in AMPL is their usage via *indexing expressions*. An indexing expression can be used to define and/or reference a set with the additional possibility to define so-called *dummy indices* for referencing anonymus set members. Dummy indices and indexing expressions in AMPL are valid "only in the scope of the defining indexing expression [FGK03]". A standard application would be the definition of a group of constraints in one statement, where a constraint will be generated for every member of the indexed set. With the definition of dummy indices, indexing can be used for referencing the i. parameter/variable value for a set-indexed collection of parameters/variables in a summation. The following example of a constraint from a diet problem stems from the AMPL Book and shall illustrate the latter constructs:

```
               subject to Diet_Min { i in MINREQ } :
sum { j in FOOD } amt[i,j] * Buy[j] >= n_min[i];
```
(3.1)

Indexing sets are `MINREQ` and `FOOD` with dummy indices `i` and `j`. Whilst `i` is used in the outer indexing expression to define a constraint for every member of set `MINREQ` it is also used in the symbolic constraint expression for referencing the i. parameter value of `n_min`. The dummy indice `j` is used for the local sum operator. AMPL provides language constructs for a variety of mathematical functions such as `abs(x),exp(x),cos(x)` and the logarithm function for different bases. The natural logarithm is, e.g., referenced by the keyword `log(x)` and the common logarithm to the basis ten by the keyword `log10(x)`. Another usefull feature in the context of sets are set operations like intersections and unions. The union of two sets can, e.g. be specified by the binary operation `union` in infix notation. Besides arithmetical expressions, *logical expression* can be used to formulate conditions. Besides the usage for defining computed parameters these can also be used to specify logical modeling situations in a compact form. These situations, for which mathematical formulation techniques exist in the literature, are then translated, e.g., into MIP constraints in an automated fashion by AMPL. An example for a piecewise defined function for which the defining arithmetic expression is case-distinguished over the intervals of the domain will be given in Section 7.3.

Model Formulation was introduced as a mechanism for model creation in Subsection 3.2.1. In the case of AML models for optimization, model formulation can be characterized as a process in which data modeling preceeds the definitions of the objective and constraints. To that end, sets, parameters and variables may be declared, basically in that order since parameters and variables provide data values that are set-indexed and variables can be seen as special cases of value-bearing data elements (this is similar to the view of Strutured Modeling). Nonetheless, necessary adaptions may destroy the strict sequential order. Afterwards, the formulation specific parts such as the objective and the constraints are defined, where the former data conceptualizations are being used. Surely the view to create sets, parameters, variables, goals and constraints in that order is idealistic. As already mentioned, especially in later steps, adaptions of previously definied elements may take place when new insights into the problem arise. Nonetheless, it is not possible in the context of the abstract and symbolic view of AMLs to formulate model constraints and data for abstract models without a prior definition of the required data entities. The process of model formulation may vary from AML Environment to AML Environment. As already mentioned above, AIMMS allows for a graphical manipulation of model constituents, whilst AMPL resides on text editing capabilites. Syntax and static semantics are checked for every model before it is instantiated with data.

3.3.3. Distributed Optimization, Frameworks and SOA

In similarity to the approaches for distributed model management and SOA that have developed around decision models and decision support systems (recap. Subsection 3.2.4), distribution, service-orientation and reusage aspects also play an important role for specific optimization technology. This subsection aims at presenting approaches that allow for the remote access to optimization related decision technology, architectural consideration for related systems as well as aspects of reuse for optimization models and implemented optimization specific functionality.

In Section 2.2, the NEOS Server [CMM98] for numerical optimization was mentioned as a an examle for providing software as a service. The NEOS Server (`www.neos-server.org/neos/`) provides access to over sixty solvers. It can be accessed via a web-interface, the Kestrel [DFG$^+$08] interface for the modeling systems AMPL and GAMS, as well as XML-RPC ("NEOS API"). Also in Section 2.2, the article [VM07] by Valente and Mitra that discusses the paradigm shift from ASP to e-Services in the provisioning of, especially optimization related, software as a service, was discussed.

In the same section, the two projects *Optimization Services* (OS) [FMM10a] and *Open Optimization Framework* (OOF) [EM03] that both provide standards for realizing optimization solvers and modeling components as web services, were presented. Whilst OS was characterized to have a solver-centric view by defining XML-languages for optimization related data, communication, discovery and registration, OOF was characterized to also lay a focus on algebraic modeling languages by providing an XML-based AML. In OS, specialized protocols for representing optimization instances, results and solver options, as well as for communications, discovery and registry of optimization related software are defined and allow for the implementation of a web service architecture. OOF also provides standards for implementing a web service architecture in the optimization domain and highlights the modelbase aspect from model management by exploiting the idea of providing models in repositories. To that purpose, OOF provides the aforementioned XML-based algebraic modeling language called AML that can also represent abstract optimization models. So-called AML Transformers can then instantiate models with concrete data. The language serves as a core language into which or from which other model representations can be transformed and that again provides a standard for the communication with solver services.

Whilst typical AML environments are monolithic systems, the Python Optimization Modeling Objects (Pyomo) software [HWW11, HLWW12] provides a modeling language and funcionality which is embedded into the programming language Python. This allows to reuse software libraries when developing own models, solution methods or whole decision support systems. Object-oriented principles are being employed within the model formulation process. The ability to implement own functionality around and inside the model instantiation and solution process is of great importance. Methods such as Benders Decomposition and its variants [Ben62] are hybrid in the sense that they decompose models into a master and several subproblems and then the instances of models are changed iteratively, e.g., by adding cuts. An implementation of hybrid methods is usually problem specific and requires programming capabilities and communication between modeling and solver components. In AML environments, this can only be done by using propriertary scripting languages in cooperation with the available solver software. In contrast to that, a framework as Pyomo is customizable and extensible, and a rich set of features from the programming language itself can be reused together with external packages that employ the principles of object-oriented design.

3.4. Conclusion

This section gives a short conclusion of this chapter. According to the design science research methodology [PTRC07], the next chapter will present the *problem identification and motivation* as well the *requirements* describing the *objectives of a solution*. The conclusion in this section can be seen as a preparation for the latter tasks, helping to identify open research questions.

Chapter 2 has introduced ontologies as a technology to represent the semantics of data and services for different purposes as there were, e.g., automated semantic service composition and data mediation. The methods and technologies were presented as a general purpose methodology for software engineering based on reuse with a special focus towards service orientation. Basing upon the techniques of Chapter 2, Chapter 3 has laid the focus on the decision support and optimization domains where the role of optimization models in the development of optimization-based decision support systems was of central importance.

The research discipline of *model management* has been introduced together with the *modeling lifecycle*, both providing the framework for discussing the central tasks and mechanisms concerned with decision models in the development and maintenance of dss-applications. Besides the discussion of the different mechanisms, Structured Modeling (SM) was presented as an established framework to support the various tasks in the modeling lifecycle. The object-oriented extensions of SM allowed for the reusage of models as modules, related functionality and model integration. The presented approaches for model integration aimed at an automated integration of models in the same representation language into a new whole. Gagliardi and Spera [GS95] have classified the levels of automation and gave examples for working integrations of models where previously user provided data inputs are parts of the computation in the final model. The resolution of an order decision for integration was a central point making fully automated model integration impossible. Concluding the different SM representations, the logical approach of the language LSM provided means to apply logical inference on model formalizations. Besides model integration, techniques for supporting the model formulation itself were presented. The bottom-up approach partially implemented in the system LPFORM [MMS96] allowed for the automated integration of algebraic pieces to LP models. The pieces had to be supplied by the user and further annotated by semantic information about form, time and place. Opposite to that, the top-down approach [Bin96] allowed for the configuration of abstract constraint types in the domain of linear problems, typically from the class LP. For the configuration steps, semantics in form of action-resource graphs had to be supplied. Dealing with aspects of model selection and reuse in a distributed environment, different frameworks and architectures were presented. The approach of El-Gayar and Deokar [EGD13] integrated different use cases of reusing models and applied semantic annotations to the elements. The approach of Bhrammanee and Wuwongse [BW08] introduced an ontological representation in multiple ontologies where the core model was represented on a statement-structural level.

Concerning optimization specific approaches, Algebraic Modeling Languages (AML) were introduced as mainly declarative languages separating model structure from data by allowing for a symbolic representation of sets, parameters, variables, goals and constraints in statements and algebraic expressions. Though some environments provide means to structure the modeling components into modules, no further abstraction, e.g., in form of configurable types of constraints is made in AMLs. A broad range of optimization solvers is available nowadays and can also be accessed remotely by different technologies such as XML-RPC or web service standards, providing practical means for SOA implementations.

The conclusion of this chapter can be drawn as follows: The creation of decision models is a central task in the modeling lifecycle. Besides the adequate representation of a practical problem situation, models need to be integrated with data and other components such as solvers in order to build a system. Research questions, besides others, cover architectures and representations that allow for suited reuse, editing and integration of decision models. Semantic representations of optimization models have been applied for annotating, searching and reusing models as a whole. First semantic SOAs for model-oriented decision support systems are available as well as technology to manage symbolic models in AMLs and to invoke solver software. The word reuse of models or model parts can cover different aspects. Model integration and model composition are approaches to model creation that reuse models as a whole to either generate an new consistent model or bring existing models into an execution process. Model integration can be performed in an automated fashion for merging model structure based on inputs and outputs, where order conflicts may have to be resolved manually. Besides the latter approaches, different fragments can be used to support the mechanism of model formulation. A model formulation approach that is based on a partially automated configuration of single constraint placeholders is proposed with the the action-resource view [Bin96]. Besides the limitation to linear models, the type configuration process of the latter approach seems to be complex. Formalizations are not available in a complete and detailed fashion and there might be limitations in the portability of specifications due to the missing of universally accepted semantics (,i.e., ontologies). In contrast to the top-down constraint configuration approach, there are also means for puzzling and integrating models out of pieces as, e.g., summations. The latter work in an automated fashion up to the point where a user has to enter additional semantic specifications.

Upon the latter observations, different research questions can be derived. This thesis is concerned with the support of the model formulation process as well as the automated generation of a system based upon a model. The idea behind this is a suited semantic representation of reusable model formulation parts (,i.e., constraint types). With this view in mind, a research gap is identified by the fact that the only known constraint configuration approach is limited to linear problems and has no formalized, portable semantic knowledge available on the one hand, and that model integration has a focus on syntactically merging the input and output interfaces of models without basing upon semantic representations of model knowledge. This limits the capabilities for automated manipulations of models and a highly automated generation of systems. Steps such as analyzing models for generating semantic service descriptions, service compositon and data mediation should be automated and integrated into a development process, where minimal effort is required to specify models, data and components for obtaining a runnable system. A special focus is laid on the model formulation step that should require minimum effort for specifications based upon the resuable formulation types. As Structured Modeling and AMLs do not provide such an abstraction and the semantics inherent in extensions of SM have a focus that is not laid on the model formulation and model/data mediation tasks, a new optimization specific representation is required.

4. Problem Identification and Objectives

The preceding chapters introduced notions and technologies both from the general computer science disciplines of ontologies and semantic software services as well as from research on optimization related software, model management and algebraic modeling languages. Furthermore, a research gap covering aspects of model formulation and the development of optimization systems was identified. This chapter will introduce the scope of this thesis as follows: In Section 4.1 the problem definition will be given. Then Section 4.2 will state the objectives for a solution in form of requirements.

4.1. Problem Identification and Motivation

This section shall give a definition of the research problem which this thesis is concerned with. The problem definition will be presented as part of a broader vision that will be further discussed in the outlook part of Chapter 9. Before the problem definition will be given, some motivational use cases shall illustrate the practical use of a solution:

1. *Formalized choices in modeling*: Consider the situation where an optimization model is to be formulated for decisions on a technical system underlying physical laws such as, e.g., the pipe headloss in pipe networks for transporting water. Constraints have to be added that represent nonlinear curves. Choices in modeling may arise. Using the standard technical formulas may introduce nonlinear curves that are in addition to that not everywhere twice continuously differentiable. This may rule out solvers which have the differentiability as a requirement. To that end, a modified, smoothed variant of the formula might be computed and inserted for the model formulation instead, allowing more nonlinear solvers to be chosen. Besides the latter, further possibilites for modeling the pipe headloss exist. As linear MIP technology has proven its practicability on a large scale, other modelers may decide to linearize the formula, thereby introducing a certain modeling error and a huge number of binaries. For the linearized formulation, further decisions have to be made. E.g., an L01 or a SOS2 linearization might be chosen, yielding polyhedral formulations of different quality and allowing for different support in solver implementations. This use case enforces the definition of formalized modeling knowledge in order to allow for explicit, possibly automatizable choices as well as a widely automatized execution of the required modeling operations when

the choice for a certain "formulation" is made. To conclude, the modeling decisions for formulations should be formalized in a way that it is simply possible to choose and later change a formulation for the respective technical curve by simple type definitions, mainly requiring to choose the right "formulation type". All further adaptions in the model structure and data, as well as the system composition, i.e., finding a suited solver, should be automated where possible. The formulations are to be chosen and reused, i.e., the work of writing down the technical formula and introducing variables and further constraints for its linearization can be seen as redundant and should not be required to be performed by a modeler in a repeated fashion. The adaptions to be performed when changing the reusable formulation type in a model specification should also be automatized as far as possible. Automatization of the choice of the formulation type itself in an iterated system generation loop can be seen as part of a future vision that extends this use case. A first step towards the latter is to provide the formalized formulation knowledge, e.g., in form of a subsumption hierarchy or a tree. After choosing a formulation and a solver and solving an instance of the model, the solution of the linearized model might have to be evaluated for its practical feasibility. This might, besides others, be due to the error introduced by a linearization. Therefore, a broader vision extending this use case would be the fully automated generation and modification of systems and models in a loop where a formalized error measure can be used to evaluate different solution scenarios based on the modeling decisions. The modeling decisions might also be influenced by instance data.

2. *Recommended adaptions in model formulation*: The previous use case highlighted the support for a choice in the model formulation step where the decision to add a certain constraint, independent of its concrete form, already took place. In contrast to that, domain typical formulation elements or extensions of existing formulations towards more general situations could be suggested automatically for a model in a certain state. This mechanism could be based upon a matching of constraint types and their interfaces. Consider, e.g., a network flow model consisting of network typical balance constraints and capacity restrictions for a single flow good. The model is known from the literature and an implementation can be retrieved from a library. Now for a practical modeling situation, extensions have to be considered. First of all, there are multiple flow goods to be treated. These different flow goods, usually being referred to as commodities, have to be introduced by a set and then adaptions in the definition of variable collections, parameters, constraints and also the objective of the optimization model have to take place. A user could manually enter a definition of a commodity set and then the system could suggest different adaptions based upon the formalized type knowledge. This would first of all concern the constraint types inherent in the model for which

ontologically similar types from the same domain that also contain the feature "commodity set" could be suggested. Adaptions in this case could, e.g., contain changes of constraint statements to more general situations, e.g., an additional commodity index for the network balance constraints, or an adaption of capacity restrictions towards a more general "bundle constraint" that captures all the different commodities and the parts of the capacity they occupy in a sum. To conclude, the suggested formulations could be used instead of the existing ones. Furthermore, domain typical formulations that contain the "commodities" and are not included in the model in a related way could be suggested to be added to the model afterwards. In the example, additional capacity restrictions for the individual commodities, basing upon individual commodity parameters, might be required. To conclude, a matching mechanism based upon the type interfaces, i.e., ontological class restrictions, has to be developed that provides the technology behind the recommendations in the model formulation process. The recommendation has to consider the current state of the model and optionally the last modification made in the data model as, e.g., the addition of a set.

3. *Generating a system for the model in a highly automated fashion*: The latter model formulation use cases included the scenario of a highly automated system generation. This setting itself is a further use case containing steps such as an automated analysis of the model in order to generate a semantic service description, the composition of the system itself and the generation of data transformations for data mediation. In order to automatize these steps, semantic representations of the model should be introduced, capturing semantics of the modeling components (both the data and the formulation components representing algebraic statements) and the properties of a model and its parts, e.g., linearity of a formulation or integrality requirements.

The latter motivational use cases outline a framework for an optimization system development or better say "generation" with certain capabilities. The problem that this thesis is concerned with can now be stated as follows:

Design a framework that allows for the manipulation of optimization models and the generation of a runnable system both in a highly automated fashion. A model is to be understood as a central component to be instantiated with data and solved by a suited solution algorithm. The framework should contain a suited representation formalism for optimization models that allows to represent semantics of the individual modeling components, data as well as the mathematical properties required to determine a suited solution strategy. The representation should furthermore formalize reusable modeling fragments (abstracted types) and organize them, such that steps in model formulation can be automated and editing operations are simple. Mathematical programming problems up to class MINLP should be supported.

The latter problem definition lays a focus on a single optimization model to be manipulated and integrated into an automatically composed system. This restricts the considerations of this thesis to a manageable scope. Nonetheless, the restricted framework outlined by the problem definition can be seen as a part of a greater vision that will be discussed further in Chapter 9.

4.2. Requirements

To conclude the latter considerations, requirements on the outlined framework should be formulated. These requirements can be stated as follows:

- **Model Formulation by abstract Types**: The creation and manipulation of an abstract optimization model should be based upon a simple process of adding or manipulating modeling components such as sets and constraints and relating them with others. In order to enforce reusage and simplify the process, the modeling components, also called *model entities* should be instances of abstract types whose type knowledge and relation to other types is represented adequately. In summary this should allow for reusing abstractions, e.g., for special constraints or constraint groups together with their implementations, i.e., the abstract structure of an AML formulation, in a simple way. Simple also means that single type manipulations on a model, e.g., when performing model integration or changing the formulation of a constraint after an execution, should intitiate an automated procedure to restore the consistency of the model as a whole and only the absolutely necessary specification in this process should be performed by a user. To conclude, a new representation of optimization models is required.

- **Model Semantics**: For different applications of the approach, different types of model semantics will need to be expressed. To that end, the optimization model representation formalism as stated in the first requirement should be able to represent semantics of *data conceptualizations, goal- and constraint formulation types* and *mathematical model properties* such as linearity.

- **Automated Modifications and Recomposition**: It should be possible for a machine to manipulate models in a suited manner. Especially the semantics of the modeling components may besides the simple and hence machine-manipulable structure of the representation support the automated choice of suited reformulations such as constraint smoothings or linearizations. After manipulations, an automated process to restore the model's integrity is required, as it has already been discussed in the first requirement. It is clear that a changed model may also require a new system composition. The step of changing the whole system composition after changes in the model should to that end also be highly automated. An automated evaluation of a system and the solution after a first execution based upon a quality measure is not part of the requirements but of a broader vision. To that end, a basis for automated model manipulations should be laid.

- **Models as Services and Composition**: It should be possible to wrap optimization models as semantic web services in a straight-forward fashion. Besides their description, the services should provide the functionality to instantiate the model with concrete data. The service description should support semantic service composition, e.g., by including model property knowledge in the composition process for the matching of model and solver.
- **System Integration**: The DSS that is being composed "in theory" by a composition algorithm should be executable in practice. This requires a target architecture and a process in which the models can be integrated with the systems as services and in which data interfaces can be brought to a match, e.g., by automatically generating data transformations for mediation. The composition and data-mediation steps are based upon the semantics inherent in the model representation.
- **Service Enabling**: It should be possible to integrate model representations in AMLs into the framework by providing transformations to the new representation in two directions. To that end, means to reengineer AML models into the optimization model representation to be developed should in principle be considered as a step before the generation of semantic model services which itself should be highly automated. On the other hand, models in the formalism will themselves have to be transformed into AML representations automatically. As the new formalism itself is based upon abstract types, so-called *AML Derivation* functionality should accompany the types and allow to encapsulate the statement structure of modeling component definitions and especially constraint types. Furthermore, it would also be practical to reuse the functionality for instantiating AML models with data, i.e., use an AML tool such as AMPL as a service that is a part of the composition for a model service.
- **Compatible with (semantic) Web Technologies**: Technologies such as OWL and XML should be used in the architecture and design in order to allow for extensions of the framework towards the integration of arbitrary semantic services.

5. Ontology-based Representation of Optimization Models

In this chapter, the design of an ontology-based representation of optimization models will be presented together with the techniques to keep such ontology represented models as a layer in-between AMLs and services. Before the representation is discussed, the general framework for generating optimization systems out of models and services will be explained in Section 5.1. At a certain point in the process, models are treated as services and the ontology representation of abstract optimization models provides the important technology behind. The ontology structure will be introduced in Subsection 5.3.1 and the respective vocabularies will be explained in further detail in the subsections that follow thereupon. After an illustrative specification example in Subsection 5.3.5, the techniques to realize ontology-represented optimization models as a layer in-between AMLs and model instantiation services will be discussed. Examples for the derivation of AML models out of the ontology representation that make use of abstracted and reusable formulation types will be given and a process for transforming a whole ontology-represented abstract optimization model into an AML representation will be defined. The last section of this chapter is concerned with the generation and usage of semantic model instantiation services for ontology represented models.

5.1. Basic Approach

The basic approach for generating optimization systems out of optimization models is visualized in Figure 5.1. A process is described that leads to an executable optimization system with an abstract optimization model as a core component. An iterated and automated treatment of changes, leading to a recomposition of the system, is included. In a first step, an initial abstract optimization model is formulated in a format suited for a human modeler. The choice is between the usage of an AML such as AMPL, GAMS, AIMMS, LINGO or XPress-Mosel, or the ontology-representation to be introduced within this chapter. When the model is formulated in an AML, AML Reengineering, which is outlined shortly in Section 5.5, can be used to obtain an ontology formulation. The discussion of the model formulation process in the ontology representation will be part of Chapter 6.
With a formulation of an abstract optimization model and further input describing the instance data it might be possible to estimate the complexity of solving the mathematical programming model. In addition to that, instance data itself could be

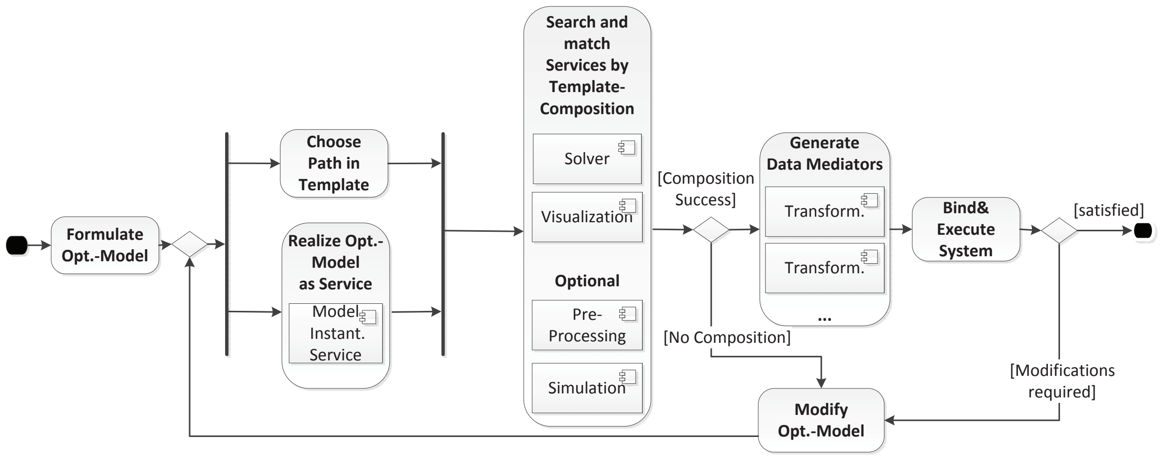


Figure 5.1.: Generation of optimization systems for target SOA with changes

considered for choosing a suited model formulation. With the knowledge about the instance data and model, a choice of the services to be included in a system composition can be made. I.e., for network models it might be useful to have a computational step that uses preprocessing of network data to simplify instances, or apply a simulation engine on mathematical programming solutions that allows to investigate the applicability of the optimization solutions in more detail. If no knowledge about instance data is available, a general strategy including all computational services in the architecture, i.e., preprocessing, solver and simulation, can be followed.

The latter step is represented in Figure 5.1 by choosing a template-path. Corresponding to considerations in Subsection 2.6.2, the service composition task will be treated as a special form of *template composition*. This means that the general task of service composition is reduced to suitably instantiating the template which in turn means the determination of *inputs, outputs, preconditions and postconditions* by inserting concretely bound services into a placeholder. Choosing a path in such templates in our case leads to determining a unique sequential workflow of services, e.g., preprocessor, data transformation, model instantiation, solution, data transformation and simulation / visualization that further simplifies the semantic service composition task.

The semantic service composition task occurs in the approach after the generation of a so-called *model instantiation service*. To that end, model instantiation services will be discussed further in Section 5.6. In parallel to the choice of a template-path, the ontology-represented model can be wrapped as such a service that provides the data instantiation functionality as well as that it has a semantic service specification that represents the model semantics needed for composition and mediation. When the model instantiation service is available, the target system can be composed. Assuming for a moment a successful composition, it might be necessary to transform data in-between the service calls of the concretely bound services. This might, e.g., be required when a preprocessing step works on network graph representations of data, whilst a mathematical programming model formulates on a general set-parameter structure. Such transformations can be seen as placeholder services in the composition procedure to be generated semi-automatically after the composition step based upon semantic service specifications (recapitulate, e.g., the approach of Bowers et al. [BL04] presented in Subsection 2.6.3). This step is indicated by the "Generate Data Mediators" placeholder.

As a next step, the system might be executed with specific instance data. If the results are not satifactory, it might be required to change the abstract optimization model's formulation. This step might be based upon automatically suggested and executed reformulations such as a smoothing of constraints for meeting NLP solver requirements. If a modification is required, the process restarts at the generation of a model service with a possible rechoice of the template-path running in parallel. The AML representation of the changed model can be updated by using the AML Derivation functionality provided with the predefined model entity types of the model representation approach (Subsection 5.3.1 and Section 5.4). Further automatized steps for the possibly required recomposition of a new system are described in Section 5.6. This thesis does not provide technology on how to and whether to automatize the decision to modify the model and/or system after a first execution. Nonetheless, if for a certain problem domain error measures and decision procedures are available, an automated analysis of a solution/decision proposal generated by a certain system composition (and model formulation) could be implemented that leads to automated reconfigurations of models and the service composition itself. An exemplary scenario for the latter is also described shortly in Section 7.2.

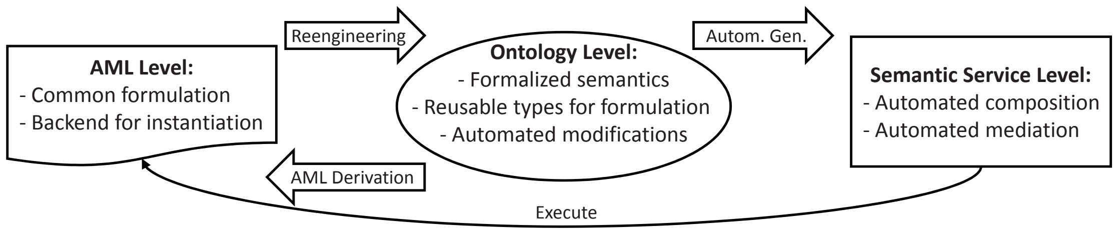


Figure 5.2.: Ontology representation as a layer in-between AMLs and services

To conclude the considerations in this section, let us outline in short how the ontology representation integrates with AMLs and services. Taking a look at Figure

5.2 one can see the ontology representation as a layer in-between the AML and the service level. That means, when editing models within the approach, one may start with a favored, supported AML and edit or just reuse a model. Suited support of AML Reengineering may allow to bring the model into an ontology representation with minimum effort. But it has to be mentioned that this step is optional. To that end, Section 5.5 only contains a very rough draft of AML Reengineering. For the scope of this thesis, one might therefore start in the ontology formalism which is presented in this chapter. Whilst the ontology representation supports different tasks such as specifying all kinds of semantics, model formulation, as well as a highly automated way to perform changes on the formulation of a model, a corresponding AML representation can always be obtained by the AML Derivation capabilities that will also be presented within this chapter. This is important as the AML representation itself serves as a backend for performing the instantiation of a model with data.

The functionality to instantiate a model with data as a frontend is provided by a model instantiation service. This service has a semantic service description to be used for semi-automated data transformation generation and service composition. It will be discussed how to put the ontology represented models' semantics into such a service description in Section 5.6.

5.2. A First Example Model: Min-Cost-Flow with integrality Requirements

The first example model that will be used in the remainder of this chapter is a common min-cost-flow problem. An AMPL formulation of this problem can be found in Figure 5.3. A standard reference on network flow problems is Ahuja et al. [AMO93]. For readers not familiar with AMPL-syntax, the AMPL book [FGK03] provides an excellent reference.

As one can see in the model, the min-cost-flow problem is formulated on a directed graph with two sets `Nodes` and `Arcs`. It is emphasized that the whole model, as typical for AMLs, is abstracted from concrete instance data. Parameters are given as follows: One can observe supply parameters (`"SupplyDemand"`) at nodes which are defined as usual with positive values for inflows/supplies, negative values for demands and a zero value in sum if a node is a *transshipment* node. There are linear flow cost parameters at the arcs plus an upper capacity for the throughflow of the single commodity considered at the moment. Lower capacities are given by enforcing zero as a lower bound on the flow variables `ArcFlow`. The arc-flows are to be determined in solutions of the abstract model's instances. An explicit integrality requirement is put on the flow variables `ArcFlow` by the keyword `integer`. The reason for this is that it will be demonstrated how to perform such a specification in the ontology-representation approach later on. Anyhow, the specification can be seen as

```
# Standard min−cost−flow problem from Ahuja et al., Network Flows
# Specialized AMPL formulations for network models are not used in the examples
# Integrality of flows is required for demonstrational purposes though integer flows are
      guaranteed for integral data!

set Nodes;
set Arcs within {Nodes, Nodes};

param Cost{Arcs};
param UpperCapacity{Arcs};
#Include general upper capacities but no lower ones.
param SupplyDemand{Nodes} integer;
#This parameter follows the convention > 0 for supply and < 0 for a demand

var ArcFlow{Arcs} >=0, integer;
#Zero lower bounds of flows in standard formulations.
#Integrality requirement for demonstrational purposes!

minimize FlowCost: sum{(i,j) in Arcs}(Cost[i,j] * ArcFlow[i,j]) ;

s.t. MassBalance{i in Nodes}: sum{(i,j) in Arcs}(ArcFlow[i,j]) − sum{(j,i) in Arcs}
(ArcFlow[j,i]) = SupplyDemand[i];

s.t. UpperCapacityConstraints{(i,j) in Arcs}: ArcFlow[i,j] <= UpperCapacity[i,j];
```

Figure 5.3.: Example: A standard min-cost-flow problem

somehow redundant since a standard min-cost-flow problem with suited, e.g., integral data that is solved by a specialized method for min-cost-flow or an LP-relaxation of the mathematical programming formulation has an integral solution.

The goal of the problem is to minimize the overall linear flow cost. Besides the `UpperCapacityConstraints`, the network problem typical `MassBalance`-constraints occur in the example model. The network balance in a very simple form as given in this example model forces the inflows and outflows at nodes to be equal. This is a requirement for an admissible network flow. The constraint formulation integrates the supply parameter for the respective node. If we, e.g., consider a supply node with a positive value of `SupplyDemand` at a node `i` with only outgoing arcs, the respective formulation for node `i` sums all the positive arc outflows and states that they should equal the positive supply value. The intuition in the latter case is that everything provided at the node should flow into the network via the outgoing arcs.

In order to keep the example simple and extendable, AML-typical constructs for network flow problems such as the `node` and `arc` keywords in AMPL (, see, e.g., [FGK03], Chapter 15) were not used.

In the progress of Chapter 6, extensions of this model will be presented. For this chapter, the simple min-cost-flow formulation of this section should suffice to discuss the basic ideas of the ontology representation approach.

5.3. Structure of the Representation

This section is concerned with the structure of the ontology representation for optimization models. A survey is given in the first subsection and then the fundamental vocabularies that define the representation structure of optimization models will be presented. An extensive example specification of the min-cost-flow problem introduced in the last subsection will be given at the end of this section.

5.3.1. Ontology Structure

The basic structure is visualized in Figure 5.4. An abstract optimization model is treated as instance knowledge, where different ontologies are used as vocabularies. The instance knowledge is indicated by the filled circles in the drawing. Goals and constraints and their relations to individuals for data and mathematical properties are aggregated in the filled circles where arrows show the structure of ontology property axioms. Also indicated by continuous black arrows is the typing of the individuals in classes that belong to ontologies for optimization model formulations, mathematical properties and data. In the non-filled ellipsoids one can observe placeholders for conceptualizations of domain data ontologies on the right, as well as aggregations of conceptualizations from the optimization modeling domain on the left.

The idea behind the approach is to have typed individuals for every *model entity*, i.e., goals, constraints, parameters, sets and variables. By *formulation entities*, the subclass of goals, constraints, or groups of related constraints encapsulated in a single class, are denoted. The model entities as individuals are similar to abstract model statements in AMLs, but there are important differences. At first, the expression structure of such statements is not explicitely represented within the ontology, but rather are the semantic types in use specified which then encapsulate the statement structure in a way that is explained in Section 5.4. More specific, the types are always accompanied by queries and implementations that together provide the functionality for building AML statement(s). The basic idea is to reuse the formulation types with different data conceptualizations and slightly varying expression parts such as inner summations.

To fortify the latter consideration, the representation is more abstract than in AMLs. In order to build an AML statement, the different conceptualizations of data and variables used in a constraint are to be specified before the actual derivation of

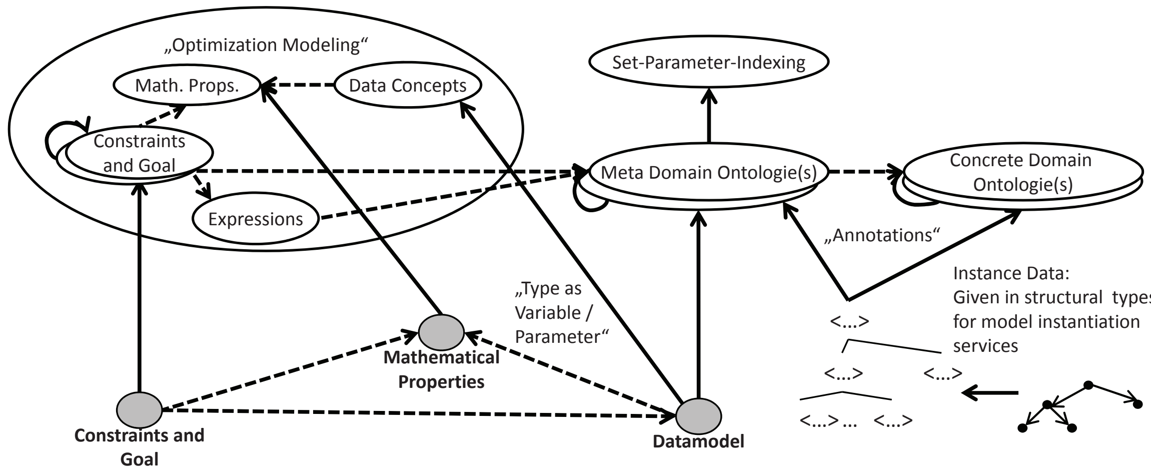


Figure 5.4.: Structure of the ontology representation

an abstract AML statement takes place. To that end, goals and constraints are grouped into ontologies for certain very abstract domains such as "networks" and also do only specify that they require abstract types from data ontologies, e.g., for "networks". Concretion is to be made by conceptualizations from subdomains such as water distribution networks or supply chains. A first short example of such a constraint specification will be given in what follows. Further elaborate examples will follow in the remainder of this thesis. The principles behind the definition of reusable formulation types and the criteria for especially creating subclasses are a topic of Chapter 6.

Another difference to AML representations should have become clear by now: Abstract optimization models are to be represented in an ontology graph structure. This means that manipulations on a model specification will also be manipulations of the knowledge graph given by the model. Besides the ontology types for formulation entities, the data conceptualizations for sets, parameters and variables are to be defined as semantic types from ontologies for respective application domains. These ontologies will be referred to as *meta domain ontologies* (mdo) in what follows. The meta domain ontologies model data conceptualizations of certain domains in terms of sets and parameters. The meta domain ontology concepts, as well as the goal and constraint types from the respective formulation ontologies to be discussed later on, are to be hierarchized by certain `subclass`-relations, as it is indicated in the graphics by continuous black arrows. This allows one to see certain constraint formulations as a specialization of others, and on the side of the meta domain ontologies, to insert concrete submodels for specialized domains such as in the already considered case of water distribution networks. Meta domain ontology types are also subtypes of general concepts for sets, parameters and variables as indicated by the arrows pointing to the "Set-Parameter-Indexing" and "Data Concepts" ontologies. "Set-Parameter-Indexing" provides a general vocabulary for explaining set-indexed data and set relations such as cartesian products, whilst "Data Concepts" allows to specify data entities, i.e., data concept individuals in the approach, to be seen as variables or input parameters in the context of a model. Both vocabularies will be explained in the following subsections. "Data Concepts" especially refers to a subset of the the `OM` vocabulary that will be discussed in the next subsection.

Since the data-related ontology classes are also to be used to support the generation of data transformations for the concrete model instantiation services, one also needs semantic specifications for concrete instance data. This relates to the notion of a *concrete domain ontology* (cdo) that can be found in a circle in Figure 5.4. An excerpt of such an ontology for concrete network data has already been given in Figure 2.7 of Subsection 2.5.2.

As the basic structure of ontology represented models has been introduced by now it is time for some first remarks. A thing to be mentioned is that Figure 5.4 provides a simplified and aggregated view on the representation approach. Therefore, the image serves as a basis to explain the standard features of a *model entity* specification. E.g., the special case of having reusable formulation types with the aid of ontology classes for expressions is indicated by the ellipsoid "Expressions" , but the respective properties will not be used until Subsection 6.1.3 of Chapter 6. The specification example below will give a first hint of a simple constraint-type usage indicated by the arrows directly connecting "Constraints and Goal" and "Meta Domain Ontolog(ies)" on the assertional and terminological level. The reader might also have

become a first idea how the requirement **Model Semantics** from Section 4.2 is covered. "Model Semantics" for *mathematical model properties* and their usage are to be explained in Section 5.6. At the moment it is observed that these properties are annotated to the entities of a model by means of properties which have concepts from a "Mathematical Properties (Math. Props.)" ontology in their range. For the requirements **Models as Services and Composition** as well as **System Integration** it was mentioned how data semantics for data transformation generation are specified within the approach. The abstract types required in **Model Formulation by abstract Types** are the core ingredients of the representation.

A First Example of a Constraint-Type Usage:
Figure 5.5 shows the ontology excerpts required to define a network problem typical balance constraint for every network node in a model. A target AMPL statement that is represented by the formulation type `Net#SingleCommodityBalance` would, e.g., be

```
                                        s.t.  MassBalance {i in Nodes}:
sum{(i,j) in Arcs}(ArcFlow[i,j]) - sum{(j,i) in Arcs}(ArcFlow[j,i])
                                                   = SupplyDemand[i];.
```
(5.1)

Note that the identifiers of sets, parameters, variables and the constraint in the AMPL model are not equal to the IRI suffixes of individuals in the ontology definitions. As typical for ontologies, a data property, in this case called `OM#EName`, can be used to annotate string names to IRIs. The respective property will be introduced in the next subsection.

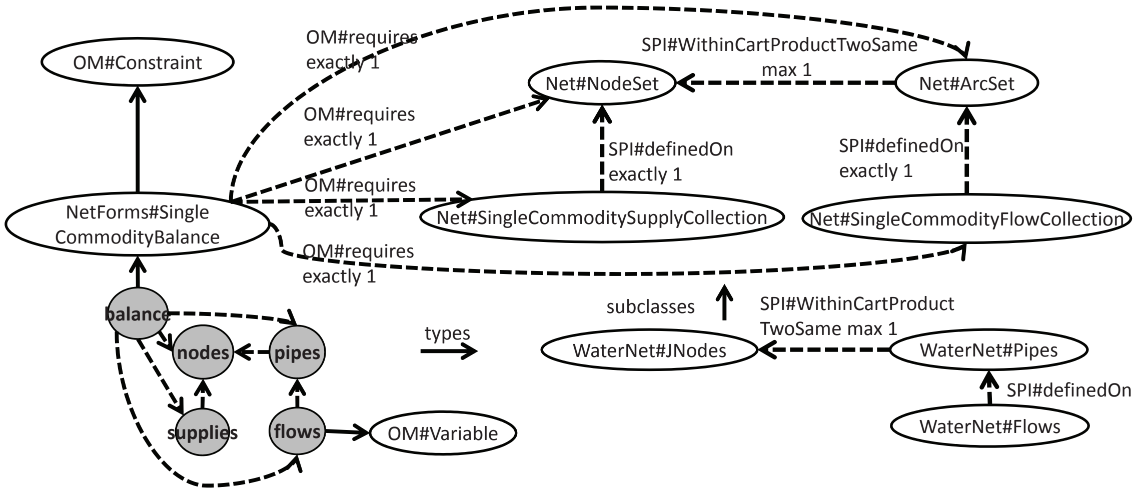


Figure 5.5.: Using a standard balance constraint type in a water distribution model

The specifications discussed at the moment only represent a single statement without a model context. A complete specification for the min-cost-flow model presented in Section 5.2 will therefore be given in Subsection 5.3.5. The specifications presented here, however, are not part of that model.

As one can see in Figure 5.5, an individual `balance` of the semantic type `NetForms#SingleCommodityBalance` has to be created for the ontology specification. The prefix `NetForms` refers to an ontology for network flow related formulations that will be considered further in the following sections, e.g., in 5.3.5. The semantic type prescribes some requirements on the terminological level. Besides other possible specifications, every instantiation of the `Net#SingleCommodityBalance`-formulation type requires network node- and arc-sets, as well as flow variable- and supply parameter collections (for single-commodity models) to be defined. The respective ontology classes stem from a meta domain ontology for networks that carries the prefix `Net`. Without prefixes the respective classes are identified as `NodeSet, ArcSet, SingleComodityFlowCollection` and `SingleCommoditySupplyCollection`. The ontology `Net` will be further discussed in Subsection 5.3.5.

The `SingleCommodityBalance` type represents flow conservation equations for a single commodity with a single flow variable on every arc. To that end, further use cases might require different formulation types for balance constraints that are more general. These will be introduced in the remainder of this thesis.

It is prescribed by some cardinality class restrictions that every instance of `Net#SingleCommodityBalance` is included in some axioms specifying the relation to concrete individuals of the abstract network types. When the constraint type is to be used in the context of a complete model, a concretion of these types may have to take place in order to be consistent with other constraints. Consider, e.g., how this constraint definition is part of an optimization model for water distribution systems. Therefore, the semantic types of network node-sets, pipe-sets and flows might have to be specialized to types for concrete junction nodes, pipes and respective flows of water in order to be consistent with other constraints. The semantic types in the ontology prefixed with `WaterNet` reflect these considerations. The type of the supply individual might not have to be adapted because its specification is not *shared* with other water network typical constraints. Note that the `flows` are also declared to be variables in the context of a model by the specification of the `OM#Variable`-type.

The latter example gave a first idea for the usage of a constraint type in the specification of a model and the vocabularies in use. The consideration of balance constraints in water networks will be further extended in a case study in Section 6.3. In scope of the complete model presented there it will also be required to overthink the orientation of balance equations and the conceptualization of supply. The specifications of the first example from above were not given in the context of a complete model in order to keep the types simple. As a next step, the fundamental vocabularies for optimization model specifications will be presented.

5.3.2. Top Level Definitions in `OM`

This subsection is concerned with the most elementary vocabulary "Optimization Modeling" that is denoted by the prefix `OM`. Figure 5.4 from Section 5.3 already suggested to bundle classes for concepts relevant in optimization into such an ontology and to furthermore structure concepts for mathematical properties, data conceptualizations in the context of optimization and formulations into specialized ontologies that in union all build the "Optimization Modeling" ontology framework. It will be discussed now how the top level definitions for modeling components can be put into the basic optimization modeling ontology and an extra, imported mathematical property ontology. Furthermore, it will be explained how formulation ontologies for goal and constraint types, generated by users and experts, can be put under this framework. The definitions are visualized in Figure 5.6.

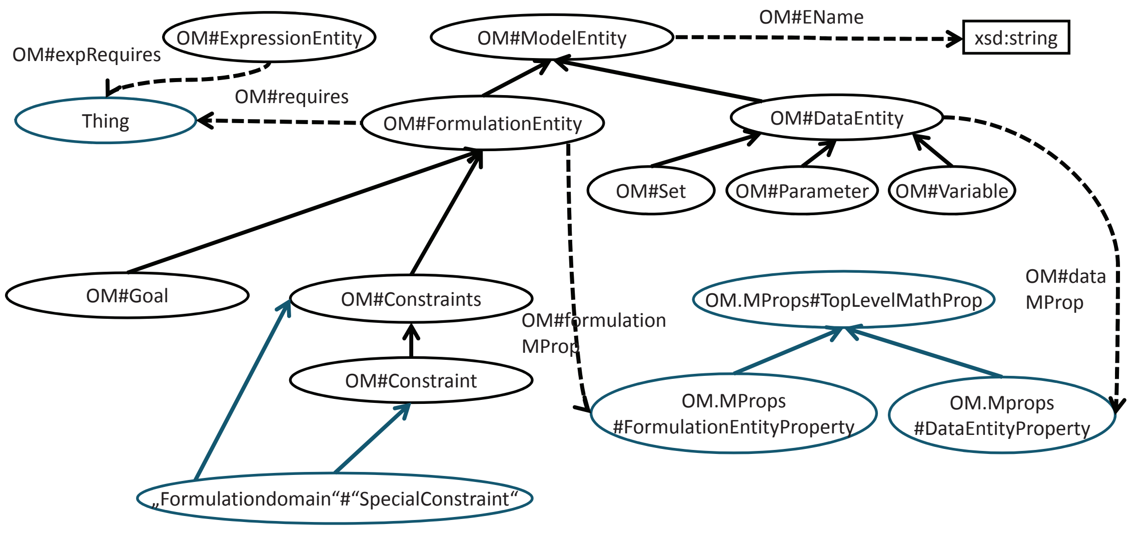


Figure 5.6.: Top level optimization modeling definitions

Except the class `OM#ExpressionEntity` which will be explained and motivated in Section 6.1.3, all definitions fit under the "top-level concept" `OM#ModelEntity`. `OM#ModelEntity` provides a class for all entities occurring in an abstract optimization model such as sets, parameters, variables, constraints and goals. Under this class, the respective conceptualizations for the model entities are separated into the classes `OM#FormulationEntity` and `OM#DataEntity`. The definitions under `OM#DataEntity` especially refer to the "`Data Concepts`" subvocabulary from Figure 5.4. Every model entity can carry a string-identifier referred to as a name that can be used in AML representations of an abstract optimization model. The respective data property `OM#EName` is defined for every model entity. Most important for the data model abstracted representation of optimization model formulations such as reusable constraints is the `OM#requires` property used for formulation type definitions to state which data conceptualizations are required for using the type. The range of `OM#requires` is left generous to be `Thing`. Reusable formulation conceptualizations will make object property restrictions that restrict the range to respective subclasses in the respective context. A complete definition of the `OM`-ontology in Manchester Syntax can be found in the appendix of this thesis.

Mathematical properties of formulation and data individuals such as linearity, integrality and further are to be concluded or specified when a specific optimization model is defined in the ontological representation (, see, Section 5.6 and its subsections). For this purpose, the "Mathematical Properties" ontology referred to under the prefix `OM.MProps` provides a suited vocabulary. The details of this vocabulary will be discussed in Subsection 5.3.4. `OM.MProps` has a top-level concept `OM.MProps#TopLevelMathProp` under which the subclasses `OM.MProps#FormulationEntityProperty` and `OM.MProps#DataEntityProperty` are structured. The most important thing to mention is that mathematical properties in the ontology formalism can be referenced and used by the object properties `OM#formulationMProp` and `OM#dataMProp` with their respective domains and ranges.

Reusable conceptualizations are to be provided in extra ontologies that use the `OM` and `OM.MProps` vocabularies as imports, e.g., by stating constraint types to be subclasses of `OM#Constraint`. The issues concerned with generating and organizing reusable formulation conceptualizations will be further discussed in Chapter 6.

5.3.3. The Set-Parameter-Indexing Vocabulary

The "Set-Parameter-Indexing" ontology referred to in this thesis under the prefix `SPI` formalizes concepts for the definition of meta domain ontologies. The ontology takes a set- and parameter-based view as it provides concepts for the mdo-elements that enter abstract optimization models as sets, parameters and variables. Ontologies for data conceptualizations stemming from other applications will have to be mapped to classes that fall under the set-parameter view provided by `SPI` in order to integrate with the `OM` and formulation ontologies in the representation approach.

The relevant excerpts of the `SPI`-vocabulary are visualized in Figure 5.7. As there is no possibility of misunderstandings, the prefixes for `SPI` concepts will be omitted in the text of this subsection. Under the top-level concept `SPITopLevelConcept`, the class `SPIEntity` can be found. The two central classes in the approach are `Set` and `ParameterCollection` which are both subclasses of `SPIEntity`. These two latter classes provide the basic vocabulary to see conceptualizations for data models as sets and parameters. A family of parameters, reflected by the type `ParameterCollection` is to be indexed by members of a `Set`. In this context one can say that the parameters are defined over a set and reflect this relation by the object property `definedOn`. The property `definedOn` is defined with domain `ParameterCollection` and the very general range `TopLevelConcept` though its favored use is with `Set` individuals in the range. Examples in later sections will also enforce the usage of `definedOn` with `SimpleParameter`. SimpleParameter provides a class for those parameters that are not defined over a set and hence have no indexing. An example would, e.g., be a simple time parameter definition such as `param T > 0;` in AMPL.

An important subclass of `Set` is `UnionSet`. This class provides means to specify implicit sets that are given as a union of two or more explicitely specified sets.

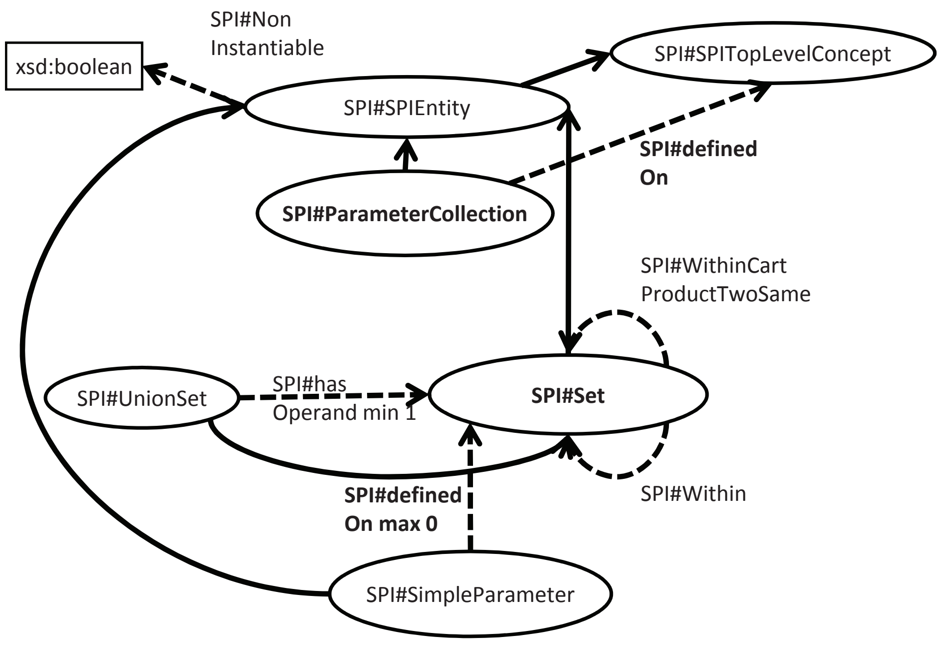


Figure 5.7.: Excerpt of the `SPI` ontology

Applications for this are, e.g., given by indexing expressions such as in the following AMPL statements:

```
                    set A;
                    set B;
   param PUnionSet{ A union B };
```
(5.2)

Further examples will be given in the examples of Section 6.1.3. The specification of a union-set works by creating an individual of `UnionSet` and relating it to the concrete sets that are operands of the set union by the object property `hasOperand`, see, again Section 6.1.3 for examples.

Since an individual that represents a union set should typically not be translated to a set-defining AML statement in the set-declaration part of a model, the data property `NonInstantiable` can be used to mark the union set as "non-instantiable". Speaking in general, all instances of `SPIEntity` can be marked as non-instantiable and will hence not be derived into AML statements. Examples for this are typically given by union sets that are defined implicitely, e.g., within an indexing as in (5.2). Note that using this property is optional. It also might seem more intuitive to put the respective property into the `OM`-vocabulary of the previous subsection with domain `OM#ModelEntity`. But since the treatment of non-instantiability for general model entities would have further complicated the design and especially the implementation of statement derivations in the approach, a demo-implementation for `SPI`-entities was chosen in order to keep the representation of this thesis manageable.

As the image only represents an excerpt of the `SPI` vocabulary, it is mentioned that a full specification of the `SPI` vocabulary in Manchester Syntax can be found in the appendix of this thesis.

5.3.4. The Mathematical Properties Ontology

The purpose of this subsection is to introduce the `OM.MProps` ontology for specifying or generating mathematical properties of abstract optimization models' entities. This refers to the informal shortcut "Math. Props." for mathematical properties in the ontology structure Figure 5.4. As already mentioned in Subsection 5.3.2, the `OM` vocabulary imports the `OM.MProps` ontology and then specifications can be included into ontology-represented models by using the `OM#formulationMProp` and `OM#dataMProp` object properties. Figures 5.8 and 5.9 show the relevant excerpts of `OM.MProps` that will be discussed here. A complete specification in Manchester Syntax can be found in the appendix of this thesis. As in the previous subsection, the prefix `OM.MProps` for the considered ontologies' classes will be omitted in the text.

It has already been stated that `OM.MProps` has a top-level concept `TopLevelMathProp` under which the two categories `FormulationEntityProperty` and `DataEntityProperty` are set up. This reflects the distinction between properties of data entities that are typically to be specified by a user, such as a variable having an integrality requirement, and formulation entities that can be linear, continuous, convex and more depending on how the related data entities are used in the model. E.g., a constraint formulation without a variable might just be a presolve computation of a parameter and does not change the character of a linear model. To that end, mathematical formulation entity properties should be derived from the model context, e.g., in an automated fashion with methods from Section 5.6. Taking a look at Figure 5.8, a point that might draw the attention of the reader is the presence of individuals, e.g., `ZeroLowerBoundInd` of the type `NonNegativeLowerBound`. These individuals are defined in the ontology such that abstract optimization models specified in the ontology formalism can use them either for their specificational purposes, or, if rule knowledge has to be applied, in order to have them available for the consequents of SWRL-rules.

Data entity property classes (Figure 5.8) especially contain `IntegerData` and `BinaryData` as consecutive subclasses for specifying integer and 0-1 ranges of parameters and variables. Further classes are for specifying bounds/ranges. A simple

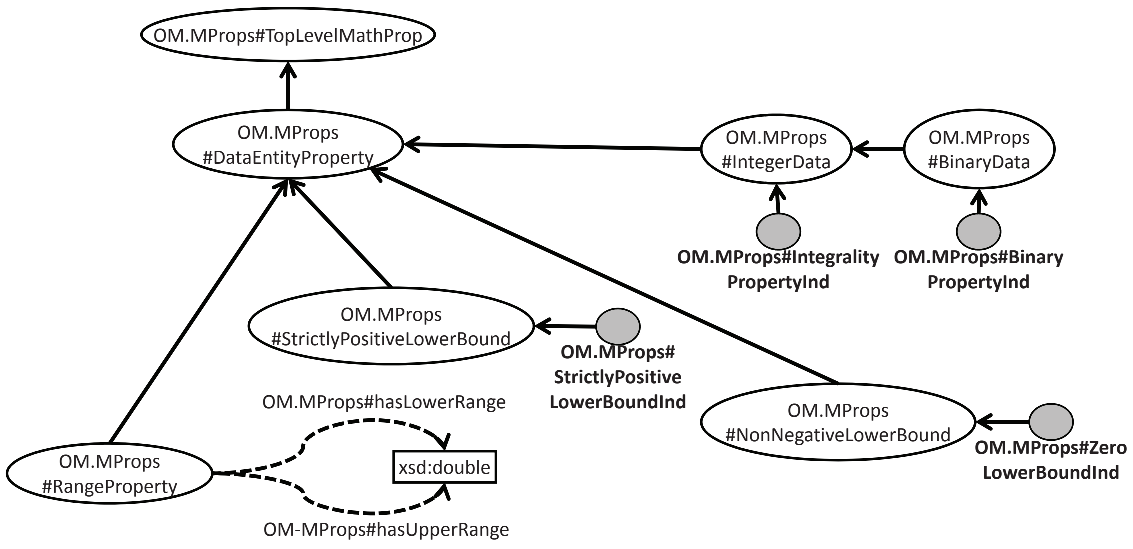

OM.MProps#TopLevelMathProp
OM.MProps #DataEntityProperty
OM.MProps #IntegerData
OM.MProps #BinaryData
OM.MProps#Integrality PropertyInd
OM.MProps#Binary PropertyInd
OM.MProps #StrictlyPositiveLowerBound
OM.MProps# StrictlyPositive LowerBoundInd
OM.MProps #NonNegativeLowerBound
OM.MProps#Zero LowerBoundInd
OM.MProps#hasLowerRange
OM.MProps #RangeProperty
xsd:double
OM-MProps#hasUpperRange

Figure 5.8.: Excerpt of the `OM.MProps` ontology for data properties

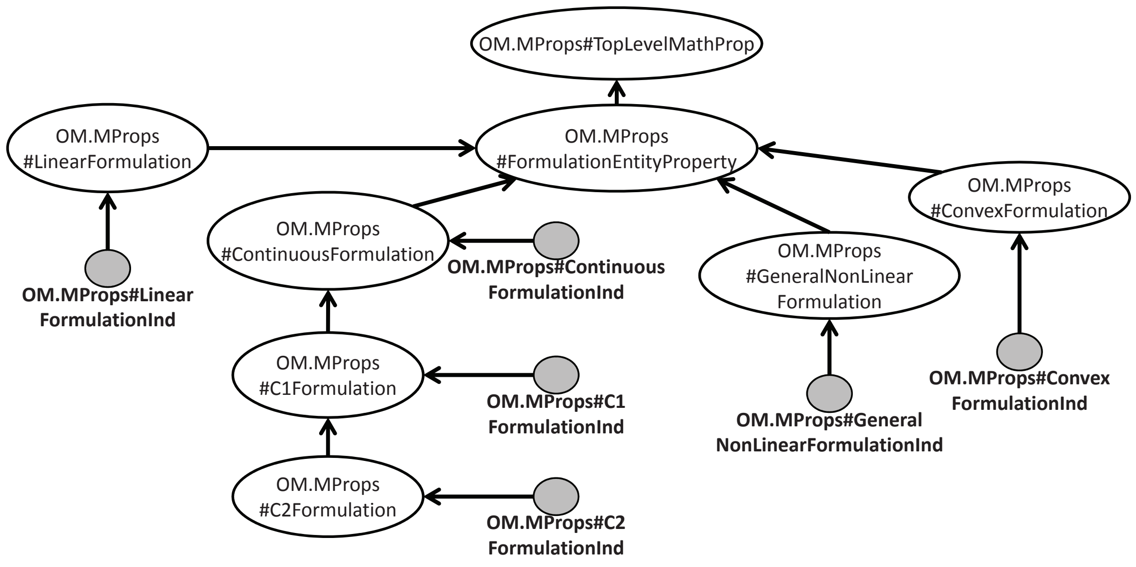

OM.MProps#TopLevelMathProp
OM.MProps #LinearFormulation
OM.MProps #FormulationEntityProperty
OM.MProps #ConvexFormulation
OM.MProps #ContinuousFormulation
OM.MProps#Continuous FormulationInd
OM.MProps #GeneralNonLinear Formulation
OM.MProps#Linear FormulationInd
OM.MProps #C1Formulation
OM.MProps#C1 FormulationInd
OM.MProps#General NonLinearFormulationInd
OM.MProps#Convex FormulationInd
OM.MProps #C2Formulation
OM.MProps#C2 FormulationInd

Figure 5.9.: Excerpt of the `OM.MProps` ontology for formulation properties

class to use often is the `NonNegativeLowerBound` which provides a concept to state a lower bound of zero. More general, the `RangeProperty` class can be used to specify lower and upper bounds, e.g., on variables. Since the definitions of general bounds differ, models making use of this construct have to define mathematical property class individuals on their own. Furthermore, the respective individual has to make use of a respective data property `hasLowerRange` or `hasUpperRange` with a double value.

Formulation entity properties specify mathematical properties of mathematical programming formulations in the context of a model. Properties included in Figure 5.9 are *linearity* and *continuity*, which is refined by one- and two-time *continuous differentiability*, as well as *convexity* and a *nonlinear* formulation property. The respective classes are `LinearFormulation, ContinuousFormulation, C1Formulation, C2Formulation, ConvexFormulation` as well as `GeneralNonLinearFormulation`.

5.3.5. Ontology Specifications for the Min-Cost-Flow Problem

The usage of the `NetForms#SingleCommodityBalance` type in Subsection 5.3.1 gave a first impression on the interplay of different vocabularies. But a complete model has not been specified yet. For the instance structure of a complete ontology-represented model, the reader may reconsider the min-cost-flow model from Section 5.10. This model will now be specified in the ontology representation. A visualization of the graph structure on the assertional level for this model is given in Figure 5.10. The drawing does not specify the used types in the respective vocabularies as well as that the respective names of properties are not included. This information has been put into two extra Figures 5.11 and 5.12. In these figures, one can find excerpts of the respective vocabularies in use plus the explicit type definitions for the model individuals. The ontology-represented model can also be found in form of an OWL-XML file on the digital material accompanying this thesis.

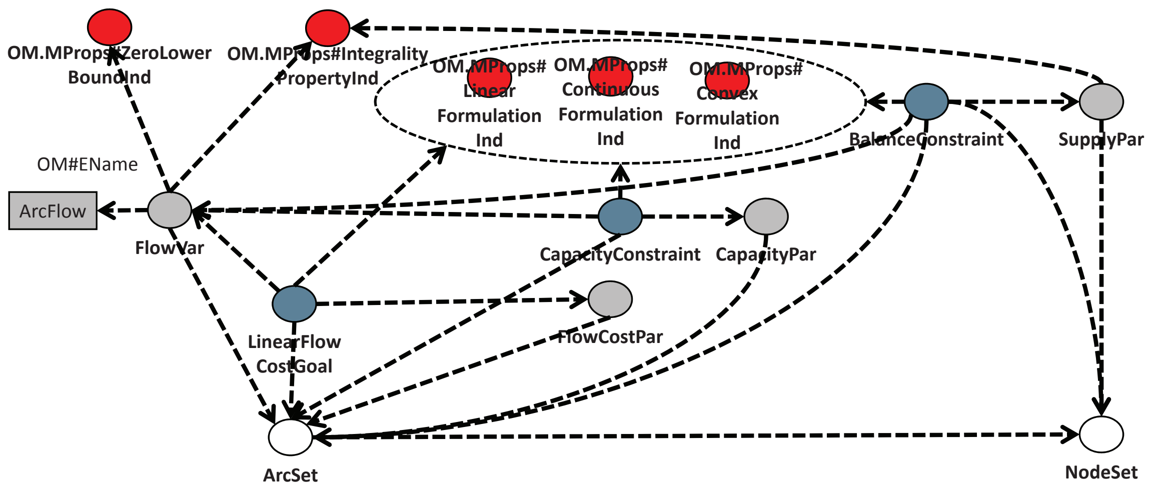

OM.MProps#ZeroLower BoundInd
OM.MProps#Integrality PropertyInd
OM.MProps# Linear Formulation Ind
OM.MProps# Continuous Formulation Ind
OM.MProps# Convex Formulation Ind
BalanceConstraint
SupplyPar
OM#EName
ArcFlow
FlowVar
CapacityConstraint
CapacityPar
LinearFlow CostGoal
FlowCostPar
ArcSet
NodeSet

Figure 5.10.: Min-cost-flow instance structure in the ontology representation

The model consists of three formulation entities, six *data entities* and some individuals for mathematical properties that are not newly defined in the model but imported from `OM.MProps`. The types of the formulation entities stem from an ontology for network formulations that will further be referred to under the prefix `NetForms`. Excerpts of this vocabulary will be explained in the remainder of this subsection. In contrast to the classes, the individuals that represent model entities carry no prefix in this simple example since they are all to be integrated into one current ontology for a model. The mathematical property individuals, which are not model entities, have a prefix since they stem from an imported ontology. Concerning the model entities, the three individuals are `LinearFlowCostGoal,` `CapacityConstraint` and `BalanceConstraint`. The respective types from the `NetForms` vocabulary are given by `NetForms#LinearSingleCommodityFlowCostGoal,` `NetForms#SingleCommodityBalance` and `NetForms#SingleCommodityFlowBoundsUp`. Ontology excerpts defining these types can be found in Figure 5.11. The type definitions for the model entities are also visualized in Figure 5.12. The three types from the `NetForms` vocabulary define their required data conceptualizations where all required classes stem from the ontology with the prefix `Net`. Excerpts for this kind of definition for the type `NetForms#SingleCommodityBalance` were already given in Figure 5.5.

In Figure 5.11, the formulation classes are visualized in the grey bordered ellipsoids. As already mentioned, every formulation class requires certain mdo-classes for its translation into a statement. This kind of interface is specified by using the `OM#requires` property in respective cardinality restrictions. All formulation classes require exactly one individual of `Net#SingleCommodityFlowCollection` representing the flow variables of the model. In order to be treated as a variable, a second type definition to the type `OM#Variable` has to be performed for the individual `FlowVar` of the class `Net#SingleCommodityFlowCollection`. This is illustrated in Figure 5.12. The prescribed usage of other mdo-classes differs from formulation class to formulation class. The `NetForms#SingleCommodityFlowBoundsUp`, e.g., require `Net#UpperCapacityCollection, Net#ArcSet` plus the already mentioned `Net#SingleCommodityFlowCollection`. The cardinalities for these specifications are forced to exactly one, which gives a first hint that sharply formalized ontology knowledge goes over the flexible usage of single formulation types. Requirements for more general formulations will be resolved by additional types to be structured into the ontologies. More general statements can be derived with more general types and the modeling principles behind these types will be a topic of Chapter 6. Nonetheless, the abstraction of the semantic data types is a feature of every formulation type. The concrete statement to be derived from the usage of `NetForms#SingleCommodityFlowBoundsUp` in the context of the min-cost-flow model as specified in the ontology graph in Figure 5.10 is

```
s.t.  UpperCapacityConstraints { (i,j) in Arcs } :
                 ArcFlow[i,j] <= UpperCapacity[i,j];
```
(5.3)

This statement is also included in the complete AMPL min-cost-flow model from Figure 5.3 in Section 5.2. It will be explained how such statements can be derived from reusable types in Section 5.4. The identifiers in the AMPL-statement differ from the abbreviated IRIs of ontology individuals as it has already been mentioned. Axioms using the data property `OM#EName` specify the respective identifiers. These axioms can be found in the full specification of the ontolgy-represented min-cost-flow problem. In order to keep the images simple, Figure 5.3 only contains the *entity name* ("`EName`") of the flow variables which is `ArcFlow`.

The character of types such as `Net#SingleCommodityFlowCollection` should be explained in more detail. The term "`SingleCommodity`" refers to the usage of the respective mdo-concept for flow in a model context where only one flow good is used and therefore flow variables, respective constraints and further parameters are not indexed over a commodity set. This restricts the generality of the mdo-concept (as well as in the respective capacity constraint type `NetForms#SingleCommodityFlowBoundsUp`) but is a necessary modeling paradigm in order to avoid contradictions in the *open-world semantics* of the ontology language OWL 2. More on these issues will become clear when the network vocabularies and example models are extended in Section 6.1, such that more types become available and their relations have to be discussed.

The type `NetForms#SingleCommodityBalance` requires exactly one individual of `Net#NodeSet, Net#ArcSet, Net#SingleCommoditySupplyCollection` and `Net#SingleCommodityFlowCollection`. The respective types for `NetForms#LinearSingleCommodityFlowCostGoal` are `Net#SingleCommodityCostCollection` (again a commodity individual type), `Net#SingleCommodityFlowCollection` and `Net#ArcSet`.

The ontology structure in Figure 5.10 shows some mathematical property individuals being specified for the model entities via axioms. The mathematical property individuals are displayed in red filled circles. The two individuals `OM.MProps#ZeroLowerBoundInd` and `OM.MProps#IntegralityPropertyInd` from the `OM.MProps` vocabulary represent a zero lower bound and an integrality requirement. They are both specified for the `FlowVar` individual representing the flow variables in the model. The two specifications reflect the AML statement

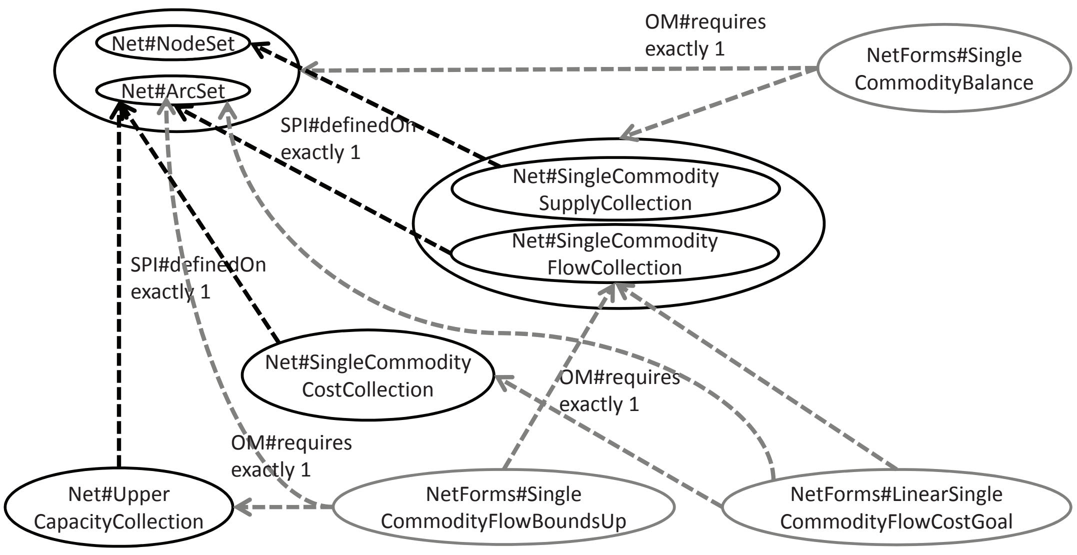


Figure 5.11.: Types in use for the min-cost-flow model in the ontology representation

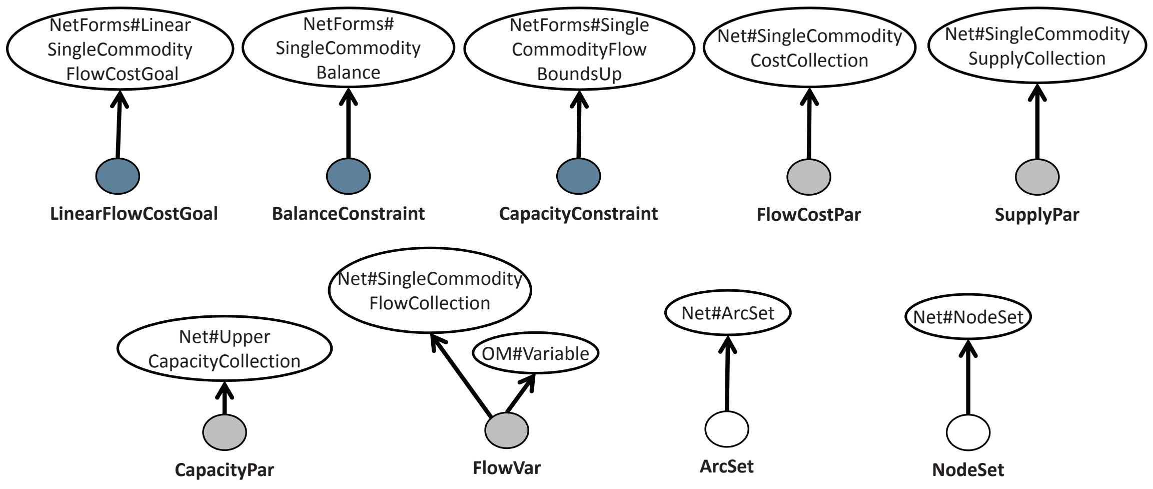


Figure 5.12.: Type definitions for the min-cost-flow model

excerpts `>= 0` and `integer` in the defining AMPL statement

```
var ArcFlow { Arcs } >= 0, integer;
```

(5.4)

of the flow variables from Figure 5.3. The `IntegralityPropertyInd` of type `OM.MProps#IntegerData` is also used by the individual `SupplyPar` which reflects an integrality requirement for the input supply parameters. This illustrates that mathematical property individuals can be shared by different model entities.

The data entity properties are to be defined when editing the model in its ontological representation or retrieving it in a semi-automated fashion by AML Reengineering (,see, Section 5.5). In contrast to that, formulation entity properties should be concluded form the context of a model definition. This point will be further discussed and demonstrated in Section 5.6. For the moment, observe that the three individuals `OM.MProps#LinearFormulationInd` of type `OM.MProps#LinearFormulation`, `OM.MProps#ConvexFormulationInd` of type `OM.MProps#ConvexFormulation` and `OM.MProps#ContinuousFormulationInd` of type `OM.MProps#ContinuousFormulation` are properties of the formulation individuals.

5.4. Expression Structure Separation and AML Derivation

This section will explain how the structure of AML statements, especially the algebraic structure of abstract optimization models' goals and constraints, can be separated from the ontology formalizations. Then, a suited derivation process, as it will be presented in Subsection 5.4.4, can instantiate the optimization model constituents into statements that are valid for a target AML.

Statements in algebraic modeling languages are valid for the context-free grammars that specify the languages. A valid parse of a statement can be represented in a parse-tree and later be processed by interpreter programs. But the tree-like structure of a statement is something very unlikely to be represented in an ontology. Especially, the representation of statement structure with all its complexity as, e.g., different operators for sums, inequalities and products, would blow up the ontology at no purpose and make query answering and reasoning tasks harder and more complex from a computational perspective. To that end, separating the expression structure from an ontology specification is of great importance. When doing so, further goals can be achieved.

The basic idea of the ontology represented abstract optimization models considered in this thesis is the typing of *model entities* such as specific sets, parameters, variables, goals and (groups of) constraints in a standard vocabulary of reusable types with explicitely formalized semantics. Already mentioned benefits are the reusage of constraint formalizations with different data models, simple model editing operations for model formulation, as well as model semantics for service composition. These benefits rely on the conceptualizations but not on the concrete statement structure.

Figure 5.13 describes the basic structure of the statement structure separation. It is centered around every *model entity type*, i.e., the types for certain constraints such as "balance" or data conceptualizations such as a network node-set for which multiple

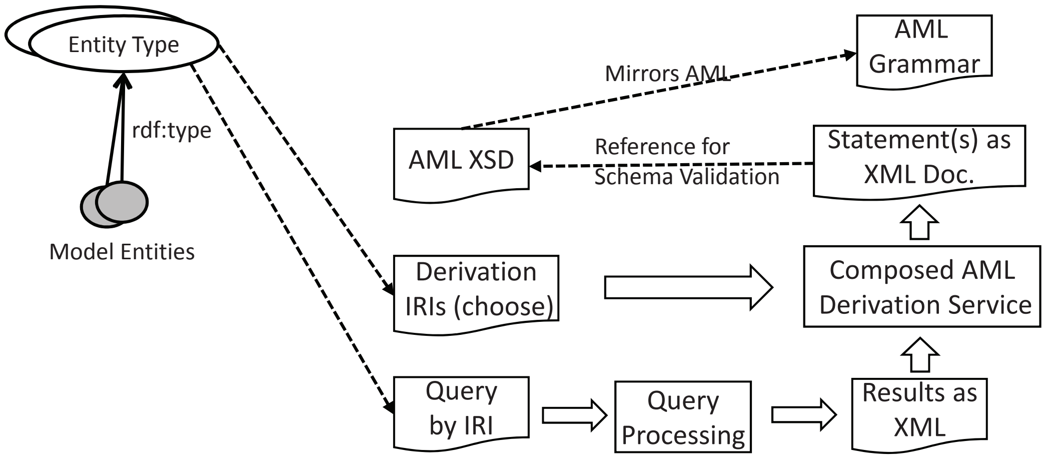


Figure 5.13.: Expression structure separation by query and derivation implementations

individuals can exist in a model. The fundamental steps for deriving all statements of individuals for a single type are the processing of a query and the transformation of the query results into statements by certain *derivation implementations*. The details are to be discussed now. First of all, the type of a model entity is accompanied by an IRI that can be resolved to a unique ontology query, as well as multiple IRIs representing derivation resources for all individuals of the model entity type. Both the unique query IRI and the implementation IRIs can be referenced inside the ontology vocabularies by annotation properties.

The implementation IRIs can be resolved to services that implement derivation capabilities for the respective entity types and a chosen target algebraic modeling language. For the latter reason, multiple such implementations for the same type may exist. Suited derivation services may be composed into greater services which allow for the derivation of whole models based on techniques for semantic service description and composition. But as such compositions would go beyond the scope of this thesis, a simplified view is taken here that leaves the latter opportunities for flexible compositions and choices of derivation implementations open: It is assumed that an implemented process for deriving a whole ontology represented abstract optimization model knows about unique, correct derivation implementations for every entity type. Nonetheless, since these implementations should in principle be deployed as services, the interaction of the derivation procedure takes place with a centralized *AML Derivation Service* (, see, also Figure 5.14). This service is a central hub for deriving all entities of a type as an XML document. The input to this service consists of the query results to be discussed the next, as well as the string IRIs of the currently chosen derivation implementations. The service is assumed to be known and existent, and the opportunity to later replace it by more individual implementations is left open. A derivation procedure for whole models making use of the central AML Derivation Service is described in Subsection 5.4.4.

Queries and derivation implementations have to fulfill certain requirements in order to be generous enough. They will, e.g., have to deal with the disambiguation of variables and parameters in the context of a model, mathematical properties, or implicit union-sets in indexings. The requirements on queries and implementations will be discussed in Subsection 5.4.2. Let us now further focus on the process of transforming all model entities of a single entity type into the respective target AML statements.

The derivation process for all individuals of a model entity type that was outlined above furthermore works as follows: At first, a suited query processor for the ontology query language in use processes the referenced ontology query with the model and all directly and indirectly imported vocabularies as a knowledge base. SPARQL-DL will be used as a query language (, see, [SP07]) in the examples of this thesis. Another idea would be to use SPARQL queries on RDF graph representations obtained from the collected OWL ontology knowledge. But due to the need for mixed Abox and Tbox queries as well as implementational considerations for the demonstrator (, see, Chapter 8), it was decided to use the distinct subset SPARQL-DL of SPARQL (see again Subsection 2.5.3), for which an API including a query processor that is based upon OWL API is available (, see, `http://www.derivo.de/ressourcen/sparql-dl-api.html` for more information). SPARQL-DL is at date no W3C standard.

A requirement on the query processor and the query language is given by the need to support an XML result format. SPARQL Query Results XML Format [JD13] provides a standard for OWL ontologies and can be used both for SPARQL and SPARQL-DL queries. The XML query results should contain all important information on the entity type individuals, e.g., model entity IRIs, e-names and the IRIs and names of all required/definedOn or further individuals existing in object property axioms of the model entities. For the complete requirements, the reader is referred to Subsection 5.4.2.

The query result XML together with a string representing the IRI of the type derivation implementation can be sent in a message to a composed AML Derivation Service. As already mentioned above, this service is a (hierarchical) composition with derivation services for model entity types, i.e., the aforementioned entity type implementations. To that end, it should be said that the AML Derivation Service has to be deployed as a "whole" in order to justify the word service (the hierarchical composition is only available for software components). The composed AML Derivation Service provides a single method for instantiation and delegates and manages the calls to the statement type derivation services based on the input string-IRI, see, again Figure 5.14.

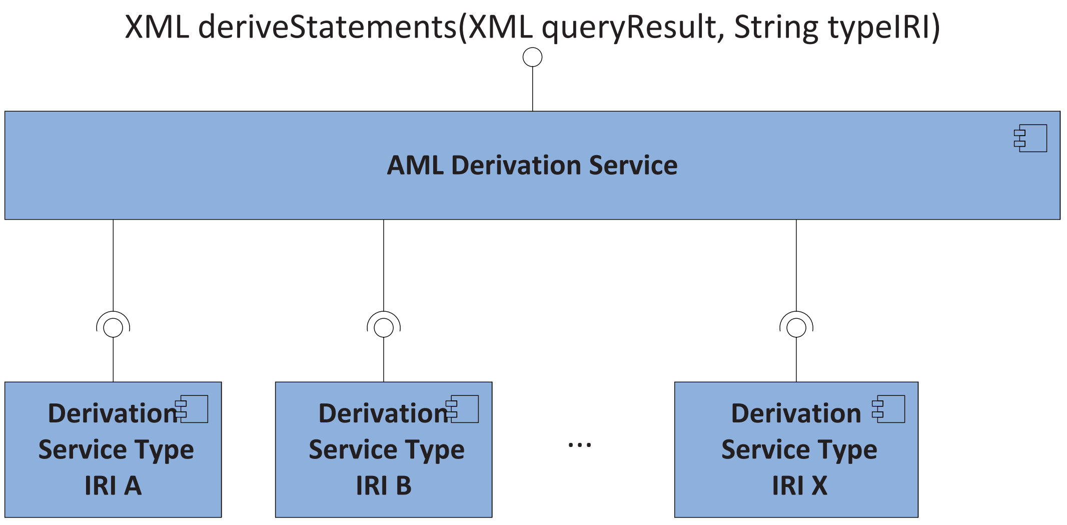

XML deriveStatements(XML queryResult, String typeIRI)
AML Derivation Service
Derivation Service Type IRI A
Derivation Service Type IRI B
...
Derivation Service Type IRI X

Figure 5.14.: AML Derivation Service as a hierarchical composition (treat derivation services as components)

After the generation of statements, the validation of derived AML statements against an XML Schema (Figure 5.13) representing the relevant excerpts of the target AML's grammar in XSD is a final possible step in the derivation process for a single entity type. A target AML XSD for the modeling language AMPL has been implemented for this thesis and can be found in the accompanying material in digital form. The structure of valid XML documents for the XSD will also be illustrated by the examples of the following subsections. The implementations of SPARQL-DL queries and type statement derivations will also be demonstrated in the following examples and further specifications can be found in the digital material.

Let us now start with a first simple example of a derived flow variable definition in AMPL and then explain the general requirements on query and type statement derivation implementations. This will be further concluded by a more complex statement example later on. After a description of the derivation process for complete abstract optimization models, a discussion of the statement structure separation approach will conclude this Section in 5.4.5.

5.4.1. Flow Type Example of a Statement Derivation

In this subsection, the statement derivation process presented for all individuals of a type will be demonstrated for the entity type

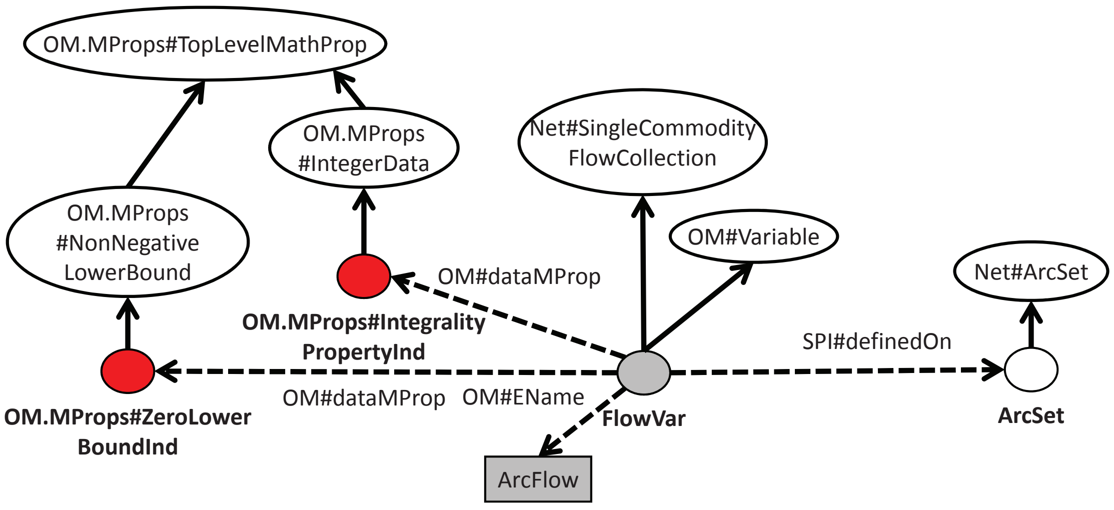

OM.MProps#TopLevelMathProp
OM.MProps #IntegerData
Net#SingleCommodity FlowCollection
OM.MProps #NonNegative LowerBound
OM#Variable
Net#ArcSet
OM#dataMProp
OM.MProps#Integrality PropertyInd
SPI#definedOn
OM.MProps#ZeroLower BoundInd
OM#dataMProp
OM#EName
FlowVar
ArcSet
ArcFlow

Figure 5.15.: Excerpt of ontology knowledge that is required for deriving the flow variable statement in the min-cost-flow model

`Net#SingleCommodityFlowCollection` and the specifications made for the min-cost-flow example in Subsection 5.3.5. This should exemplify the process and illustrate the inner workings of query and derivation implementations. At first, we reconsider the single target AMPL statement

```
var ArcFlow { Arcs } >= 0, integer;
```
(5.5)

for the defintion of a family of flow variables for every network arc in the min-cost-flow model from Figure 5.3. The required ontology definitions were given in Subsection 5.3.5. Figure 5.15 provides a survey of the required definitional excerpts and combines instance and type knowledge. The figure shows the excerpts of the model and vocabulary specifications required for deriving the target flow-variable statement.

The Query:
The first step to perform for the derivation of the target statement is the processing of the unique SPARQL-DL query for the type `Net#SingleCommodityFlowCollection`. SPARQL-DL allows for mixed ABox and TBox queries which is important for our purposes. Figure 5.16 shows the first part of the query which already contains all important aspects. After a definition of vocabulary prefixes the variables for which the results are to be returned are defined in a `SELECT` statement. Since SPARQL-DL is a distinct subset of SPARQL, the syntax is similar to SPARQL as well as to SQL. The query variables are also given in

```
PREFIX om: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.
    owl#>
PREFIX mdo: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/
    MDO−Network.owl#>
PREFIX spi: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−
    Set−Parameter−Indexing.owl#>
PREFIX mprop: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/
    OM.MProps.owl#>
SELECT ?parvarflow ?nonInstantiable ?omEntityType ?setarc ?unionsetoperandsarc
     ?unionsetoperandnames ?mathprops
WHERE
{
Type(?parvarflowiri, ?omEntityType),
PropertyValue(?parvarflowiri, spi:NonInstantiable, ?nonInstantiable),
EquivalentClass(?omEntityType, om:Parameter),
PropertyValue(?parvarflowiri, om:EName, ?parvarflow),
Type(?parvarflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?parvarflowiri, spi:definedOn, ?setarciri),
Type(?setarciri, mdo:ArcSet),
PropertyValue(?setarciri, om:EName, ?setarc),
Type(?setarciri, spi:UnionSet),
PropertyValue(?setarciri, spi:hasOperand, ?unionsetoperandsarc),
PropertyValue(?unionsetoperandsarc, om:EName, ?unionsetoperandnames),
PropertyValue(?parvarflowiri, om:dataMProp, ?mathprops)
} OR WHERE
{
 ...
} ...
```

Figure 5.16.: First part of query for flow-type derivation

the following listing with an explanation of their meaning:

- `?parvarflow`: This variable queries the e-names of all individuals of `Net#SingleCommodityFlowCollection`. The respective variable querying the ontology individuals of the type is `?parvarflowiri`. The conjuncted query-patterns `PropertyValue(?parvarflowiri, om:EName, ?parvarflow)`, `Type(?parvarflowiri, mdo:SingleCommodityFlowCollection)` build-up the respective part of the query.

- `?nonInstantiable`: This variable determines boolean values of the

```
... OR WHERE
{
Type(?parvarflowiri, ?omEntityType),
EquivalentClass(?omEntityType, om:Variable),
PropertyValue(?parvarflowiri, om:EName, ?parvarflow),
Type(?parvarflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?parvarflowiri, spi:definedOn, ? setarciri ),
Type(?setarciri, mdo:ArcSet),
PropertyValue(?setarciri, om:EName, ?setarc),
PropertyValue(?parvarflowiri, om:dataMProp, ?mathprops)
} ...
```

Figure 5.17.: Matching group of query atoms

`SPI#NonInstantiable` property for the above mentioned data entity individuals.

- `?omEntityType`: The conjuncted query atoms `Type(?parvarflowiri, ?omEntityType), EquivalentClass(?omEntityType, om:Parameter)` restricts the results to data entities of the flow type that are defined as parameters in the context of a model. It will be explained how to query for variables below.

- `?setarc`: The flow-variables or parameters in the model to be queried are indexed over arcsets whose e-names are determined for the `?setarc` variable. The respective variable `?setarciri` for the entity IRI is implicit. The query atom `PropertyValue(?parvarflowiri, spi:definedOn, ?setarciri)` states the required connection of flow-individuals and arc-set individuals for indexing.

- `?unionsetoperandsarc`: If `?setarciri` query results are defined as union-sets, the `SPI#hasOperand` objects can be queried by this variable and the respective query patterns.

- `?unionsetoperandnames`: E-names of the latter union-set operands.

- `?mathprops`: Query for all mathematical properties of the `Net#SingleCommodityFlowCollection`-individuals.

The reader may have observed that the query excerpt given in Figure 5.16 is not satisfiable for the specifications made in the considered ontology represented model. The unsatisfiability of the first `Where`-block results from three reasons:

1. The individual `FlowVar` is not of type `OM#Parameter` but of type `OM#Variable`.
2. A value for the data property `SPI#NonInstantiable` as required by the query atom `PropertyValue(?parvarflowiri, spi:NonInstantiable, ?nonInstantiable)` is not defined.
3. The individual `ArcSet` to be retrieved for the query variable `?setarciri` is not a `SPI#UnionSet`.

In order to make the whole query satisfiable for all possible specifications including the given case, the respective query patterns should be put into blocks that are marked as optional. Unfortunately, the SPARQL-DL implementation used here does not provide such a nested form of an `OPTIONAL` construct as it is, e.g, known from SPARQL. To that end, one has to make explicit distinctions of conjuncted groups of query atoms by the construct `OR WHERE`. The construct `OR WHERE` can only be applied on the top level of the query and a nesting into blocks of query patterns is to the best of the author's knowledge from the documentation of SPARQL-DL and the query engine implementation not possible. For the three cases mentioned above and the additional need to allow for a missing of mathematical property specification one gets $2^4 = 16$ different groups of conjuncted query atoms (,see, the digital material for the complete query). The matching pattern in the case of the given specifications reads as in Figure 5.17.

The Query Results:
The query results in XML form are given in Figures 5.18 and 5.19. The <`results`>-part of the document is made up of three <`result`>-blocks. Every block contains the e-name of the single data entity as a binding of the query variable `"parvarflow"`, the arc-set e-name (binding name `"setarc"`) and the typing as a variable (binding name `"omEntityType"`). The two extra blocks result from the specification of the two mathematical properties `OM.MProps#ZeroLowerBoundInd` and `OM.MProps#IntegralityPropertyInd` that are to be transformed into the subexpressions `>=0` and `integer` in AMPL.

The Derivation Implementation:
As a next step, the query results can be sent to the composed AML Derivation Service and by processing the entity type implementation IRI an internal call to a derivation service for the type `Net#SingleCommodityFlowCollection` can be executed with the XML query results as input. In this example, the AML Derivation Service and its invoked statement type derivation services produce AMPL statements in an XML format that was developed for this thesis. The XSD defining valid AMPL statements can be looked up in the accompanying material in digital form. It is important to mention that the prototypical XSD does not support all language features of AMPL. The features that are supported can roughly be described as the standard expressions for defining abstract AMPL models and will

```
<?xml version="1.0" encoding="UTF−8"?>
<sparql xmlns="http://www.w3.org/2005/sparql−results#">
        <head xmlns="">
                <variable name="mathprops" />
                <variable name="setarc" />
                <variable name="parvarflow" />
                <variable name="omEntityType" />
        </head>
        <results xmlns="">
                <result>
                        <binding name="mathprops">
                                <uri>http://www.semanticweb.org/florianstapel/
                                    ontologies/2015/6/OM.MProps.owl#
                                    IntegralityPropertyInd</uri>
                        </binding>
                        <binding name="setarc">
                                < literal >Arcs</literal>
                        </binding>
                        <binding name="parvarflow">
                                < literal >ArcFlow</literal>
                        </binding>
                        <binding name="omEntityType">
                                <uri>http://www.semanticweb.org/florianstapel/
                                    ontologies/2014/8/OM.owl#Variable</uri>
                        </binding>
                </result>
```

Figure 5.18.: Query results for SingleCommodityFlowCollection Type - Part 1

be illustrated by the examples in the remainder of this thesis. An important construct that is not supported so far is given by *logical expressions*.

A derivation implementation for `Net#SingleCommodityFlowCollection` has to provide the following functionality:

- Derive a statement for every individual of the entity type.
- Start the statement with the keyword `var` or `param` depending how the corresponding individual is used in the model.
- Construct the indexing with a single allowed arc-set as an index, but check whether the related arc set is built as a union of other sets.
- Check for mathematical property definitions of every individual and build the respective subexpressions.

```
            <result>
                    <binding name="mathprops">
                            <uri>http://www.semanticweb.org/florianstapel/
                                ontologies/2015/6/OM.MProps.owl#
                                ZeroLowerBoundInd</uri>
                    </binding>
                    <binding name="setarc">
                            < literal >Arcs</literal>
                    </binding>
                    <binding name="parvarflow">
                            < literal >ArcFlow</literal>
                    </binding>
                    <binding name="omEntityType">
                            <uri>http://www.semanticweb.org/florianstapel/
                                ontologies/2014/8/OM.owl#Variable</uri>
                    </binding>
            </result>
            <result>
                    <binding name="setarc">
                            < literal >Arcs</literal>
                    </binding>
                    <binding name="parvarflow">
                            < literal >ArcFlow</literal>
                    </binding>
                    <binding name="omEntityType">
                            <uri>http://www.semanticweb.org/florianstapel/
                                ontologies/2014/8/OM.owl#Variable</uri>
                    </binding>
            </result>
    </results>
</sparql>
```

Figure 5.19.: Query results for SingleCommodityFlowCollection Type - Part 2

```
Algorithm 1: deriveStatements(XML queryResults)
   Data: XML queryResults
   Result: XML resultStatementsforType
 1 Initialize resultDocument with a <root>-tag ;
 2 Load queryResults into a suited object representation data ;
 3 for model entity ent do
 4     Determine the entitykind as var or param from data ;
 5     Insert a <statement>-tag with respective <mod_entity> child and a
       <var> or <param> statement-tag depending on entitykind under the
       <root>-tag ;
 6     Insert <VAR> or <PARAM>-tags under the current <var> or <param>
       statement-tag and insert the e-name of ent that was retrieved from data in a
       <LITERAL> tag that resides under the current <var> or <param>
       statement-tag ;
 7     Depending on the results in data: Insert an indexing with the e-name
       identifier as a result of ?setarc in conjunction with ent under the current
       <var> or <param> statement tag. If the single indexing argument is a
       union-set, instantiate a respective union of unionset operands ;
 8     Insert the mathematical property subexpressions depending on the results in
       data under the current <var> or <param> statement-tag ;
 9     Insert a <SEMIKOLON>-tag with node-content ";" under the current <var> or
       <param> statement tag ;
10 end
11 return resultDocument ;
```

Figure 5.20.: Abstract algorithmic description for a derivation implementation of `Net#SingleCommodityFlowCollection`.

An abstract algorithmic description for the derivation implementation can be found in Figure 5.20. Since the concrete implementation contains further details, the reader is referred to the complete source-code in the digital material accompanying this thesis.

XML Document containing the Statement(s):
The xml document resulting from the derivation process for the type `Net#SingleCommodityFlowCollection` is given in Figure 5.21. Starting with the nested tags `<root>, <statement> and <mod_entity>`, a `<var>` statement has been constructed. The document's leave nodes contain the important identifiers for model entity name and indexing indentifier names that were returned form the query. The mathematical properties have been translated into respective subexpressions. The whole file can be validated against the AMPL XSD in order to

guarantee syntactical correctness for AMPL.

The example of this subsection shall now be concluded with a collection of all requirements for general query and derivation implementations in the following subsection. This will extend the specific requirements for the example variable/parameter type presented above by requirements for deriving sets, goals and (groups of) constraints.

5.4.2. Requirements and Design Considerations for Queries and Derivation Implementations

This subsection shall illustrate the requirements and design considerations underlying queries and derivation implementations for reusable model entity types. All features that are part of the ontology versions accompanying this thesis, as well as of the demonstrator application to be presented in Chapter 8, will be discussed.

The requirements on queries and derivation implementations have to be covered when implementing a new ontological type. They can be stated as follows:

1. All queries and derivation implementations, independent of wether they are for data entity or formulation entity types have to guarantee the following functionality:
   a) Retrieval of e-names: The values of `OM#EName` property axioms have to be extracted from the ontology representations and processed into identifiers for the respective entities they refer to.
   b) Valid statements: All statements constructed by the derivation process have to be valid for the target AML XSD describing the XML representations of AML statements.
   c) Treat all instantiable individuals: All individuals of the entity type currently processed have to be considered for instantiation into valid statements.

2. Besides the latter general requirements, certain functionality has to be guaranteed for special entity types that are restricted to either sets, parameters and variables or formulations. For the data entity types that are suclasses of `SPI#Set` the following is required:
   a) Check for and deal with axiom definitions of `SPI#NonInstantiable true`: Set-entities that define this property must not be transformed into AML statements.
   b) Set-expressions for axioms with `SPI#WithinCartproductTwoSame`: Process specifications made with the latter property into respective AML statements that cover the respective set relations.

```
<?xml version="1.0" encoding="UTF−8"?>
<root>
        <statement>
                <mod_entity>
                        <var>
                                <VAR>var</VAR>
                                <id>
                                        <LITERAL>ArcFlow</LITERAL>
                                </id>
                                <indexing>
                                        <LCURLBRACKET>{</LCURLBRACKET>
                                        <sexpr_list>
                                                <sexpr>
                                                        <id>
                                                                <LITERAL>Arcs</
                                                                    LITERAL>
                                                        </id>
                                                </sexpr>
                                        </sexpr_list>
                                        <RCURLBRACKET>}</RCURLBRACKET>
                                </indexing>
                                <v_attributes>
                                        <v_attribute>
                                                <RANGEEXP>
                                                        <GT>&gt;=</GT>
                                                        <NUMBER>0.0</NUMBER>
                                                </RANGEEXP>
                                        </v_attribute>
                                        <COMMA>,</COMMA>
                                        <v_attribute>
                                                <INTEGER>integer</INTEGER>
                                        </v_attribute>
                                </v_attributes>
                                <SEMIKOLON>;</SEMIKOLON>
                        </var>
                </mod_entity>
        </statement>
</root>
```

Figure 5.21.: SingleCommodityFlowCollection individual statement(s) as XML

   c) Union-set operands in set espressions: Derive objects of `SPI#WithinCartproductTwoSame` axioms into *union* statements of respectively specified operands in case of a `SPI#UnionSet`.

3. Queries and derivation implementations for data entities that are in contrast to the latter subclasses of `SPI#ParameterCollection` have to provide the following special functionality:

   a) Distinguish variables from parameters and instantiate respective statements with AML keywords and syntactical structure.

   b) Determine and process all objects of `SPI#definedOn` axioms that are prescribed by the data entity type. Processing in this case means the generation of a respective subexpression in the indexing part of the *parameter* or *variable* statement.

   c) Union-set indexing: As comparable to set-expressions instantiate set-objects of `SPI#definedOn` axioms into *union* statements of respectively specified operands in case of a `SPI#UnionSet`.

   d) Mathematical properties: Process mathematical property specifications for `OM#dataMProp` into AML representations. Within this thesis, only the code snippets `integer, binary, >= 0 and >0` for the target AML AMPL are supported.

   It also has to be mentioned that the `SPI#NonInstantiable` property is currently only supported for set-types.

4. Finally, formulation types also have their own specific requirements on the implementation of queries and statement derivations:

   a) Determine and process all objects of `OM#requires` axioms that are prescribed by the data entity type. Processing in this case means either a respective subexpression in the indexing part of the *goal* or *constraint* expression or a direct treatment in the exression part describing the formulation entity.

   b) Union-set indexing: As comparable to set-expressions instantiate set-objects of `OM#requires` axioms into *union* statements of respectively specified operands in case of a `SPI#UnionSet`.

   c) Guarantee consistency in index order: Indices of constraint-groups, parameters and variables should be used in the correct order as intended by the parameters and variables. E.g., the order of arc-set indices shall not be interchanged.

   d) Support of `OM#ExpressionEntity` specifications: Examples and explanations for this requirement will be given in Section 6.1.3. The functionality provided intuitively allows for the implementation and specification of formulation types with a dynamic number of summation-operations. E.g., a variable number of variable families can be instantiated into multiple summation expressions that are themselves put together in an outer sum.

Design Principles for Queries:
The requirements above lead to specific design considerations for the SPARQL-DL queries that are in one-to-one relation to the model entity types. To that end the following aspects of a query design are proposed:

1. The following can be stated for every query implementation:
    a) An implicit and an explicit query variable for the entity individuals and their e-name has to be introduced. Respective query atoms determine individuals of the entity type and relate them to the value of the `OM#EName` property. This corresponds to the requirements 1a and 1c from above.
    b) All query patterns that are optional or in disjunction, i.e., for which matching variable allocations may not exist or where types may vary have to be treated suitedly. A matching group of query patterns has to be introduced for every intended possible query result. The groups are to be disjuncted by the `OR WHERE` keyword.
2. The generation of valid statements from requirement 1b is an issue for the derivation implementations. Besides the general specifica of queries, the following parts of query implementations are specific to either set, parameter or variable or formulation entities. For the data entity types that are suclasses of `SPI#Set` the following variables and query atoms shall furthermore be introduced:
    a) A `SPI#NonInstantiable-PropertyValue` query atom: For checking whether a respective property axiom is defined with value "true" such that a respective statement will not be instantiated, see requirement 2a.
    b) A query variable for `SPI#WithinCartProductTwoSame` axiom objects and their e-name as well as respective query atoms, see requirement 2b.
    c) Two query variables for every possible union-set operand and their e-names. Respective query atoms, see requirement 2c.
3. Queries for data entities that are in contrast to the latter subclasses of `SPI#ParameterCollection` contain following extra variables and patterns:
    a) A variable for determining a type of the data entity individual as eiter `OM#Variable` or `OM#Parameter` and respective query atoms. This concerns requierement 3a

   b) Two variables for every object type required by a `SPI#definedOn` class restriction. One variable for the individual's IRI and one for the e-name. Query atoms for the `SPI#definedOn`-property, the `OM#EName` property and the ontology type of the object, see requirement 3b.
   c) Two query variables for every possible union set operand and their e-names. Respective query atoms, see requirement 3c.
   d) Mathematical properties: Add a single query-variable for mathematical property individuals (IRIs) and a single query atom for the property `OM#dataMProp`. The concrete type of the mathematical property can later be identified by the individual for the limited amount of properties supported within this thesis. This concerns requirement 3d.

4. The formulation types then add the following variables and patterns:
   a) Two variables for every object type required by a `OM#requires` class restriction. One variable for the individual's IRI and one for the e-name. Query atoms for the `OM#requires`-property, the `OM#EName` property and for checking the ontology type of the object. See requirement 4a.
   b) Two query variables for every possible union-set operand and their e-names. Respective query atoms. See requirement 4b.
   c) In case of `OM#ExpressionEntity` specifications: Respective variables and query patterns.

   The *consistency of indice order* as stated in requierement 4c is an issue for the derivation implementation.

Derivation Implementations:
Finally, the general ingredients of formulation type derivation implementations can be described. A pseudo-code for a derivation of statements with the type `Net#SingleCommodityFlowCollection` has already been given in Figure 5.20. Another example for a balance-formulation type will follow in the next subsection. The design of implementations described here has already been realized by classes and a demonstrator (Chapter 8) in Java 7. Every type is implemented in a class for its derivation that provides the single public method `deriveStatements(...)`. The classes are invoked by the imaginary AML Derivation Service via reflection. A full service realization was not provided but a design that fulfills the signatures given within the beginning of this section and can in principle be deployed as a service. For further details on the installation, usage and design of the demonstrator application, the reader is refered to Chapter 8. The following description of the implementation design is partially specific to the target demo language AMPL.

The issues concerned with the design of derivation implementations can be stated and grouped as follows:

1. All statement derivations implement the following tasks:
    a) Generate the initial document with a <root>-tag.
    b) Implement a loop over every model-entity of the respective type that is contained in the model. Instantiate the respective tags <mod_entity> and <statement> for model entities and statements. Insert the respective statement-tag for the `OM`-entity type, e.g., <set> or <param>. Every statement is closed by a ";"-Symbol with a respective tag.

    The points from above are especially of importance for the valid statements requirement 1b. All following computations happen inside the loop over model entities.

2. The points in the next enumeration are then implemented for the types that are subclasses of `SPI#Set`:
    a) Finish the current loop iteration without a statement in case of a `SPI#NonInstantiable true` specification for the model entity under consideration.
    b) Determine the e-names of the `SPI#WithinCartProductTwoSame` operands and build respective set-expressions.
    c) Instantiate union-set statements in case of union-set operands.

3. For parameters and variables the following is considered:
    a) Check for type declaration of model entity as `OM#Parameter` or `OM#Variable` and insert a respective statement tag.
    b) Instantiate the `SPI#definedOn` sets as set-expressions in an indexing.
    c) Instantiate union-sets with union-set expressions.

4. For formulation-types one has the following:
    a) Instantiate the indexing tag in case of constraint-types.
    b) Define and later use dummy indices for the indexing sets and those of operators in expresssions in a consistent way. See requirement 4c.
    c) Instantiate union sets with union set expressions.
    d) Insert a colon ":" symbol with respective tags.
    e) Insert the expressions for goals and constraints with dummy-indices and union set declarations for the sum-operator.
    f) In case of required `OM#ExpressionEntity` individuals: Insert the respective generic number of expressions and tie them together, e.g., by an outer sum.

The latter considerations shall now be concluded by another example.

5.4.3. A Balance-Type Example of a Statement Derivation

The considerations of the preceding subsections shall now be illustrated by another example of a statement derivation for the single entity type `NetForms#SingleCommodityBalance`. This example is especially intended to extend the flow-variable declaration example from Subsection 5.4.1 by the aspects concerned with deriving statements from *formulation entity* specifications. The target statement

```
  s.t.  MassBalance { i in Nodes }:
 sum{ (i,j) in Arcs }(ArcFlow[i,j])
-sum{ (j,i) in Arcs }(ArcFlow[j,i]) = SupplyDemand[i];
```
(5.6)

is a part of the min-cost-flow model of Section 5.2. The relevant parts of the ontology specifications are given in an extra Figure 5.22.

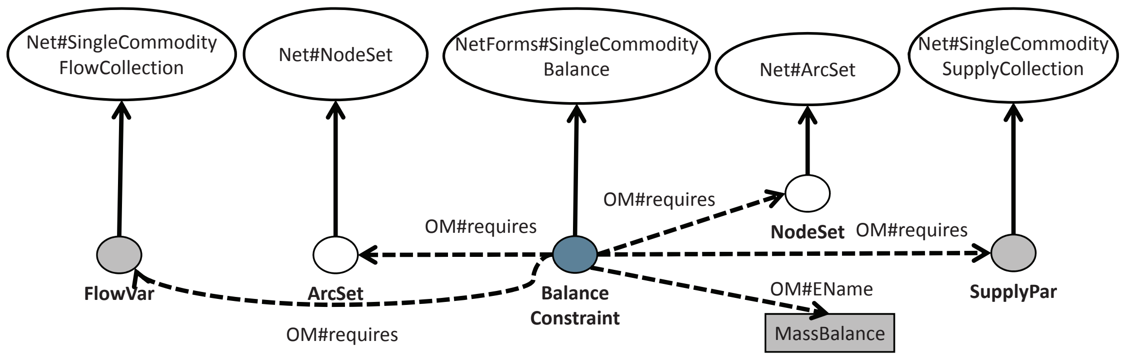


Figure 5.22.: Ontology excerpts for deriving the mass-balance statement

The SPARQL-DL query which is given in Figures 5.23 - 5.25 introduces the following variables and query patterns according to the design considerations from the last subsection:

- `?forment`: This variable allows to query the e-names of all indivudals of `NetForms#SingleCommodityBalance`. The respective variable querying the ontology individuals of the type is `?formentiri`. The conjuncted query-patterns `PropertyValue(?formentiri, om:EName, ?forment)`, `Type(?formentiri, formont:SingleCommodityBalance)` buid-up the respective part of the query.

```
PREFIX om: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#>
PREFIX mdo: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/MDO−
    Network.owl#>
PREFIX formont: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM−
    Network−Formulations.owl#>
PREFIX spi: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#>
PREFIX mprop: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.
    MProps.owl#>
SELECT ?forment ?setarc ?unionsetoperandsarc ?unionsetoperandnamesarc ?setnode ?
    unionsetoperandsnode ?unionsetoperandnamesnode ?varparsupply ?varparflow
WHERE
{
PropertyValue(?formentiri, om:EName, ?forment),
Type(?formentiri, formont:SingleCommodityBalance),
PropertyValue(?formentiri, om:requires, ? setarciri ),
Type(?setarciri , mdo:ArcSet),
PropertyValue(?setarciri , om:EName, ?setarc),
Type(?setarciri , spi:UnionSet),
PropertyValue(?setarciri , spi:hasOperand, ?unionsetoperandsarc),
PropertyValue(?unionsetoperandsarc, om:EName, ?unionsetoperandnamesarc),
PropertyValue(?formentiri, om:requires, ?setnodeiri),
Type(?setnodeiri, mdo:NodeSet),
PropertyValue(?setnodeiri, om:EName, ?setnode),
Type(?setnodeiri, spi:UnionSet),
PropertyValue(?setnodeiri, spi:hasOperand, ?unionsetoperandsnode),
PropertyValue(?unionsetoperandsnode, om:EName, ?unionsetoperandnamesnode),
PropertyValue(?formentiri, om:requires, ?varparsupplyiri),
Type(?varparsupplyiri, mdo:SingleCommoditySupplyCollection),
PropertyValue(?varparsupplyiri, om:EName, ?varparsupply),
PropertyValue(?formentiri, om:requires, ? varparflowiri ),
Type(?varparflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?varparflowiri, om:EName, ?varparflow)
} OR WHERE
```

Figure 5.23.: Query for SingleCommodityBalance - Part 1

```
{
PropertyValue(?formentiri, om:EName, ?forment),
Type(?formentiri, formont:SingleCommodityBalance),
PropertyValue(?formentiri, om:requires, ? setarciri ),
Type(?setarciri , mdo:ArcSet),
PropertyValue(?setarciri , om:EName, ?setarc),
Type(?setarciri , spi:UnionSet),
PropertyValue(?setarciri , spi:hasOperand, ?unionsetoperandsarc),
PropertyValue(?unionsetoperandsarc, om:EName, ?unionsetoperandnamesarc),
PropertyValue(?formentiri, om:requires, ?setnodeiri ),
Type(?setnodeiri, mdo:NodeSet),
PropertyValue(?setnodeiri, om:EName, ?setnode),
PropertyValue(?formentiri, om:requires, ?varparsupplyiri ),
Type(?varparsupplyiri, mdo:SingleCommoditySupplyCollection),
PropertyValue(?varparsupplyiri, om:EName, ?varparsupply),
PropertyValue(?formentiri, om:requires, ? varparflowiri ),
Type(?varparflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?varparflowiri, om:EName, ?varparflow)
} OR WHERE
{
PropertyValue(?formentiri, om:EName, ?forment),
Type(?formentiri, formont:SingleCommodityBalance),
PropertyValue(?formentiri, om:requires, ? setarciri ),
Type(?setarciri , mdo:ArcSet),
PropertyValue(?setarciri , om:EName, ?setarc),
PropertyValue(?formentiri, om:requires, ?setnodeiri ),
Type(?setnodeiri, mdo:NodeSet),
PropertyValue(?setnodeiri, om:EName, ?setnode),
Type(?setnodeiri, spi:UnionSet),
PropertyValue(?setnodeiri, spi:hasOperand, ?unionsetoperandsnode),
PropertyValue(?unionsetoperandsnode, om:EName, ?unionsetoperandnamesnode),
PropertyValue(?formentiri, om:requires, ?varparsupplyiri ),
Type(?varparsupplyiri, mdo:SingleCommoditySupplyCollection),
PropertyValue(?varparsupplyiri, om:EName, ?varparsupply),
PropertyValue(?formentiri, om:requires, ? varparflowiri ),
Type(?varparflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?varparflowiri, om:EName, ?varparflow)
} OR WHERE
```

Figure 5.24.: Query for SingleCommodityBalance - Part 2

```
{
PropertyValue(?formentiri, om:EName, ?forment),
Type(?formentiri, formont:SingleCommodityBalance),
PropertyValue(?formentiri, om:requires, ?setarciri),
Type(?setarciri, mdo:ArcSet),
PropertyValue(?setarciri, om:EName, ?setarc),
PropertyValue(?formentiri, om:requires, ?setnodeiri),
Type(?setnodeiri, mdo:NodeSet),
PropertyValue(?setnodeiri, om:EName, ?setnode),
PropertyValue(?formentiri, om:requires, ?varparsupplyiri),
Type(?varparsupplyiri, mdo:SingleCommoditySupplyCollection),
PropertyValue(?varparsupplyiri, om:EName, ?varparsupply),
PropertyValue(?formentiri, om:requires, ?varparflowiri),
Type(?varparflowiri, mdo:SingleCommodityFlowCollection),
PropertyValue(?varparflowiri, om:EName, ?varparflow)
}
```

Figure 5.25.: Query for SingleCommodityBalance - Part 3

- `?setarc`: The sums of ongoing and outgoing flows in the statement are indexed over arc sets whose e-names are determined for the `?setarc` variable. The respective variable `?setarciri` for the entity IRI is implicit. The query atom `PropertyValue(?formentiri, om:requires, ?setarciri)` specifies the need for a "requires"-axiom concerning balance constraint individuals and arc set individuals.

- `?unionsetoperandsarc`: If `?setarciri` query results are defined as union sets, the `SPI#hasOperand` objects can be queried by this variable and the respective query patterns.

- `?unionsetoperandnamesarc`: E-names of the latter union set operands.

- `?setnode`: The balance constraints are indexed over node sets whose e-names are determined for the `?setnode` variable. The respective variable `?setnodeiri` for the entity IRI is implicit. The query atom `PropertyValue(?formentiri, om:requires, ?setnodeiri)` specifies the need for a respective "requires"-axiom.

- `?unionsetoperandsnode`: If `?setnodeiri` query results are defined as union-sets, the `SPI#hasOperand` objects can be queried by this variable and the respective query patterns.

- `?unionsetoperandnamesnode`: E-names of the latter union-set operands.

```
Algorithm 2: deriveStatements(XML queryResults)
  Data: XML queryResults
  Result: XML resultStatementsforType
1 Initialize resultDocument with a <root>-tag ;
2 Load queryResults into a suited object representation data ;
3 for model entity ent do
4     Insert a <statement>-tag with respective <mod_entity> child and a
      <constraint> statement-tag under the <root>-tag ;
5     Insert a <SUBTO>-tag with respective node-content and a <LITERAL> tag
      containing the e-name of ent that was retrieved from data as child nodes of
      the actual <constraint>-tag ;
6     Insert the indexing for the results of Net#NodeSet e-name results of the query
      variable ?setnode that are in conjunction with ent under the current
      <constraint>-tag. The results are to be retrieved from data. If the single
      indexing argument is a union set, generate a respective union of unionset
      operands. The indexing has to declare an index "i" for referencing the nodes
      in the inner expressions ;
7     Insert a <COLON>-tag with node-content ":" under the current
      <constraint>-tag ;
8     Insert a <cexpr>-tag for the constraint expression under the current
      <constraint>-tag ;
9     Retrieve the results of ?setarc and ?varparflow (from data) as e-names of
      Net#ArcSet and Net#SingleCommodityFlowCollection individuals that are
      in conjunction with ent ;
10    Generate the content of the left-hand side of the equality: Insert an outer
      expression with a "minus" operator for the two sums of flow variables over the
      arc-set under the current <cexpr>-tag. Insert the two individual sums as
      expression-childs of the outer sum in the same way where the difference lies in
      the interchanged subscripts "i,j" and "j,i" for the network arcs. The
      identifiers for arcs and flows were retrieved in the last steps ;
11    Insert a <EQUAL>-tag with node-content "=" under the current <cexpr>-tag ;
12    Insert the right hand side content of the equality under <cexpr>: Retrieve
      the e-name of the required Net#SingleCommoditySupplyCollection as a
      result of the query-variabe ?varparsupply for the current entity from data
      and insert a respective expression-tag with the identifier and a subscript with
      the single index "i" ;
13    Insert a <SEMIKOLON>-tag with node content ";" as a child node of the
      current <constraint>-tag ;
14 end
15 return resultDocument ;
```

Figure 5.26.: Abstract algorithmic description for a derivation implementation of NetForms#SingleCommodityBalance.

- `?varparsupply`: The query atom `PropertyValue(?formentiri, om:requires, ?varparsupplyiri)` specifies the need for a respective "requires"-axiom. `?varparsupply` with a respective query atom is the query variable for the respective e-name.
- `?varparflow`: The query atom `PropertyValue(?formentiri, om:requires, ?varparflowiri)` specifies the need for a respective "requires"-axiom. `?varparflow` with a respective query atom is the query variable for the respective e-name.

The matching group of query atoms in this case is the last block of query atoms from Figure 5.25 that does not require sets to be defined as `SPI#UnionSet`.

As the query results would blow up the description of this example, the interested reader is referred to the digital material in this case.

The derivation procedure for the type `NetForms#SingleCommodityBalance` in an abstract algorithmic form is given in Figure 5.26. According to the design considerations from the last subsection the procedure iterates over every model entity to generate a respective <`constraint`> statement. The e-names of individuals `OM#requires`-axioms are inserted as identifiers into respectively build expressions.

The XML document resulting from the derivation can be found in the accompanying digital material.

5.4.4. Algebraic Model Derivation Process

Until now it has been explained how to derive AML statements for all individuals of a single ontological entity type. This subsection describes the process to transform a whole ontology represented abstract optimization model into an AML represented abstract optimization model. The input to the process is an ontology represented optimization model in OWL 2 which is imports-closed, i.e., for which all directly or indirectly imported vocabularies have been loaded into the model as an ontology. Furthermore, the following validations should have taken place before the process is started:

- Validations by Model Editor: Model editors should validate ontology models before the derivation. Are the type definitions for all individuals as intended? Are the additional type definitions such as `OM#Set` or `OM#Variable` present in the model? Do distinct individuals refer to distinct model entities and are references to the same individual really intended as such?
- Ontology Validation: The model can be validated ontologically by a resoning procedure. This can reveal contradictory specifications, e.g., missing axioms for specifying related data entities or non-matching types of such stated entities

```
Algorithm 3: transformOntologyModel2AML(OWL model)
   Data: OWL model
   Result: AML-File modelFile
1  Retrieve all entity types that are used in the model: There has to be at least one
   individual with that type and the type must have been directly specified (no
   inferred type axioms ) ;
2  Sort the entity types partially by grouping them into containers for
   simple-parameters, sets, parameters, variables, goals and constraints ;
3  Initialize the resultString for the target AML file as empty ;
4  for every entity type in order do
5      Resolve the query location from the annotated IRI and process the query on
       model. Store the resuls in XML queryResult ;
6      Call deriveStatements(queryResult,typeIRI) on the AML Derivation
       Service and store the XML statementResults. The typeIRI is internally
       known to this method ;
7      Extract the text-node content of the statementResults into a local variable
       currentResultString ;
8      Append currentResultString to resultString and add a line-break;
9  end
10 Generate the modelFile of the target AML format from the resultString ;
11 return modelFile ;
```

Figure 5.27.: Algebraic model derivation process

according to the class restrictions in the vocabulary. Reasoners also provide capabilities for debugging ontologies in terms of localizing these contradictions. It is important that inferred type axioms are not stored in the ontology model (see the following explanations of the process).

Figure 5.27 specifies the process from an algorithmic perspective. The derivation process itself is an iterated application of the derivation of all individuals for an entity type where the latter types are retrieved from the ontolgy model by the aid of the specified type axioms (therefore no inferred type axioms). The types and their individuals are sorted in the order of non-dimensional parameters, sets, parameters, variables, goals and constraints. Every entitity type is processed in terms of the query processing and a call to the AML Derivation Service. Text node content will be extracted from the XML documents and written into a string that can be used to build the content of an AML file.

The following remarks are important:

- It is assumed that the query IRIs can be extracted from the vocabularies used in the ontology model and resolved to the unique and correct queries for the entity types.

- The AML Derivation Service and correct IRIs of implementations for entity types, i.e., especially ones that yield statements of the target AML, are fixed and known to the procedure as it has already been described in the beginning of this section.

- It is optional to validate the resulting statements against an XML schema for the target AML. This could be done either within the loop (after step 6.) or, if all `XML statementResults` are stored in a separate loop, after step 9. The demonstrator application that is presented in Chapter 8 provides means to validate the results after the derivation of the model as a string. XML validation can, e.g., be helpfull in case of mistakes in AML derivation implementations.

- The whole process does not contain a treatment of errors for reasons of simplicity. The process is designed to demonstrate the feasibility of the derivation approach in the entity type and service context. As such, the process is implemented in the demonstrator application presented in Chapter 8.

- The process terminates respective the operations on ontologies: The retrieval of all *directly* specified types can be performed in finite time by looping over the finite number of model individuals and looking for such explicit definitions without a need for further reasoning. The grouping into containers can basically be seen as a standard OWL 2 resoning task that queries for individuals of a fixed class and is decideable since all model and vocabulary definitions in use fall under the OWL DL profile. Furthermore, SPARQL-DL is decideable and as such are the queries. The derivation implementations with their design as presented in Subsection 5.4.2 will terminate. Further considerations of performance issues are not part of this thesis.

- Derived models will be complete in the sense that every model entity declared in the ontology model will be derived into a respective statement or statements in case of types that represent groups of AML declarations.

- The derived algebraic models will be correct so far as that every statement is syntactically correct in terms of the AML syntax and semantically correct in terms of the intended mathematical exressions iff the implementations for the type in use are correct and the ontology specifications are free of contradictions according to the class-restrictions imposed in the vocabulary. In case the AML derivation implementations are not trusted, means for XML validation can be used. Identifiers for model entities will be consistent according to the dependencies specified in the ontology model. The correct order of indices is guaranteed by correct formulation type derivation implementations. The correct order of declarations in the resulting AML document is not guaranteed as the next remark states.

- A topologically sorted order of data declaring statements in terms of the dependencies they have on other identifiers is not guaranteed by the process. As long as there are no cycles introduced in the definitional dependencies, the order could be retrieved by an additional topological sort (this is also a standard method for models in Structured Modeling). Some AMLs and their respective environments may have suitedly ordered declarations as a requirement. E.g., for `.mod` in AMPL, the declarations have to be sorted by the occurence of identifiers. What can be guaranteed is the order of blocks for model entity declarations. E.g., set-statements will be stated before parameter declarations.

5.4.5. Conclusion of Statement Derivation and Structure

This subsection gives a short conclusion of the entity type and statement derivation approach as it has been presented so far. Based upon the vocabulary and ontology specifications from Section 5.3, this section explained how the structure of abstract model statements with their respective identifiers, semantic types of identifies and expressions can be decoupled from a conceptualization of a reusable model entity type. The special feature of this structure is the definition of data model independent formulation types. It was shown how ontology represented abstract optimization models can be transformed into algebraic modeling language models. The process as a whole could generate complete and syntactically correct algebraic models, but the semantical correctness depends on the ontology specifications performed by humans and the correctness of the implementations. The topological order of statements in terms of the definitional dependencies of identifiers was neglected but can be achieved by a topological sort in case of non-cyclic definitions. Another important aspect is the service-orientation for the derivation functionality provided as an AML Derivation Service. Though the considerations of this thesis were simplified to a central and known derivation service, the possibilities for flexible compositions of algebraic model derivations, e.g., for representing a fixed abstract model in a fixed language, are given by the presented framework.

At this point it is not possible to discuss the full capabilities for reusability of entity types in the approach. Nonetheless, it can be recapitulated what has been demonstrated and mentioned so far:

- It was demonstrated that it is possible to reuse formulation types with different identifiers and semantic data types. This included the usage of subtypes for required mdo-types.

- The indexing of the entity types is flexible in the point that one can not only generate static sets but also dynamic unions of sets in an indexing expression. This has so far only been mentioned but will be demonstrated by the examples of the next chapter.
- Data entities which are not sets can be reused as variables or parameters either.
- What has not been demonstrated so far but will be shown in Subsection 6.1.3 is the definition of formulation types with generic summation expressions. E.g., these type can be reused with one, two or *-many inner summations of different structure.

In the remainder of this chapter, issues concerned with the two missing interconnections of the three model representation layers from Figure 5.2 in Section 5.1 will be discussed. Namely, the reengineering of AML represented models into the ontology representation and the generation of semantic service descriptions for model instantiation services out of ontology represented models. This will highlight further aspects and capabilities of the ontology vocabulary.

5.5. Reengineering of AML Models

Besides the means to bring an ontology represented optimization model into an AML representation there also should be a highly automated process to reengineer AML models into the ontology representation. This has been visualized in Figure 5.2 of Section 5.1 by the arrow "Reengineering". Reengineering of AML models is of importance for model formulation and reuse, since a large number of models that are defined in AMLs is available. Furthermore, the creation of abstract optimization models within AMLs seems to be a very intuitive process for human modelers. The detailed investigation of AML Reengineering is not part of this thesis. To that end, only an outline for illustrating the feasibility is given. The following steps may be performed when reengineering an AML model into the ontology representation. Steps that might be automated are marked as such:

1. Parse AML Model: Parse the model with a parser for the AML and obtain a valid parse tree in a tool for AML reengineering. This step can be automated.
2. Extract IDs: Perform a visit of the parse tree in order to extract the identifiers of model entities and assign iris to these identifiers in an automated way.
3. Specify ID Semantic Types: This step requires the user to interact. Since AMLs naturally provide no semantic types, suited ontology vocabularies, e.g., for goal and constraint types as well as meta domain ontologies have to be loaded and semantic types have to be specified. The resulting knowledge is to be stored in an ontology.

4. Build Statement Ontologies: With the parse tree and the semantic type specifications of the unique model individuals, first ontology representations of the models' statements may be built in an automated fashion. In order to increase the performance of the following step one may build an ontology explaining the tree-structure for every statement of the chosen AML. These structures can be generated automatically out of the parse tree and the knowledge of the semantic types of the unique model entity identifiers.

5. Conclude Relationships: Having the context of a parse tree should allow to apply specialized rule knowledge to conclude new relationships automatically. These relationships are the typical ontology model relationships such as the required data conceptualizations of goals and constraints or relations of sets.

6. Extract Parts: With the model structure explained on the tree ontologies for every statement one is likely to build a consistent ontology optimization model by a union of the individual model entity definitions. Before doing this, one should separate the statement representations from the parse-tree structure overhead in an automated way.

7. Build Single Model: Finally, the model entities with their specifications can be merged into a consistent model in an automated way. Merge in this case is a simple ontology integration operation on the assertional level, by which the triple axioms for stating the model entities definitional dependencies are brought together. Basically it has to be taken care that refrences to the same objects are treated as such and do not lead to the generation of a copy. Semantic integration on the terminological level should not be required by the step as the types of model entities stem from unique and compatible vocabularies that were selected by the user in the prior steps of the process.

AML Reengineering helps to cover the requirement of **Service Enabling**. That is, one can support the generation of model instantiation services with the aid of AML Reengineering indirectly by providing first of all the ontology structure that can be used to generate a semantic service by techniques discussed in the next section.

Further investigations and considerations of AML Reengineering that are not documented within this thesis took place during the work on this thesis. Especially, an AML parser for a relevant part of the AMPL language was developed and parts of the steps two and three from above were implemented in a demonstrator application together with a student helper. Please contact the author of this thesis for further information.

5.6. Model Instantiation Services

The template-based semantic service composition of an optimization system as it was outlined in Section 5.1 searches and binds software services for solvers and further computing steps such as preprocessing and simulation. Before this step, the abstract optimization model has to be described and implemented as such a software service, called the *model instantiation service*, in order to allow for a suited matching of solver capabilities and model, as well as the matching of data interfaces. The model instantiation service provides the functionality to instantiate an abstract algebraic optimization model with instance data, such that the resulting instance can be sent to a solver service. To that end, an ontology represented model should be transformed into an abstract AML model by the methods from Subsection 5.4.4 in a first step. The implementation of the functionality to derive the abstract AML optimization model out of the ontology representation in the current state of the approach is, in contrast to the AML model instantiation functionality with data, not intended to be due to the service composition. When providing the AML derivation as a component, it can be hierarchically composed with the AML Derivation Service (Figure 5.14). The main aspect of such a software is that one can automatically generate correct abstract AML models out of ontology represented models. With a correct abstract AML model in a target language and a compiler for that language, e.g., AMPL running as a suited service, the data instantiation functionality can then be provided as a service in a step before the service composition of the optimization system. With the aforementioned steps, the "implementation" of the model instantiation service is also being retrieved in an automated fashion.
The aspect that requires discussion is the automated generation of semantic service descriptions out of the optimization model's ontology description such that semantic service composition for the target optimization system (recap. Section 5.1) can take place. The term service descriptions comes as a plural, since besides the model instantiation service, also solver services' required descriptions are influenced by this. This section is intended to present an approach for the generation of semantic service descriptions for model instantiation services and solvers.

The methods and examples from this section work towards the realization of an automated semantic model instantiation service generation and semantic service composition of optimization systems. The proposed methods will be illustrated and their capabilites will be discussed with the help of examples. Complete specifications and a demonstrator implementation of the methods from this section are not part of this thesis.

5.6.1. Obtaining Model Properties from the Ontology Representation

The first step for generating semantic service descriptions is an automated analysis of the ontology represented optimization model. The analysis step should reveal properties of the model such as linearity or the presence of integer variables that are important for the choice of a solution method. The inferred properties of an optimization model can afterwards be translated into predicates for the semantic description of model instantiation and solver services such that an automated composition of an optimization system guarantees the matching of solution method and model.

In this subsection, a method for concluding mathematical properties of ontology-represented optimization models will be presented. Since the considerations in this subsection are only intended to discuss the feasibility of an automated model analysis, the mathematical properties to analyze will be restricted to the following informal list: *Presence of binary and integer variables, linearity and general nonlinearity, convexity, continuity and differentiability* aspects. The latter aspects can be seen as important for discussing the principle choice of mathematical programming algorithms for the classes *LP, MIP, convex and nonconcex MINLP.* Further aspects such as the application of meta-heuristics to non-mathematical program representations of models, specialized algorithms for, e.g., network flow and quadratic programming, are neglected. They are to be considered in the future research.

The first step for the analysis of the ontology represented model is the extension of the mathematical property specifications that are included in the model after a user or an automated procedure has manipulated an ontology represented model. I.e., the given model should be complete in terms of all data and formulation entities being defined and related correctly. The specification of mathematical properties by a user such as bounds, integrality and binary-value requirements on parameters and variables has already been demonstrated with the aid of the mathematical properties vocabulary `OM.MProps` from Subsection 5.3.4. This vocabulary provides further classes and individuals for stating mathematical properties of goals and constraints in a model, such as the above mentioned linearity and continuity properties. Specifications for these properties can be added for every formulation entity. In principle, a user can also specify these properties by respective axioms making use of the `OM#formulationMProp`-property. But since the requirement of **Automated Modifications and Recomposition** from Section 4.2 demands for an automated recomposition of a system after changes in the model, this specification step should be automated. More specific, changes in the model, e.g., the declaration of an additional constraint or the change of a parameter to a variable that occurs in a product with another variable in the algebraic form of the model can also change the mathematical properties of goals and constraints, such that the inference process of these properties should be automated in the same way, as the other procedures to retrieve a model instantiation service and recompose a system are.

Concluding Mathematical Properties of the Formulation Entities:
To that end, a process has to be defined that has a given ontology represented

optimization model as input and returns an extended specification of the same model, where the formulation entities carry specifications of the mathematical properties under consideration. Completeness of this specification in terms of all considered mathematical properties would be a desireable property. Unfortunately, the process that will be presented now can not guarantee this property. In the end, this may lead to a classification of the overall model that is worse than reality. I.e., a model may be classified as nonlinear though it is linear. Since the methods in this section are only intended to serve demonstrational purposes, better knowledge bases and reasoning procedures should be a part of the future research.

An algorithmic specification of the mathematical property conclusion process for the formulation entities is given in Figure 5.28. The direct types of formulation entities are to be retrieved such that they can be processed iteratively. For every such formulation type a copy of the ontology model is generated that only contains the individuals that have the current formulation type as a directly specified type. Together with a set of SWRL-Rules that are locally valid for these "one-formulation type" ontologies, a reasoning procedure can be started. For properly designed rules, e.g., in a form that satisfies the *safety conditions* from Subsection 2.5.3, the reasoning procedure will terminate. The inferred axioms that are of the form `formulationEntity om:formulationMProp mprop:mathematicalProperty` are afterwards inserted into the complete optimization model.

The process, in similarity to the Algebraic Model Derivation Process from Figure 5.27, is based on a separated treatment of all individuals of a certain formulation type. This is especially due to the complexity of specifying valid rule knowledge for the inference of mathematical properties. In an ontology where formulation types can be subtypes of others it seems hard to specify globally valid rule knowledge that does not lead to contradictions. I.e., the criteria for one constraint type to be linear may differ for a subtype that extends the formulation of the latter by additional terms and the usage of further data entities. By now, the specifica of formulation subtypes have not been discussed in this thesis. To that end, a first illustrative example will be presented in the rule examples below. The intention of this first example is to discuss the issues concerned with locally valid rules for formulation types. The general modeling paradigms behind formulation types and especially formulation subtypes are a topic of Chapter 6.

The following aspects require commentation:

- Running Time and Performance: Performance and running time of methods for analyzing models and generating service descriptions are out of the scope of this thesis. What can be guaranteed for a suited rule design is termination:
- Termination: Since the number of individuals in a model is finite, as in the case of the AML Derivation process, the retrieval of the respective formulation types that are *explicitely* specified, takes finite time. The loop over the

formulation types performs a finite number of iterations. Inside the loop the reasoning step has to be considered. As already mentioned, this step will terminate for rules in a Horn-form that fulfill the *safety conditions* (see again Subsection 2.5.3). It will be explained how the rules considered fulfill these requirements in the examples below.

- Complete Property Inferences: It cannot be guaranteed that all mathematical properties under consideration are decided. Sufficient conditions, e.g., for the linearity of a type can be formulated. But these are not necessary and sufficient at once. As the examples below will illustrate, multiple rules that have sufficient conditions as antecedents can exist for a mathematical property. But in order to have a rule-set that "decides" the mathematical property for a given model, one would have to model all situations in rules that either yield the property or prove that it is not satisfied. For the formulation types presented so far in this thesis, cardinality restrictions with `exactly 1` were used and thus such a modeling seems possible by an explicit case enumeration. But as it will be shown in a counter-example below, reusable formulation types that require "+"-requires types specifications with cardinality restriction `min 1` exist and lead to problems. For this generous formulations, the open-world semantics, together with the capabilities of SWRL, are too limited.

Rule Examples:
The latter process will now be illustrated by some rule examples. For the demonstration purposes of rule-sets, two rules, where each rule can be seen as a part of a whole rule-set for the respective formulation type, will be presented. Both rules are concerned with the inference of linearity properties for individuals of the respective formulation types. The two types both represent balance constraints as they are, e.g., known from network flow problems. Their ontology specifications can be looked up in Figure 5.29. The first type `NetForms#SingleCommodityBalance` is already known from the previous examples of this chapter. The second type `NetForms#GainLossGeneralizedSingleCommodityBalance` is new and therefore requires an introduction:
Taking a look at Figure 5.29, one can see that `NetForms#GainLossGeneralizedSingleCommodityBalance` is defined as a subclass of `NetForms#SingleCommodityBalance`. This especially means that all class resstrictions are inherited to the subclass. To that end, a formulation entity of type `NetForms#GainLossGeneralizedSingleCommodityBalance` requires the same node- and arc-sets, supply-parameters and flow-(variable) families as the supertype, but also extends the definition by requiring an additional `Net#SingleCommodityGainLossMultiplier`. The specification gives a hint that the constraint group represented by the semantic type models a more generous situation as the supertype does. The min-cost-flow model from Figure 5.3 already included a

```
Algorithm 4: concludeandAddFormulationProperties(OWL model)
  Data: OWL model
  Result: OWL model extended by mathematical property axioms
1 Retrieve all formulation types that are used in the model: There has to be one
  individual at least with that type and the type must have been directly specified
  (no inferred type axioms ) ;
2 for every formulation type do
3     Generate OWL localModel as a copy of model where only the formulation
      entities / individuals that have a direct specification of formulation type as
      their type are contained ;
4     Load the locally valid annotated rule-set for formulation type into
      localModel ;
5     Apply the SWRL rule-set on localModel by a SWRL-reasoning capable
      reasoner ;
6     Add the inferred OM#formulationMProp-axioms to model ;
7 end
8 return model ;
```

Figure 5.28.: Adding mathematical properties of formulation entities to an OWL optimization model in an automated way

target statement for the type `NetForms#SingleCommodityBalance`. A target statement for the type `NetForms#GainLossGeneralizedSingleCommodityBalance` differs from the latter by an additional parameter `mu` that reflects an individual of the type `Net#SingleCommodityGainLossMultiplier`:

```
                        param mu { Arcs } > 0;
  s.t.  GeneralizedMassBalance { i in Nodes }:
             sum{ (i,j) in Arcs }(ArcFlow[i,j])
 -sum{ (j,i) in Arcs }(mu[j,i] * ArcFlow[j,i]) = SupplyDemand[i];
                                                                (5.7)
```

The multiplier `mu` is used to model a loss of flow-good in the incomming arcs. The factor also depends on the specific arcs. Another variant of this type for min-cost-flow models will be presented in the examples of Chapter 6, Section 6.1. Let us now conclude with the rule-examples for linearity properties of the two latter formulation types.

The rule following below describes a sufficient condition for a constraint that is represented by a formulation entity of type `NetForms#SingleCommodityBalance` to

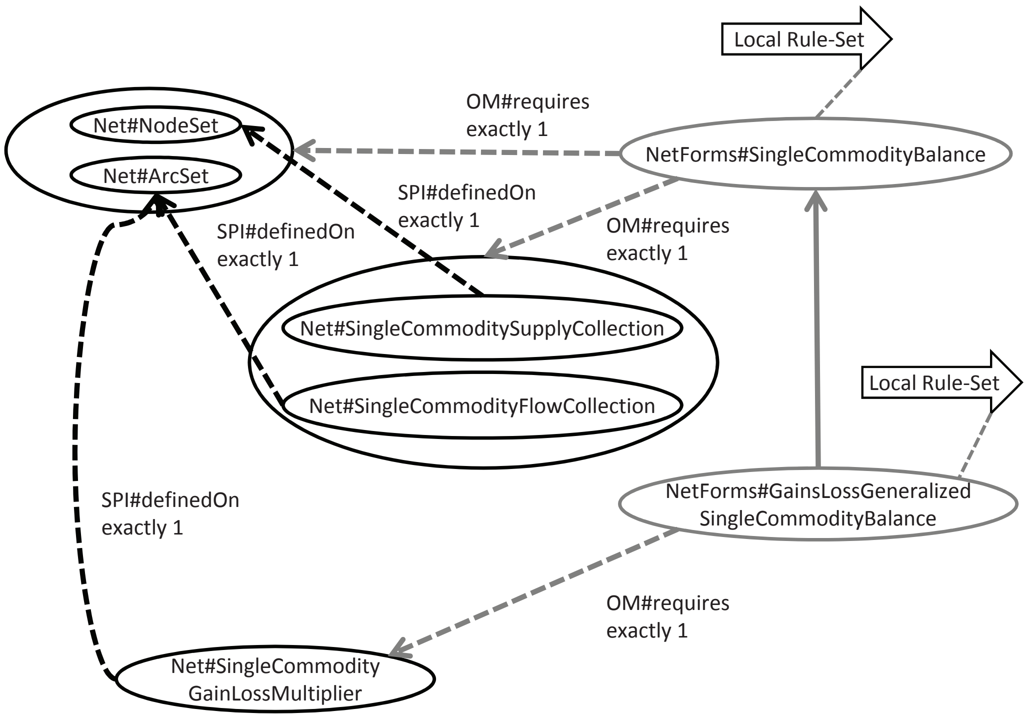

Local Rule-Set
OM#requires exactly 1
Net#NodeSet
Net#ArcSet
NetForms#SingleCommodityBalance
SPI#definedOn exactly 1
SPI#definedOn exactly 1
OM#requires exactly 1
Net#SingleCommoditySupplyCollection
Net#SingleCommodityFlowCollection
Local Rule-Set
SPI#definedOn exactly 1
NetForms#GainsLossGeneralized SingleCommodityBalance
OM#requires exactly 1
Net#SingleCommodity GainLossMultiplier

Figure 5.29.: Two balance formulation types in a subclass relation

be a linear formulation. This single rule does not provide the composition of necessary and sufficient conditions. Furthermore, one has to say that linearity wouldn't be the only property to be considered in a complete rule-set for `NetForms#SingleCommodityBalance`. As already mentioned above, completeness of a rule-set in terms of its capability to decide every mathematical property would be a desireable property, but cannot be guaranteed by the ansatz in general. The rule below is given in a standard FOL-Syntax. Since it represents a Horn-clause, and all predicates refer to respective ontology resources, it can simply be translated into SWRL syntax for implementation purposes:

$$\begin{aligned} &\texttt{NeForms\#SingleCommodityBalance(?x)} \\ &\wedge\ \texttt{OM\#requires(?x,?y)} \\ &\wedge\ \texttt{Net\#SingleCommodityFlowCollection(?y)} \wedge\ \texttt{OM\#Variable(?y)} \\ &\wedge\ \texttt{OM\#requires(?x,?z)} \wedge\ \texttt{Net\#SingleCommoditySupplyCollection(?z)} \\ &\wedge\ \texttt{OM\#Parameter(?z)} \\ &\Rightarrow\ \texttt{OM\#formulationMProp(?x,OM.MProps\#linearFormulationInd)}. \end{aligned} \tag{5.8}$$

The rule states that any individual of `OM#SingleCommodityBalance` for which the single required flow individual is a variable and the single required supply individual is a parameter is itself linear. To that end, the consequent constructs the respective `OM#formulationMProp` axiom for the variable of the antecedent that captures the constraint individual (*safety condition*). This is not the only rule that provides sufficient conditions for linearity. Since the balance constraint reflected by the semantic type does not contain any products, it is also a linear formulation when the supply individual in addition to the flow individual is considered as a variable. In general, for `NetForms#SingleCommodityBalance` there seems to be no possibility to destroy the linearity property. This situation changes when taking a look at the rule-set for the subtype `NetForms#GainLossGeneralizedSingleCommodityBalance` which contains a product term. For this type, the rule above has to be extended by some further parts in the antecedent which will be presented below. Before continuing the considerations for the subtype, the interested reader my also take a look at Figure 5.30. The figure illustrates the assumed situation for a successfull application of aboves rule together with the newly inferred axiom in a bordered box. The specifications in the figure again constitute an excerpt of the ontology specifications for the min-cost-flow model from Subsection 5.3.5.

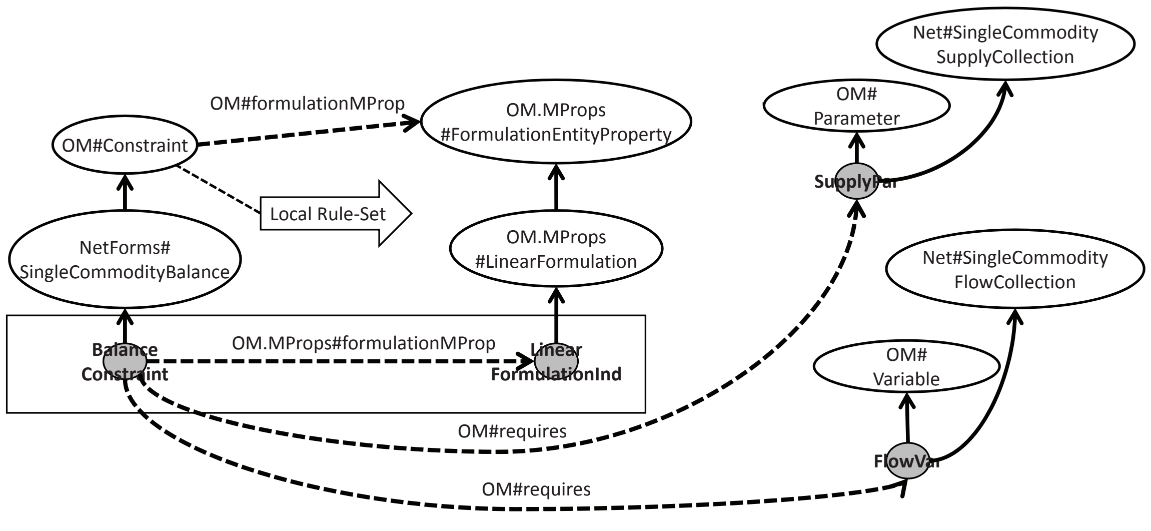

Net#SingleCommodity SupplyCollection
OM#formulationMProp
OM.MProps #FormulationEntityProperty
OM# Parameter
OM#Constraint
SupplyPar
Local Rule-Set
OM.MProps #LinearFormulation
NetForms# SingleCommodityBalance
Net#SingleCommodity FlowCollection
Balance Constraint
OM.MProps#formulationMProp
Linear FormulationInd
OM# Variable
OM#requires
FlowVar
OM#requires

Figure 5.30.: Given and inferred knowledge for the single linearity rule and a balance constraint

The respective rule for the type `NetForms#GainLossGeneralizedSingleCommodityBalance` now reads as follows:

$$\begin{aligned}
&\texttt{NetForms\#GainLossGeneralizedSingleCommodityBalance(?x)}\\
&\wedge\ \texttt{OM\#requires(?x,?y)}\\
&\wedge\ \texttt{Net\#SingleCommodityFlowCollection(?y)} \wedge\ \texttt{OM\#Variable(?y)}\\
&\wedge\ \texttt{OM\#requires(?x,?z)} \wedge\ \texttt{Net\#SingleCommoditySupplyCollection(?z)}\\
&\wedge\ \texttt{OM\#Parameter(?z)}\\
&\wedge\ \texttt{OM\#requires(?x,?gl)} \wedge\ \texttt{Net\#SingleCommodityGainLossMultiplier(?gl)}\\
&\wedge\ \texttt{OM\#Parameter(?gl)}\\
&\Rightarrow\ \texttt{OM\#formulationMProp(?x,OM.MProps\#linearFormulationInd)}.
\end{aligned} \tag{5.9}$$

This rule models a part of the general condition that linearity of the constraint can only be guaranteed when either `mu` or `ArcFlow` is a parameter to the model. In the given case, the rule antecedent forces `ArcFlow` to be a variable as usual and `mu` to be a parameter (as it is also usual for min-cost-flow models). But other rules with interchanged roles may also be formulated and would be required in a complete rule-set (such a set exists for the type above since there only is a finite number of variable/parameter allocations). It is also important to mention that the first rule that was presented above for the type `OM#SingleCommodityBalance` is not valid for the subtype, as the respective product situation for the new parameter `mu` is neglected.

Limitations of the SWRL Rule Ansatz:
As already mentioned, the rule ansatz cannot always provide rule-sets that decide a mathematical property for a given modeling situation. A general problem for rules lies in type specifications that use a cardinality restriction other than `exactly 1`. Examples of types requiring a flexible number of, e.g., expression-entities with a `min 1` cardinality-classifier will be given in Subsection 6.1.3. Nonetheless, Figure 5.31 provides a specification pattern that models the relevant situation. The `ExampleType` requires minimum one individual of `ExampleExpression` which itself requires exactly one `ExampleMDOType`.

A rule that would, e.g., imply linearity of a formulation entity of the given type would have an antecedent that states that every of the `min 1` individuals of `ExampleExpression` is a linear expression (this would also have to be reflected by a suited ontology property). But since the number of individuals is arbitrary, such a rule cannot be defined in SWRL. A first idea for a work-around would be to split the rule into a part that tries to prove that the "implication" of linearity holds for all expression individuals in the model. But this proof cannot be performed as the predefined class `ExampleExpression` is not closed, i.e., not "all" individuals are known and hence it cannot be assured that no "other" nonlinear individual of an

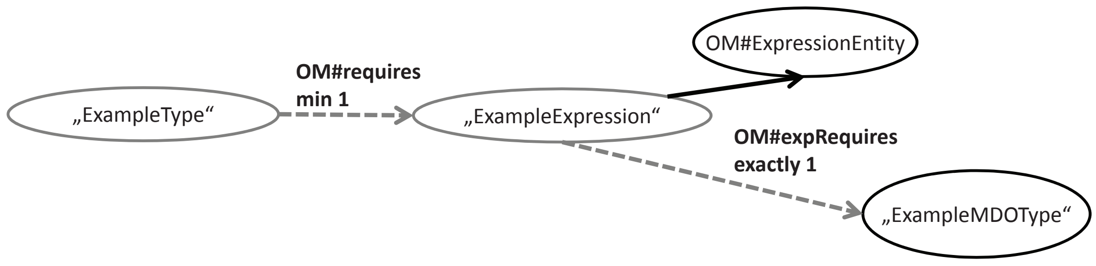


Figure 5.31.: Pattern for an undecideable mathematical property with the SWRL rule ansatz

expression will be added.

Concluding Remarks on Rules:
The latter counter example suggests that SWLR-based reasoning might not be the most accurate way to perform the extension of a model by mathematical properties such as linearity. A procedure that could neglect the open world semantics of OWL could simply check all given subexpressions of a fixed formulation entity in a loop in order to come up with a decision. Nonetheless, SWRL provides a standardized approach to reasoning tasks on OWL ontologies. As such, the author decided to include the SWRL approach in order to cover the requirement **Compatible with (semantic) Web Technologies**. The usage of standards also underlines the portability of the reusable type formulations.

To conclude the latter examples, the general design of rules will be discussed now. This shall cover all the mathematical property aspects that were mentioned in the beginning of this subsection. To repeat a general remark, it is emphasized that all rules are to be designed as Horn-clauses satisfying the *safety conditions*. The design of a complete rule set for a formulation type is a goal for providing necessary and sufficient conditions for every mathematical property of formulations under consideration, though it cannot be achieved in all situations.

- Integer and binary variables: These properties are assumed to be specified a priori, e.g., by a user. Therefore, no rules are required.
- Linearity and general nonlinearity: Specifications of linearity are reflected by the two types `OM.MProps#LinearFormulation` and `OM.MProps#GeneralNonLinearFormulation`. For concluding linearity, examples of rules were given. General nonlinearity has to be considered with more care as a further detailed modeling of nonlinearity aspects such as quadratic formulations that are currently not present in the formalism would change the conditions for this property. Nonlinearity can, e.g., be concluded from the presence of variables in certain situations, e.g., a product of variables

or the usage of a variable as an exponent. But this is not a complete characterization. A formulation entity without a concluded or existing nonlinearity property will be treated as a nonlinear one (see "Model Analysis Step" below).

- Convexity: Rule sets can be extended for checks of convexity by rules that may also use preceeding inferences on linearity. As such, any linear formulation can be considered as convex and a respective rule can be added to any rule set. As a formulation entity can be either convex or non-convex and in contrast to linearity and general nonlinearity there are no possible states in-between (such as a quadratic formulation), all rules can be formulated for proving a convexity property to be described by the class `OM.MProps#ConvexFormulation`. That means, if convexity cannot be concluded, non-convexity is assumed to be the case and no extra rules have to be defined. There is no concept for non-convexity in the `OM.MProps` ontology. As an example, in Section 6.3, a headloss-constraint in a pipe network for water will be presented that is non-convex in the treatment of a standard flow-variable.

- Continuity and differentiability aspects: The three concepts `OM.Props#ContinuousFormulation`, `OM.MProps#C1Formulation` and `OM.MProps#C2Formulation` and their pre-defined individuals reflect the continuity and differentiability properties of a formulation. The most important property in this case may be `OM.MProps#C2Formulation` that reflects twice-continuously differentiable expressions in the domain of variables for a formulation entity. This property is required by some solvers for the problem classes NLP and MINLP and as such should be guaranteed. Respective rules can be designed, though a complete rule-set may again be unrealistic.

The Model Analysis Step:
With suitedly defined mathematical properties for both formulation and data entities, the whole model can be analyzed in terms of relevant properties for mathematical programming. The description of the analysis process is given in Figure 5.32.
The process starts with the assumption of a linear, convex and smooth model that only contains continuous variables. This is reflected by the initializations in lines `1-5`. A loop over every model entity will change the values of the respective boolean variables if either the missing of a mathematical property is detected, or a `OM#dataMProp`-axiom with object `OM.MProps#IntegerData` or its subclass `OM.MProps#BinaryData` is found. Depending on the boolean values, names of predicates that can describe the mathematical properties in a service description of the model are added to a list. This list is returned by the procedure and can be used for the generation of service descriptions (next subsection). As it has already been

Algorithm 5: analyzeModel(OWL model)

```
Data: OWL model
Result: List<String> results of predicate-names
List<String> results = null ;
Bool linearity = true ;
Bool convexity = true ;
Bool ctwo = true ;
Bool requiresintegrality = false ;
for model entity in model do
    if there is no mathematical property axiom with object of type
    OM.MProps#LinearFormulation for model entity in model then
        linearity = false ;
    end
    if there is no mathematical property axiom with object of type
    OM.MProps#ConvexFormulation for model entity in model then
        convexity = false ;
    end
    if there is no mathematical property axiom with object of type
    OM.MProps#C2Formulation for model entity in model then
        ctwo = false ;
    end
    if there is a mathematical property axiom with object of type
    OM.MProps#IntegerData for model entity in model then
        requiresintegrality = true ;
    end
end
if linearity then
    results.add("isLinear") else results.add("isNonlinear") ;
end
if convexity then
    results.add("isConvex") else results.add("isNonConvex") ;
end
if ctwo then
    results.add("isC2Smooth") else
    results.add("isNonDifferentiable") ;
end
if requiresintegrality then
    results.add("requiresIntegrality") else
    results.add("continuousVariableModel") ;
end
return results ;
```

Figure 5.32.: Analyzing mathematical properties of a model

mentioned before, missing mathematical property specifications, e.g., for linearity, lead to worse properties than reality, i.e., nonlinearity in this example. For the convexity property the fact that an intersection of convex feasible regions is again convex is exploited. I.e., if all formulations describe convex sets, the feasible region described by the whole model will be convex (in this case a relaxation dropping integrality requirements is considered).

5.6.2. Generating Model Instantiation and Solver Service Descriptions

The template-based semantic service composition of an optimization system as it was outlined in Section 5.1 searches and binds software services for solvers and further computing steps such as preprocessing and simulation. The generation of semantic service descriptions, e.g., in a simple scenario for model instantiation service and solver is a required step before the composition of the system. This subsection shall explain how such descriptions can be generated in an automated fashion at the hand of this simplified scenario. The investigation and realization of the composition approach for the whole system is not part of this thesis.

The description of the semantic services, in accordance to the remarks on "Service specifications in this thesis" from Subsection 2.6.1, requires signatures, pre- and postconditions, where the aspects considered for the service desciptions of models and solvers should cover data structure and semantics, as well as the mathematical properties of a model, at least. The latter properties should especially allow to match the capabilities of a solver which is to be inserted in the composition step. Nonfunctional properties of solvers such as running times and costs are not part of the considerations here. Figure 5.33 provides the respective simple semantic service descriptions for the running min-cost-flow model example from Section 5.2.
The figure contains the single methods `Instantiate(...)` and `Solve(...)` of the two services. The method signatures declare some inputs and outputs with respective formats:

- `Instantiate(...)`: The input `instdata` of type `NFlowData` provides the input data for the model. Remarks on this data are given in the text below this listing. For the output `osimod` that represents and optimization instance, the flexible XML-format OSiL from the Optimization Services (OS) project [FMM10b] is used as a type. This format should fit many solvers.
- `Solve(...)`: The input `optInstance` of format `InstMOD` represents the input to the solver. A respective assignment `optinstance:=osimod` is made in the data-flow. The semantic type `InstMod` represents a general superclass of all possible solver input formats. To that end, no solver should be rejected in a composition due to format inconsistencies. Since the type `OSiL` was fixed for the generated model instantiation service, an adaption of the output format for

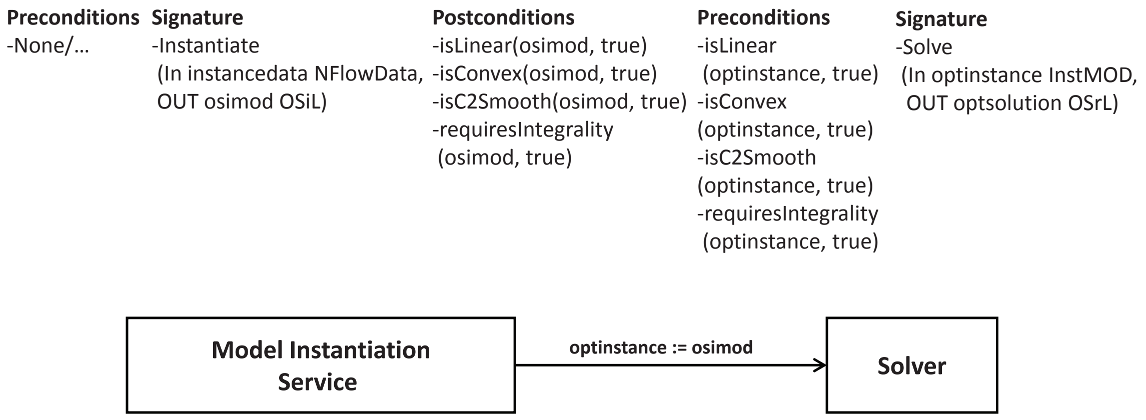


Figure 5.33.: Simple service descriptions for model and solver in case of the min-cost-flow-model

> that service should be made in case a composed solver is not capable of OSiL. For further possible output formats, the postconditions of the model instantiation service could be extended with other formats. As this may complicate the composition procedure, such as postcondition has not been added to the example. The output `optsolution` of the solver then has the semantic type `OSrL` representing the respective generous solver result format from the OS-project.

The signatures and types can be considered as similar for every model and hence the respective parts can be generated automatically, where the following remark has to be made on the semantic data type of the model instantiation service's input:
The input `InstanceData` of the method `Instantiate(...)` in the example is of the semantic type `NFlowData` that represents data in a suited format for the underlying min-cost-flow model. But this semantic type is just a placeholder that can be generated automatically for every model. The placeholder is required for the description of a data transformation that transforms data of preprocessing services into the format of the model. This data transformation should be generated semi-automatically after the template-composition step that in our case binds concrete solver and preprocessing services. If no transfomation is required, a suited template-path might have been chosen a priori or the transformation works as an identity mapping. Note that the matching of instance data and the (flexible and changing) model is not self-evident. In general, the composition of a model instantiation service with a service providing the instance data, e.g, a data-base service or as envisioned in this thesis, some computational service performing a preprocessing, will not work since there are differences in data. In cases of network

models, a preprocessor or data-base may store/process data in a network-typical format, whilst the model instantiation service requires sets and parameters in respective lists and tables that are specific to the model.

The considerations of an automated service description generation can be concluded with the mathematical properties of a model. Predicate names from a standardized vocabulary have been generated by the Model Analysis Step 5.32 in combination with a preceding extension of mathematical property specifications 5.28 as it was presented in the last subsection. The task now is to suitedly put them into the service descriptions for model instantiation service and solver in terms of pre- and postconditions:

- Linearity: The predicate `isLinear(osimod,true)` is generated for the fixed identifier `osimod` of the service description. `isLinear` was returned by the Model Analysis Step for stating the linearity of the min-cost-flow model. The same predicate is also used for the respective input identifier `optinstance` of the solver service to specify an *allowed* precondition. I.e., solvers that have the linearity requirement of instances as a precondition will not be rejected by a composition procedure. Solvers that do not have the linearity requirement and hence can treat nonlinear models are allowed anyway. Due to the assignment of `osimod` to `optinstance`, the preconditions of any matching solver service will always be fulfilled by the results of the model instantiation service. This allows to consider the service composition task as a form of template composition, where matching services have to be found for every placeholder that is specified by a semantic service description. In the considered case, the model instantiation service is fixed in a first step and does not need to be searched for in a composition.

- Convexity: In a fashion similar to the one mentioned above, the results of the Model Analysis Step lead to the post- and preconditions `isConvex(osimod,true)` and `isConvex(optinstance,true)` with the same predicate `isConvex`.

- Continuity: Similar specifications are retrieved for the case of continuity with the post- and preconditions `isC2Smooth(osimod,true)` and `isC2Smooth(optinstance,true)`.

- Integrality Requirements: Finally, `requiresIntegrality(osimod,true)` and `requiresIntegrality(optinstance,true)` can be generated.

To conclude the considerations of this subsection, the following should be mentioned: It was demonstrated how simple semantic service specifications for model instantiation and solver services can be generated out of the model specifications. The specifications integrate with the system generation and service composition approach 5.2 from Section 5.1. The choice of a template path which is a part of the template configuration runs parallel to the generation of the model instantiation service. The latter step implicitely includes the generation of service descriptions which is not exploited to the full amount in this thesis. Besides the model instantiation service that is nonetheless not due to a composition, it was demonstrated how such descriptions can be generated for the core component, the solver service. The generation of descriptions for further services such as a preprocessing or simulation is left open. In the end, after a successful template composition, all service placeholders besides the data transformations should be bound to concrete services. Due to the correspondence of model instantiation's post- and solver's preconditions, the solver's algorithmic capabilities should match the problem class of the model's instance. Right before the execution of the system, data transformations should be generated semi-automatically where the generation of these is furthermore due to the ontological specifications. This subsection only introduced a generic type for the model's instance data. In an automated data transformation generation approach as described in Subsection 2.6.3, the mapping of structural data to ontological types as well as the mappings in-between ontologies need to be exploited. This is facilitated by the representation approach of the ontology represented abstract optimization models via the introduction of meta domain ontologies and concrete domain ontologies.

5.7. Conclusion

This section concludes the content of this chapter and discusses the coverage of requirements by the features of the approach that were presented so far. In this chapter, the basic vocabularies and representation structure of ontology represented abstract optimization models were introduced. After discussing the standard vocabularies `OM, SPI` and `OM.MProps`, a complete specification example for a min-cost-flow problem was given. In the remainder of the chapter, according to the view from Section 5.1, methods for having the ontology-representation as a layer in-between AMLs and semantic model instantiation services were presented. Expression Structure Separation and AML Derivation were presented in detail together with two examples for a data and a formulation entity. Processes for both deriving all statements of the individuals of a single model entity type, as well as all statements of an ontology model, were presented. An AML Derivation Service served as a basis for the consideration of a model instantiation service in Section 5.6. Concluding the remarks on Expression Structure Separation and AML Derivation, the discussion in Subsection 5.4.5 highlighted that model entity types and expecially formulation entity types are flexible in their indexing structure and the semantic types of the data model conceptualizations. Thereby, flexibility of the expression structures for summations was described as a feature to be demonstrated in the next chapter. The next chapter will also clarify that flexible types are not the ultimate goal to achieve, but that rather many slightly different entity types should be organized in the ontologies in an intelligent way.

For the AML Reengineering, only some initial considerations were presented. As it is possible to formulate and manipulate any model in the ontology formulation with the aid of standard tools for ontology editing, this topic has been marked as somewhat out of the scope of this thesis. Nonetheless, AML Reengineering should be an important technique for making ontology represented models practical. The formulation and editing of models in the ontology representation will also be a topic of the next chapter.

The last section on model instantiation services introduced methods to generate a semantic service and especially semantic service descriptions for the automated generation of an optimization system by means of semantic service composition. The role of a possibly changing abstract optimization model was highlighted and processes to generate the service descriptions in an automated fashion were presented. Though a combination of an automated specification extension of a model, together with an automated model analysis, can always be used to generate semantic service descriptions for models and solvers, the generated descriptions might represent the properties of the model "worse" than they are (, e.g., claim a model to be nonlinear and nonconvex instead of being linear and convex). But as a user may also perform specifications by hand that in the case of changes to the model might not change or where the changes will be detected automatically, the perspective that the presented methods work out well should be emphasized.

Let us now shortly discuss the coverage of the requirements from Section 4.2 by the techniques presented in this chapter. Open issues will be mentioned and solutions will be introduced in the next chapter:

- **Model Formulation by abstract Types**: A representation for abstract optimization models that is centered around reusable model entity types has been introduced in this chapter. The view of modeling components is reflected by the top-level definitions of the vocabulary `OM`. By this, model formulation in the proposed representation is a process of instantiating and relating reusable types. Examples for reusable types were formulated and basic ontologies have been defined. Nonetheless, the core discussion of the model formulation process and capabilities to reuse formulation types is left open for the next chapter. This will also cover the aspect that manipulations in the model should be simple and automatable.

- **Model Semantics**: All required types of semantics can be expressed by the representation approach as it should have become clear from the definitions of the basic vocabularies `OM, OM.MProps` and `SPI` as well as the specifications for the min-cost-flow example.

- **Automated Modifications and Recomposition**: This requirement has not been considered in its full extent so far. Especially, the automated modification capabilites for ontology represented models have been neglected and will be a topic of the next chapter. Nonetheless, when models get manipulated and changed, the analysis of models and the generation of services can be automated such that a new composition of an optimization system can take place. The latter aspects were part of the last section of this chapter.

- **Models as Services and Composition**: This requirement should by now be covered with the aid of the considerations of Section 5.6. Especially the matching of model and solver was discussed there.

- **System Integration**: The framework presented in the beginning of this chapter included different types of services. Especially, the model instantiation services and the data transformations to be generated after the composition procedure were introduced. The matching of model and solver, as also discussed in the last requirement, can be seen as a part of the system integration. The matching of data interfaces and generation of data transformations is included in the framework by the means for ontology and service specifications as well as the respective consideration of transformation services in the system generation process. Nonetheless, the practicability needs to be further evaluated in the future and goes beyond the scope of this thesis.

- **Service Enabling**: Methods for deriving AML models out of the ontology representation were a topic of Section 5.4 and have been demonstrated in detail. Further means will also be provided by the demonstrator presented in Chapter 8. The automated generation of semantic service descriptions together with the AML Derivation capabilities and the associated possibility to use AML tools for instantiation with data then complete the basic coverage of this requirement. Nonetheless, AML Reengineering has only been presented in terms of some basic ideas and should be investigated further in the future research.

- **Compatible with (semantic) Web Technologies**: This requirement is covered as the design is based on current standards such as OWL, XML and XSD, as well as emerging technologies such as SPARQL-DL or SWRL.

Why Ontologies?
At this point, a remark should be made why ontologies were chosen as the core technology for the framework and the approach of this thesis. Object-oriented technologies in cooperation with the Structured Modeling (SM) framework presented in Chapter 3 provide established technologies for the management of decision models as well as the development of related decision support systems. Though the explicit

formalization and design of reusable formulation types has been identified as a research gap in Chapter 3, one could think of realizing the latter under OO/SM. But here, the means of ontologies to provide explicitely specified semantics in a portable fashion that can be distributed over the web represents a novelty. By the aid of the latter semantics, type formalizations are made explicit and can be analyzed and queried by different agents on the web. Reasoning, as it is enforced by ontology technology, is a further relevant feature. The representation in ontologies in the latter regard interacts well with semantic services. Semantic service approaches might then provide a framework to overcome drawbacks of component-based approaches. Besides the novel features, old benefits of Structured Modeling, such as the extensible graph form and the means for model integration, are also available in the ontology approach. To that end, the usage of ontologies for a framework as presented within this thesis is highly motivated. Nonetheless, in order to be practical, OO-technology might also be exploited and integrated in the future. The implementations of AML derivations for multiple types should, e.g., be designed well by the aid of object-oriented design patterns and they should be provided as a component for suited reuse in different contexts. Furthermore, tools for model formulation should benefit from a proper object-oriented design and supporting technology.

6. Reusability and Model Formulation in the Ontology Representation

The last chapter introduced the ontology representation for abstract optimization models and discussed its role in-between AMLs and services. The top level ontologies for the representation approach were introduced and their usage for the specification of abstract optimization models was demonstrated for a simple min-cost-flow example. This chapter will now deepen the consideration of optimization model formulation in the language framework by discussing the ontology structure of reusable formulation types and their usage in the model formulation. This will mainly be done by the aid of exemplary modeling situations which will typically be embedded into the context of case study models. The modeling situations will lead to the encapsulation of reusable blocks of AML statements within respective formulation types. The examples will cover different conceptualizations of network balance constraints, generous network resource availability constraints, linearizations and smoothings of nonlinear constraints with the opportunity to switch in-between them, integral and continuous nonlinear formulations for a same modeling situation, again with the opportunity to choose between them, as well as formulation type variants for providing the modeling technique of soft constraints. As most of of the examples can be seen as parts of either the domain of network flow problems or the domain of water distribution systems planning, two sections that are each dedicated to one of the latter two domains make up the main part of this chapter. Different case study models from the latter domains will furthermore provide the required modeling context for many of the reusable type formalizations.

As this chapter is structured according to exemplary formalizations, it is important to mention that the language framework introduced within this thesis is capable of formalizing any mathematical programming model up to class MINLP. Anyhow, the formulation of a group of constraints in a model requires the existence of suited reusable type(s), as the ontology representation approach is not designed to directly represent AML statements, but rather treats them in form of a type associated derivation functionality. Recapitulate, e.g., Section 5.4 for the latter issue.

6.1. Considerations for Min-Cost-Flow Models

This section studies those type formalizations and model formulation situations that can be associated to the domain of network flow problems. For this purpose, two extensions of the min-cost-flow model from Chapter 5 will be introduced as case study models in the remainder of this section. The formalizations to be introduced will be suited to discuss the inheritance of types and the possibilities to define flexible formulation types in terms of the innner expression building blocks. Further features will be discussed at the example of a water-network case study in a later section of this chapter.

6.1.1. The Single Commodity Balance Hierarchies

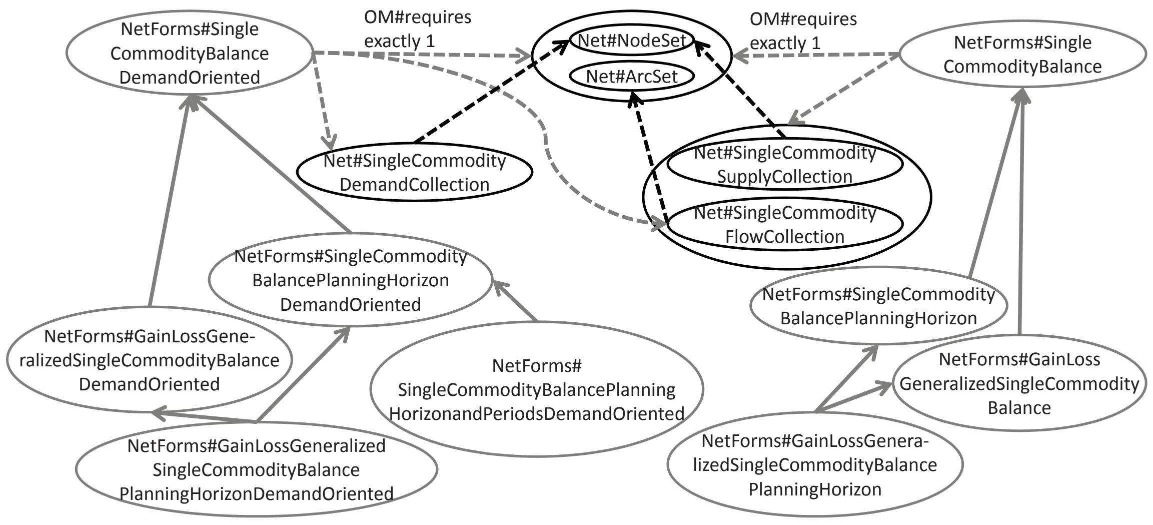

NetForms#SingleCommodityBalanceDemandOriented
OM#requires exactly 1
Net#NodeSet
Net#ArcSet
OM#requires exactly 1
NetForms#SingleCommodityBalance
Net#SingleCommodityDemandCollection
Net#SingleCommoditySupplyCollection
Net#SingleCommodityFlowCollection
NetForms#SingleCommodityBalancePlanningHorizonDemandOriented
NetForms#SingleCommodityBalancePlanningHorizon
NetForms#GainLossGeneralizedSingleCommodityBalanceDemandOriented
NetForms#SingleCommodityBalancePlanningHorizonandPeriodsDemandOriented
NetForms#GainLossGeneralizedSingleCommodityBalance
NetForms#GainLossGeneralizedSingleCommodityBalancePlanningHorizonDemandOriented
NetForms#GainLossGeneralizedSingleCommodityBalancePlanningHorizon

Figure 6.1.: Taxonomy of single commodity balance types

Network balance or flow conservation equations are common constraints in network-based optimization models. Depending on the underlying model and domain, e.g., standard network flow problems or practical application networks such as water distribution systems, the equations can occur in different forms. This subsection presents network balance constraint formulation types in the ontology optimization model representation. Different types will be presented and brought into a taxonomy such that choices and required adaptions of specifications in a model formulation process are made explicit. Inheritance and multi-inheritance are central concepts, where a hierarchy of subtypes can be developed towards formulations that are more and more general, e.g., by including more parameters, variables and indices. Nonetheless, the general types will also need a general model context in order to be applicable.

The taxonomy is visualized in Figure 6.1. Due to place limitations, the image does not contain all required mdo-classes but just the ones for the top-level types. The type requirements of the subtypes will be discussed in a passage below and are visualized for a certain branch of types in Figure 6.2. Taking a look at Figure 6.1 one can identify nine formulation types in grey ellipsoids that are partially organized in subclass hierarchies by the usage of (multi-) inheritance. Two isolated types `NetForms#SingleCommodityBalanceDemandOriented` and the already known type `NetForms#SingleCommodityBalance` can be observed in the upper part of the figure. The split of types for *demand-oriented* and *normal-oriented* ones is explained by different conventions on the prefix of nodal flow good requirements. This is reflected by a distinction of supply and demand parameters in network models which are semantically represented by the types`Net#SingleCommoditySupplyCollection` and `Net#SingleCommodityDemandCollection` in the figure. Whilst network flow models typically introduce a supply parameter with the convention of lower zero values for a consumption of flow good, network models for, e.g., water networks use positively defined demands for modeling a required measure of flow good to be consumed in a node. To that end, the orientation in terms of indexing or prefixes in balance constraint defining statements differs in the latter models. To give an example, e.g., a target AMPL statement for `NetForms#SingleCommodityBalance`, as already given in Section 5.3, reads

```
                                       s.t.  MassBalance { i in Nodes} :
sum{(i,j) in Arcs}(ArcFlow[i,j]) - sum{(j,i) in Arcs}(ArcFlow[j,i])
                                                   = SupplyDemand[i];.
```
(6.1)

In contrast to that, a similar balance statement in a water network model, as it will be presented in Section 6.3, reads

```
                  s.t.  Massbalance { x in JunctionNodes} :
sum{(i,x) in Pipes}(Q[i,x]) - sum{(x,j) in Pipes}(Q[x,j])
                                           = Demand[x];.
```
(6.2)

One can see that with the change of the fixed constraint indice `i` to `x`, the position of the outgoing flow good summations (`(i,j)` and `(x,j)`) is changed from being positively counted to being subtracted. Complete models containing the latter statements will be presented in the following paragrahs of this chapter. By now it is important to mention that the differing orientation of statements in combination with the differing usage of demand and supply concepts leads to different mdo-type requirements in the formulation types. This is reflected in Figure 6.1 where the visualized type requirements of both formulation types differ in the requirement of `Net#SingleCommodityDemandCollection` and `Net#SingleCommoditySupplyCollection`. As the two mdo-types represent distinct classes, the respective formulation types also make up different root nodes of subsumption hierarchies. This is especially due to the inheritance of class restrictions in ontology class subsumption hierarchies.

The hierarchies for `NetForms#SingleCommodityBalanceDemandOriented` and `NetForms#SingleCommodityBalance` are quite similar. In both cases, two direct subtypes are specified that extend the given types by means for "`GainLoss`" or a "`PlanningHorizon`". Furthermore, the two subtypes themselves have a common subtype that inherits the type requirements of both types by means of multi-inheritance. This type then provides means for a "`GainLossGeneralizedSingleCommodityBalance`" with "`PlanningHorizon`". An exemplary statement for the type `NetForms#GainLossGeneralizedSingleCommodityBalance` would, e.g., be

```
            s.t.  MassBalance { i in Nodes} :
                  sum{(i,j) in Arcs}(ArcFlow[i,j])
    - sum{(j,i) in Arcs}( mu[j,i] * ArcFlow[j,i])
                                  = SupplyDemand[i];,
```
(6.3)

with a multiplicative parameter `mu` that models the gain or loss of flow good in incoming arcs. The type will be further explained and brought into the context of a model in Subsection 6.1.4. The type `NetForms#SingleCommodityBalancePlanningHorizon` adds a time index to the balance constraint. An exemplary statement would be

```
s.t.  MassBalance { i in Nodes, t in 0..T} :
         sum{(i,j) in Arcs}(ArcFlow[i,j,t])
      - sum{(j,i) in Arcs}( ArcFlow[j,i,t])
                          = SupplyDemand[i,t];,
```
(6.4)

with a planning horizon `0..T` that is build by the aid of a positive, integral parameter `T`. Respective additional indices `t` occur in the arc-flows, supply-demand parameter and the constraint group itself. Finally, the formulation type `NetForms#GainLossGeneralizedSingleCommodityBalancePlanningHorizon` which inherits the cardinality class restrictions of both the latter types combines the features. An example statement could, e.g., read

```
        s.t.  MassBalance { i in Nodes, t in 0..T} :
                  sum{(i,j) in Arcs}(ArcFlow[i,j,t])
    - sum{(j,i) in Arcs}( mu[j,i] * ArcFlow[j,i,t])
                                 = SupplyDemand[i,t];,
```
(6.5)

where the gain/loss multiplier is constant over time.

For now, let us continue the investigation of the type hierarchy by observing the type `NetForms#SingleCommodityBalancePlanningHorizonandPeriodsDemandOriented` which is specified as a subtype of `NetForms#SingleCommodityBalancePlanningHorizonDemandOriented` and extends the latter by the consideration of an additional parameter for periods with respective indices. An exemplary statement for this demand oriented balance type with two time indices will be given in the context of a water network renewal planning model in Section 7.3. For the moment it suffices to observe that subsumption can be iterated such that lower level types add additional `OM#requires` restrictions for certain mdo-types. These restrictions will be explained the next. Before doing so, let us conclude the latter observations as follows: Two subsumption hierarchies with different subtrees and also joins of subtrees due to multi-inheritance were defined. Subtypes on lower levels represent formulation types that are capable of representing more and more features. Besides the formulations shown in the figure, further types could be defined for future applications. The next, the subsumption hierarchy for demand-oriented types shall be further discussed in terms of mdo-requirements.

The type hierarchy for the demand-oriented types together with the mdo-requirements is visualized in Figure 6.2. As one can see there, formulation subtypes typically extend the type requirements of their ancestor(s) by additional type requirements from respective meta domain ontologies. The type `NetForms#GainLossGeneralizedSingleCommodityBalanceDemandOriented` inherits all requirements fom `NetForms#SingleCommodityBalanceDemandOriented` and adds an extra requirement for a `Net#SingleCommodityGainLossMultiplier` which provides the respective parameter to be integrated into the constraint formulations. We conclude with the type `NetForms#SingleCommodityBalancePlanningHorizonDemandOriented` which also is a subtype of `NetForms#SingleCommodityBalanceDemandOriented` and extends its father by requiring a `TimeHorizons#PlanningHorizonLength`. `TimeHorizons` is a further meta domain ontology that provides conceptualizations of sets and parameters for time intervalls in optimization models. The length of a planning horizon in this case is given as a simple nonnegative, integer value that can be used in models to reference a number of time steps until this parameter.

`NetForms#SingleCommodityBalancePlanningHorizonDemandOriented` furthermore specializes the requirement of a `Net#SingleCommodityDemandCollection` to the subtype `Net#SingleCommodityDemandCollectionPlanningHorizon` which extends his father by a further indexing over the latter time steps. A similar type requirement refinement for network flow types is only indicated in the figure due to placing limitations. On the left side of the subsumption hierarchy, the formulation type `NetForms#GainLossGeneralizedSingleCommodityBalancePlanningHorizon-DemandOriented` inherits all type requirements and hence "features" from his two parents without adding a new one of his own. The usage of multi-inheritance hence makes the subtype definitions in this example very simple from an ontology specification perspective. The last type to consider, `NetForms#SingleCommodityBalancePlanningHorizonandPeriodsDemandOriented` extends his father by the additional requirement of exactly one `TimeHorizons#PeriodNumber` and refines the requirement of the demand parameter that is already indexed for the time steps of the planning horizon to one with an additional index for periods. Again, the refinement of the flow type requirement is only indicated in the figure by an arrow.

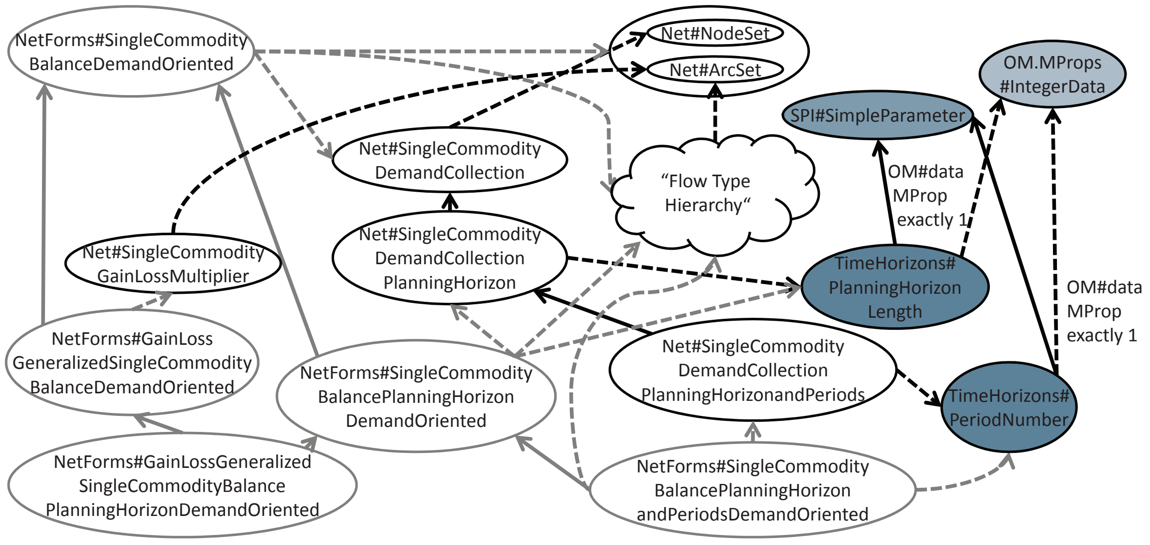

NetForms#SingleCommodity BalanceDemandOriented
Net#NodeSet
Net#ArcSet
OM.MProps #IntegerData
SPI#SimpleParameter
Net#SingleCommodity DemandCollection
"Flow Type Hierarchy"
OM#data MProp exactly 1
Net#SingleCommodity GainLossMultiplier
Net#SingleCommodity DemandCollection PlanningHorizon
TimeHorizons# PlanningHorizon Length
OM#data MProp exactly 1
NetForms#GainLoss GeneralizedSingleCommodity BalanceDemandOriented
NetForms#SingleCommodity BalancePlanningHorizon DemandOriented
Net#SingleCommodity DemandCollection PlanningHorizonandPeriods
TimeHorizons# PeriodNumber
NetForms#GainLossGeneralized SingleCommodityBalance PlanningHorizonDemandOriented
NetForms#SingleCommodity BalancePlanningHorizon andPeriodsDemandOriented

Figure 6.2.: `OM#requires` specifications for the demand-oriented balance types

The subsumption hierarchies of formulation types with explicitely formalized mdo-requirements provide a formalization of knowledge in the model formulation process that lays a fundament for different automatizations. If modeling decisions for adding a model entity of a certain type or changing a type are performed by a human modeler, then required adaptions of the model formulation could be automated under certain assumptions. This could, e.g., mean that an mdo-individual for a new formulation type requirement (in form of a class restriction) is created, typed and added in an automated fashion. In this case, axioms for relating other mdo-concepts to the new one could also need to be adapted or added in a recursive fashion. The possibilities to choose a formulation type in a certain model context are furthermore made explicit by the formulation types and their mdo-requirements.

The latter capabilities are also part of the use cases *Formalized choices in modeling* and *Recommended adaptions in model formulation* from Section 4.1. According to that, the following subsections and sections will discuss examples of situations in which new formulation entities could be recommended based upon the existing model's state (second use case, commodity set example), or the change to a subclass formulation type could be automated (first use case, waternet pipe headloss constraints). Concerning the latter, tools for model formulation with integrated capabilites to build a system automatically could also try different constraint formulations from a subsumption hierarchy in a loop/ in parallel. With the provision of means for automatization, a possible exponential growth of types, e.g., in a subsumption hierarchy should not be a problem. The representation of the knowledge in types is required to enforce automation capabilites and human modelers might only be concerned with the "big" ontologies when defining new types.

6.1.2. Considerations for a Multicommodity Flow Model

This subsection studies further ontology definitions of network flow types as well as operations in the model formulation. To that purpose, a multicommodity min-cost-flow model, as it can be found in the literature, see, e.g., [AMO93], is introduced. The multicommodity definitions for this example model reflect the ones outlined in the second motivational use case "*Recommended adaptions in model formulation*" from Section 4.1.

Figure 6.3 provides an AMPL version of the model. In comparison to the first min-cost-flow model from Figure 5.3, a set `Commodities` is contained in the model that allows to treat more than one flow good. The multiple flow goods are to be transported through the network in an admissible and cost minimal way. The consideration of multiple commodities makes it necessary that certain parameters and variables are defined with an additional index. This concerns the parameters `Cost, SupplyDemand` and the variable collection `ArcFlow`. The upper capacities ("`UpperCapacity`") do not need to be adapted directly as they will be required in the usual, only arc specific, way by more general bounding constraint ("bundle-constraints") to be explained below. Nonetheless, additional commodity dependent bounding parameters ("`param CommodityUpperCapacity`") need to be defined for later usage in respective commodity-individual bounding constraints.

Concerning the formulation of the model in terms of goals and constraints, an additional commodity indexing inside the objective function can be observed that allows to build a sum over all flows of all commodities. The mass-balance equations

```
# Formulation follows Ahuja et al., Network Flows, p. 649 ff.

set Nodes;
set Arcs within {Nodes, Nodes};
set Commodities;
#Commodity set is new compared to the first min-cost-flow model.
#Parameters will be commodity dependent

param Cost{Arcs, Commodities};
param UpperCapacity{Arcs};
param CommodityUpperCapacity{Arcs, Commodities}; #New Individual Commodity
    Capacities!
param SupplyDemand{Nodes, Commodities} integer;

var CommodityArcFlow{Arcs, Commodities} >= 0, integer ;

minimize MultiCommodityFlowCost: sum{k in Commodities}(sum{(i,j) in Arcs}
(Cost[i,j,k] * CommodityArcFlow[i,j,k]));
#Summation also over each commodity

s.t. CommodityMassBalances{k in Commodities, i in Nodes}: sum{(i,j) in Arcs}(
    CommodityArcFlow[i,j,k]) - sum{(j,i) in Arcs}(CommodityArcFlow[j,i,k]) =
    SupplyDemand[i,k];
#Additional indexing for commodities

s.t. BundleConstraints{(i,j) in Arcs}: sum{k in Commodities}(CommodityArcFlow[i,j,k
    ]) <= UpperCapacity[i,j];
#Arc flow bounds extended to a sum over all commodities

s.t. IndividualCommodityCapacityConstraintsUp{k in Commodities, (i,j) in Arcs}:
    CommodityArcFlow[i,j,k] <= CommodityUpperCapacity[i,j,k];
#New bounding constraints that are individual for every commodity
```

Figure 6.3.: A multicommodity min-cost-flow model

carry an additional index for commodities such that a constraint is generated for every node and every commodity. The statement structure is also adapted to respective parameters and variables that carry additional indices for commodities. The upper capabity limitations of network arcs are treated by two different groups of constraints. The generalization of the `UpperCapacityConstraints` from Figure 5.3 to the so-called `BundleConstraints` will be of further interest in the remainder of this section. The constraints use an only arc dependent upper capacity value to express that the sum of all commodity flows (for all commodities together) is bounded from above. Later extensions in another generalized min-cost-flow model will treat the consumption of capacity by different commodities with respective weights. As the bundle constraints do not bound the flows for every individual commodity, additional `IndividualCommodityCapacityConstraintsUp` might be defined which are indexed over commodities and arcs and make use of the new `CommodityUpperCapacity` parameter.

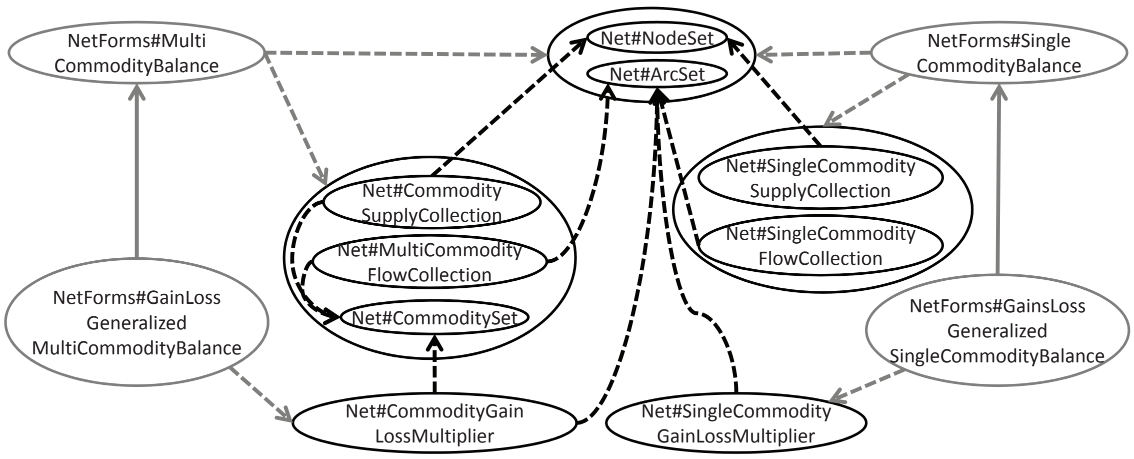

NetForms#Multi CommodityBalance
Net#NodeSet
Net#ArcSet
NetForms#Single CommodityBalance
Net#Commodity SupplyCollection
Net#MultiCommodity FlowCollection
Net#CommoditySet
Net#SingleCommodity SupplyCollection
Net#SingleCommodity FlowCollection
NetForms#GainLoss Generalized MultiCommodityBalance
NetForms#GainsLoss Generalized SingleCommodityBalance
Net#CommodityGain LossMultiplier
Net#SingleCommodity GainLossMultiplier

Figure 6.4.: Balance types with and without commodity sets

Before the ontology specifications on the assertional and terminological level can be discussed, the ontology types for the multicommodity balances should be introduced. As one can see in Figure 6.4, the hierarchy of balance constrain types is furthermore splitted into types for single commodities and multiple commodities. Subtype hierarchies as in the last subsection can be defined again, where the gainloss-generalized types are specified in the figure as examples. An exemplary usage of the type `NetForms#GainLossGeneralizedMultiCommodityBalance` together with exemplary statements will be given in the further extended min-cost-flow example of Subsection 6.1.4. The type in use by the balance constraints from the multicommodity model in this section is `NetForms#MultiCommodityBalance`. This type and his children differ from the single commodity types by requiring a `Net#CommoditySet` as well as that the mdo type requirements of flow and supply are changed to the types `Net#CommodityFlowCollection` and `Net#CommoditySupplyCollection` which are defined over the commodity set. As these requirements are modeled by the cardinality restrictions of the form `NetForms#MultiCommodityBalance requires exactly 1 Net#CommoditySet`, a usage of the types in a model context without commodity sets is not possible. On the other and, extending single commodity types by commodity set specfications is not supported by the statement derivations, such that respective commodity indices would not be generated. Alltogether, the principle to prefer sharply defined types over flexible ones is applied by the ontological type modeling. This is intended to support the automation capabilites in model formulation. The ontology design principles of model entity types will be a topic of Section 6.2.

The ontology specifications for an abstract optimization model are given by the instance knowledge on the assertional level in combination with the allocation of semantic types. When treating the model formulation of the multicommodity min-cost-flow model as a stepwise extension of the first min-cost-flow model (specifications were given in Subsection 5.3.5), adaptions have to be performed both on the assertional level and for the types. The adaptions on the assertional level are visualized in Figure 6.5.

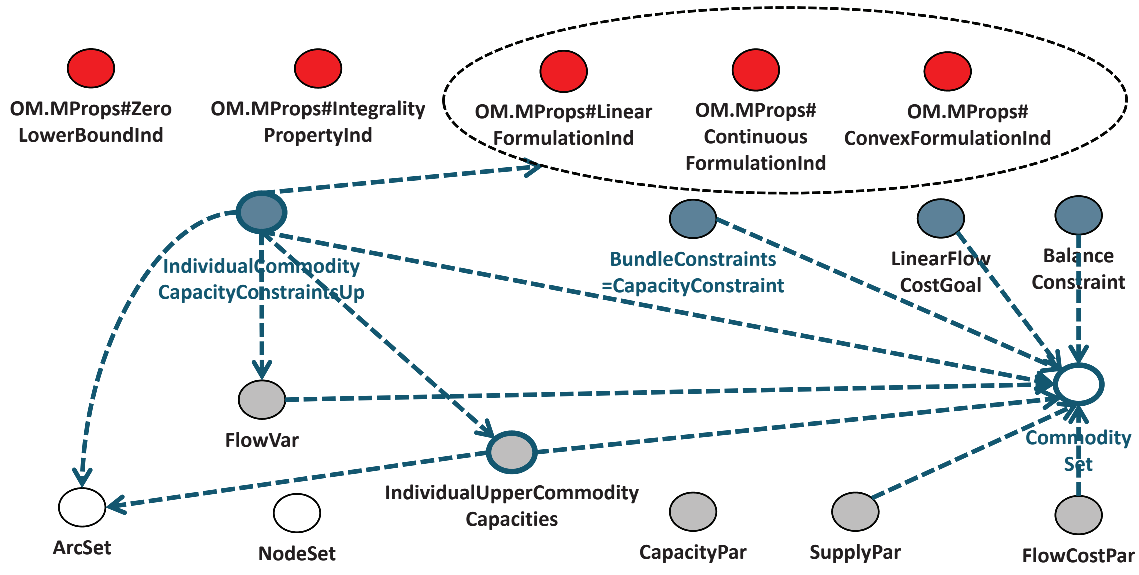

OM.MProps#Zero LowerBoundInd
OM.MProps#Integrality PropertyInd
OM.MProps#Linear FormulationInd
OM.MProps# Continuous FormulationInd
OM.MProps# ConvexFormulationInd
IndividualCommodity CapacityConstraintsUp
BundleConstraints =CapacityConstraint
LinearFlow CostGoal
Balance Constraint
FlowVar
IndividualUpperCommodity Capacities
Commodity Set
ArcSet
NodeSet
CapacityPar
SupplyPar
FlowCostPar

Figure 6.5.: Changes in the assertional specification when extending the first min-cost-flow model towards the multicommodity flow model

The first step towards a generalization of the min-cost-flow model is the creation of an individual `CommoditySet` of the class `Net#CommoditySet`. By this initial step, a conceptualization that allows to adjust the other entities of the model towards a multicommodity formulation is provided. The respective adaptions for the other model entities are visualized in the figure by the blue/green dashed lines that connect model entities with each other. Most of the adaptions stand for `SPI#definedOn` or `OM#requires` specifications with the commodity set individual as an object. `SupplyPar`, `FlowCostPar` and `FlowVar` need to be adapted by this on the side of the data entities, as well as that an individual of the new commodity-individual capacity parameters `IndividualUpperCommodityCapacities` needs to be created and related to the commodity set and the arc set. All formulation entities require an extra commodity set which is also reflected by their new semantic formulation types to be discussed below. The `IndividualCommodityCapacityConstraintsUp` individual needs to be created and related to the commodity set, as well as to the flow variables, arc sets and the newly introduced `IndividualUpperCommodityCapacitites`.

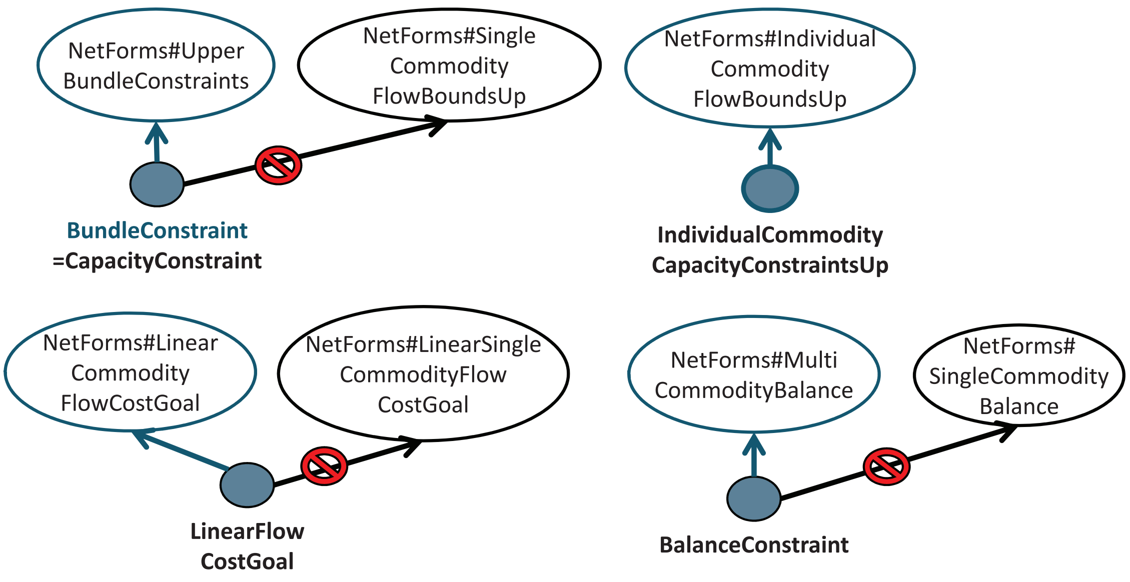

NetForms#Upper BundleConstraints
NetForms#Single Commodity FlowBoundsUp
NetForms#Individual Commodity FlowBoundsUp
BundleConstraint =CapacityConstraint
IndividualCommodity CapacityConstraintsUp
NetForms#Linear Commodity FlowCostGoal
NetForms#LinearSingle CommodityFlow CostGoal
NetForms#Multi CommodityBalance
NetForms# SingleCommodity Balance
LinearFlow CostGoal
BalanceConstraint

Figure 6.6.: Formulation type adaptions when extending the first min-cost-flow model towards the multicommodity flow model

The changes in the assertional specifications come along with changes in the semantic model entity types. On the side of the formulation entities, four new types need to be assigned whilst the old formulation type axioms for the three existing model entities need to be deleted. The changes and types are visualized in Figure 6.7. The new balance-constraint type `NetForms#MultiCommodityBalance` was explained together with his mdo-type requirements in Figure 6.4. The new type now has the additional requirement for a commodity set and requires commodity indexed types for flow variable collections and supply. The respective assertional adaptions were illustrated in Figure 6.5 as a whole but their order will be further discussed below. The type `NetForms#UpperBundleConstraints` to which the type `NetForms#SingleCommodityFlowBoundsUp` is changed, is a sole type of his own and can be extended by subtypes as it will be shown in the next subsection. The type requirements of the bundle constraint type are given by exactly one of `Net#CommoditySet`, `Net#ArcSet`, `Net#CommodityFlowCollection` and `Net#UpperCapacityCollection`. The type `NetForms#LinearCommodityFlowCostGoal` is similar to the old `NetForms#LinearSingleCommodityFlowCostGoal` but requires an additional commodity set as well as commodity types. To that end, it again can not be defined as a subtype. Nonetheless, the adaptions for the individual `LinearFlowCostGoal` only consist of the the additional specification of the required commodity set and the change of the type. The type `NetForms#IndividualCommodityFlowBoundsUp` is also no subtype of any single commodity flow bound type. As a new individual is created for the latter type in model, the type needs to be newly defined for that individual and all the assertional specifications from Figure 6.5 need to be created.

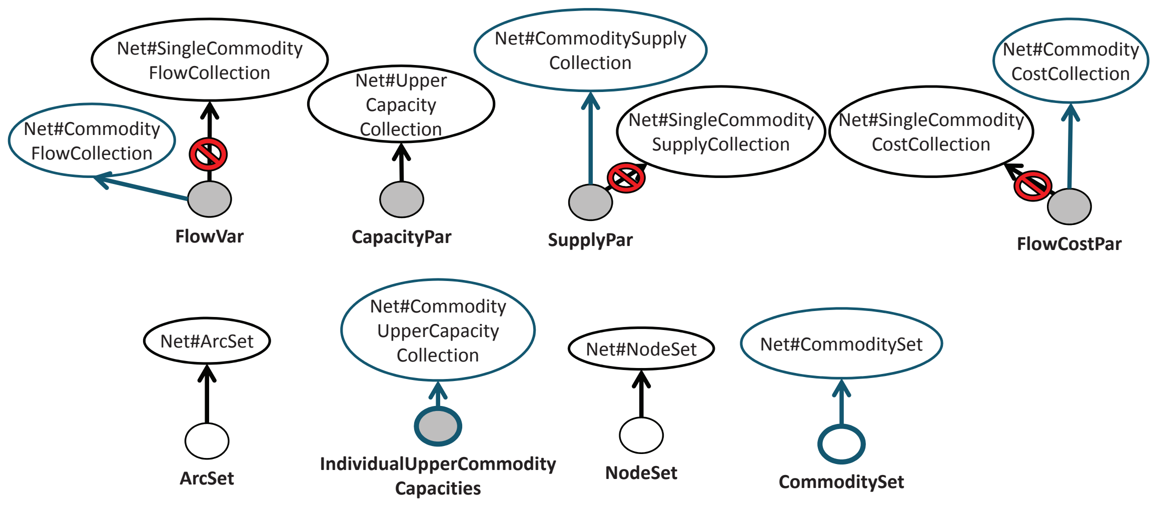

Net#SingleCommodity FlowCollection
Net#Commodity FlowCollection
Net#Upper Capacity Collection
Net#CommoditySupply Collection
Net#SingleCommodity SupplyCollection
Net#SingleCommodity CostCollection
Net#Commodity CostCollection
FlowVar
CapacityPar
SupplyPar
FlowCostPar
Net#ArcSet
Net#Commodity UpperCapacity Collection
Net#NodeSet
Net#CommoditySet
ArcSet
IndividualUpperCommodity Capacities
NodeSet
CommoditySet

Figure 6.7.: MDO-type adaptions when extending the first min-cost-flow model towards the multicommodity flow model

The type definitions of all data entites are given in Figure 6.7. As not every type changes, the real changes in types as well as the newly created individuals are represented by blue/green arrows and circles. The cardinality restrictions for the new data entity types for supply and flow have been visualized in Figure 6.4 whilst the specifications for the types `Net#CommodityCostCollection` and `Net#CommodityUpperCapacityCollection` can be looked up in the ontology specifications in digial form that accompany this thesis. All new data entity types, except `Net#CommoditySet` itself, differ from their single commodity pendants by additionally being defined on exaclty one `Net#CommoditySet`. From a specificational point of view, three data entity types need to be adapted with an additional relation of the commodity set individual, as well as that two data entities need to be created and typed the new, where in case of `Net#CommodityUpperCapacityCollection` additional specifications for the `SPI#definedOn` cardinality restrictions need to be created.

Consequences for Model Formulation:
The introduction of a commodity set and the respective adaptions of model entities in terms of their types and specifications reflect the example of the second motivational use case "*Recommended adaptions in model formulation*" from Section 4.1. Considering the situation where the commodity set has just been specified for the model, including the creation and typing of the individual `CommoditySet`, a specialized module in a model formulation environment could identify the missing usage of this set in the current model. To that end, it could look for mdo and formulation types that are *similar* to the ones actually contained in the model in the following way: The type requirements in form of cardinality restrictions (`OM#requires, SPI#definedOn, ...`) are widely comparable for a searched type as well as that the additional requirement for `exactly 1 Net#CommoditySet` occurs. This could then lead, e.g., to proposing a change of `NetForms#SingleCommodityBalance` to `NetForms#MultiCommodityBalance` as it was done in the example of this section. A challenge for this approach would be the definition of a suited "similarity measure" for matching the ontology class restrictions. In the example, e.g., not only an additional requirement for a commodity set occurs in the mdo-requirements of the new type, but also do the types for supply and flow vary in comparison to the old types that are contained in the model. Nonetheless, though different, the required supply and flow types of the `NetForms#MultiCommodityBalance` type to be found are somewhat as intended as they include the cardinality class restriction for a `Net#CommoditySet` that was also considered for the formulation type itself. To that end, a respective recommendation mechanism should not only select formulation types based upon a matching of class restrictions to the existing model, but as a consequence of an integrated decision which considers also the further meaningful adaptions of the model's data model. The recommendation to change the formulation type in the balance example would then come along with the further recommendation to also adapt the types of supply and flow entities in the model.

The automated recommendation of adaptions in the model formulation as outlined above is stated as a future research perspective at this point. This thesis can only present informal textual examples to illustrate important aspects of the model formulation technology to develop.

What should also be discussed now is the automation of adaptions after a modeling decision. This step was mentioned in the first motivational use case "*Formalized* *choices in modeling*". The situation after the commodity set has been added to the model is considered once again: All individuals that are being changed are assumed to have their types changed at first. The existing definitions for the entity type individual first of all stay as they are when changing the type. Now the entity type changes in this example require adding `OM#requires` and `SPI#definedOn` specifications as well as many changes of related types to multicommodity variants. The two latter tasks could be automatized as follows: At first, in a situation where a single individual of type `Net#CommoditySet` exists in a model, the respective axioms for cardinality restrictions that are so to say additionally required from the new type could be guessed and inserted automatically. An example would be the relation of the commodity set individual after changing the network balance constraints to the type `NetForms#MultiCommodityBalance`. Here, a commodity set needs to be related where there was no such requirement before. Unfortunately, this modification might not always be correct as the selected commodity set might not be the intended one and another, maybe newn commodity set would have to be related in the oppinion of the modeler. To that end, automatizations in model formulation could be implemented in an experimental fashion such that ontologically consistent models are generated, but interaction with the modeler is required for obtaining an intentionally correct model. The degree of automation should be configured before using respective tools.

The same conclusion can be drawn in the case of changing type requirements: In the context of the network balance example, the adaption of the type `Net#SingleCommodityFlowCollection` to the type `Net#CommodityFlowCollection` could also be triggered automatically after the new formulation type for balance was defined. By this change, two further aspects should be considered: At first, the type change of the identified mdo-individual for flow might not be intended, as the individual might have to be replaced by a new, additional one in the opinion of the modeler. And second, the type change of the mdo flow-individual triggers further adaptions in a recursive fashion. I.e., the change to `Net#CommodityFlowCollection` requires an additional `SPI#definedOn` axiom with an object of the type `Net#CommoditySet`. An automated relation of the single commodity set individual contained in the model then bears the same questions for the modeler's intention as above.

6.1.3. Two Types with Generic Summations

This subsection introduces two formulation types that are generic in the way how their inner expression structure can be instantiated. Based upon a decoupling of expression types that form another ingredient for the specification of formulation types, a generic number of inner summation terms that differ amongst each other can be considered. This subsection will discuss the ontology specifications for the latter types on class level. In the next subsection, a further generalized min-cost-flow model will be presented which uses these types. The instance level specification will be further discussed there.

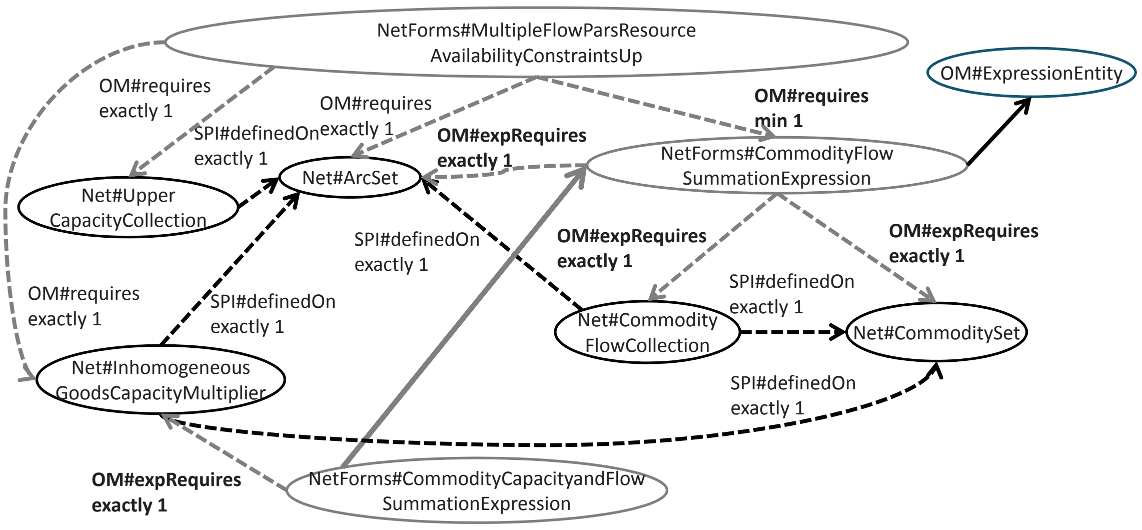


Figure 6.8.: Type specification for a generic resource availability type

The first type to be discussed is the type `NetForms#MultipleFlowParsResourceAvailabilityConstraintsUp`. The type specifications are visualized in Figure 6.8. A target statement for the type (without model context) would be

```
              s.t.  ResourceAvailabilityConstraintsUp {(i,j) in Arcs}:
    sum{k in ContinuousCommodities}(CommodityArcFlowContinuous[i,j,k])
                                          + sum{k in IntegerCommodities}
                            (rho[i,j,k] * CommodityArcFlowInteger[i,j,k])
                                                   <= UpperCapacity[i,j];.
```
(6.6)

Without knowing the complete model context one can observe that the described group of constraints bounds the flow through an arc in a fashion such that different flow variables for different commoditites are aggregated by sums to yield an overall quantity of capacity to be bounded from above. The flow variables have especially been split into two families of variables, making it necessary to define two different inner summations. For the `CommodityArcFlowInteger`, a parameter `rho` is multiplied to the flow variable, making it possible to weight the consumption of network capacity differently for every commodity. The respective idea can, e.g., be found in the book on network flows from Ahuja et al., [AMO93]. The other family of variables, `CommodityArcFlowContinuous` has a different expression structure in the sum which does not require the latter parameter. The story behind this modeling is a different treatment of commodity flow goods in the respective model, where continuous commodities represent liquids that can be treated uniformly in terms of some transport capacity, whilst integer commodities represent some items of different size to be transported. The constraint can be seen as a generalization of the bundle constraints from the last subsection. By viewing the statement, certain type requirements (cardinality class restrictions) from Figure 6.8 become obvious: Exactly one `Net#ArcSet` is required for indexing over the network arcs as well as that exactly one of `Net#UpperCapacityCollection` is required. The `Net#InhomogeneousGoodsCapacityMultiplier` is also required exactly once as the formulation type in principle treats flow goods with different portions of the capacity. If the consideration of inhomogeneous flow goods was intended to be left out for all commodities, the usage of a more simple type would be indicated. How the parameter can be left out for only parts of the commodities will be discussed right now.

The relevant type requirement to make the formulation type generic in a way as described above is given by the cardinality restriction `OM#requires min 1 NetForms#CommodityFlowSummationExpression`. By this, a generic and non negative number of the expression type `NetForms#CommodityFlowSummationExpression` individuals can be speficied for a constraint group individual. These individuals represent inner summations in the target statements. The expression types themselves can be used with different data conceptualizations to yield a sum over `Net#CommoditySet` members which sums up a family of flow variables (`Net#CommodityFlowCollection`). The latter two mdo-types are brougth into cardinality restrictions by the property `OM#expRequires` which provides a pendant of `OM#requires` for the case of `OM#ExpressionEntity` types (recapitulate the `OM` vocabulary definitions in Subsection 5.3.2). Instances of `OM#ExpressionEntity`, such as individuals of the subclass `NetForms#CommodityFlowSummationExpression`, are not to be considered as model entities and therefore provide an extra construct in the repertoire of modeling components that, speaking in an OO-design vocabulary, enforces decoupling.

The next design concept to use for the general resource availability type is inheritance. As the `NetForms#CommodityFlowSummationExpression` only allows to sum up simple flow variable collections over commodities, an extension is required in order to consider the multiplicative parameter `rho` for the `CommodityArcFlowInteger` variables. This is done by the aid of the expression subtype `NetForms#CommodityCapacityandFlowSummationExpression` which inherits the `OM#expRequires` requirements from his father and extends the latter by the requirement `OM#expRequires exactly 1`

`Net#InhomogeneousGoodsCapacityMultiplier`. The latter mdo-type can be used for the respective multiplicative parameters `rho`. Two different expression individuals can be specified with the aid of the latter discussed types for the instance structure of the target statement (6.6), where due to the subsumption relation, the cardinality restriction `NetForms#MultipleFlowParsResourceAvailabilityConstraintsUp OM#requires min 1 NetForms#CommodityFlowSummationExpression` is still satisfied. An illustration of the respective assertional knowledge in the context of a min-cost-flow model will be given in the next subsection.

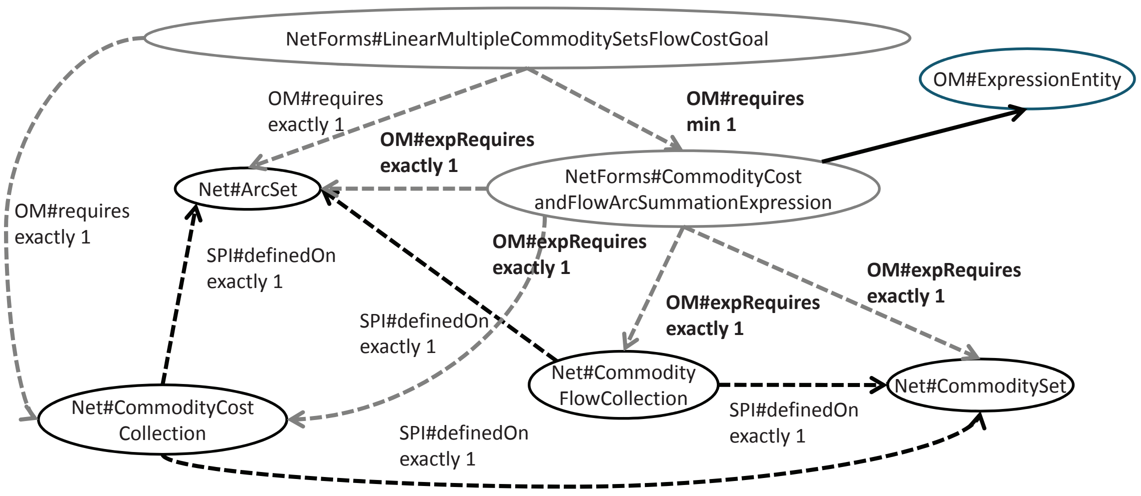


Figure 6.9.: Type specification for a multiple commodity sets min-cost-flow goal

To continue the considerations of this subsection, another summation-generic formulation type, which is to be used in the min-cost-flow model of the next subsection, will be presented. Figure 6.9 shows the type specification for the type `NetForms#LinearMultipleCommoditySetsFlowCostGoal`. The type represents an objective of a min-cost-flow model that allows to consider multiple commodities, where the flow variable collections are split up again. As in the case of the resource availability type presented above, the decoupling of an `OM#ExpressionEntity` subtype, `NetForms#CommodityCostandFlowArcSummationExpression` in this case, allows to instantiate a generic number of possibly different inner summations. The example statement of the next subsection that is to be generated with this type is more simple as in the case of the resource availability type. More specific, the expression entity type for the two summations will be the same, and only the individuals and consequently the identifiers of commodity sets and flow variables will differ. Hence, no further subtype of the expression type is used. The remainder of the type specification should become clear from the figure. Further information on statements and instance knowledge is to be given immediately in the next subsection.

```
#General min−cost−flow model as it is traceable in the book of Ahuja et al., "Network
    Flows" until p. 650 ff.

set Nodes;
set Arcs within {Nodes, Nodes};

# Split the commodities into continuous and integer ones in order to consider only some
    as indivisible goods.
set ContinuousCommodities;
set IntegerCommodities;

param Cost{Arcs, ContinuousCommodities union IntegerCommodities};
param LowerCapacity{Arcs};
param UpperCapacity{Arcs};
param mu{Arcs,ContinuousCommodities union IntegerCommodities} > 0;
# mu is the multiplier to model loss and gain in arcs. If mu< 1 arcs are lossy. If mu >
    1 they are gainy
param rho{Arcs,IntegerCommodities} >= 0; #parameter for inhomogeneous goods
param CommodityLowerCapacity{Arcs, ContinuousCommodities union
    IntegerCommodities};
param CommodityUpperCapacity{Arcs, ContinuousCommodities union
    IntegerCommodities};
param SupplyDemandInteger{Nodes, IntegerCommodities} integer;
param SupplyDemandContinuous{Nodes, ContinuousCommodities};

var CommodityArcFlowContinuous{(i,j) in Arcs, k in ContinuousCommodities };
var CommodityArcFlowInteger{(i,j) in Arcs, k in IntegerCommodities } integer;
```

Figure 6.10.: A general min-cost-flow model - Data definitions

6.1.4. Ontology Specifications for a general Min-Cost-Flow Model

This subsection discusses network formulation types and operations in model formulation at the example of a further generalized min-cost-flow model. The model includes two sets of commodities with and without an integrality requirement for the respective commodity flows. Furthermore, the network balance constraints are generalized to a variant that allows for gain/loss. The bundle constraints for multiple commodities are generalized to the resource availability constraints discussed in the last subsection. Finally, multiple individuals of the same formulation types will further enforce the idea of reuse as well as that the subsumption of formulation types will be exploited.

The AMPL model for the generalized min-cost-flow-problem is given in Figures 6.10 and 6.11. Data and goal/constraint declarations are distributed over both figures. As

```
minimize MultiCommodityFlowCost: sum{k in ContinuousCommodities}(sum{(i,j) in Arcs
    }(Cost[i,j,k] ∗ CommodityArcFlowContinuous[i,j,k])) + sum{k in IntegerCommodities
    }(sum{(i,j) in Arcs}(Cost[i,j,k] ∗ CommodityArcFlowInteger[i,j,k]));

s.t. ContinuousCommodityGeneralizedMassBalances{k in ContinuousCommodities, i in
    Nodes}: sum{(i,j) in Arcs}(CommodityArcFlowContinuous[i,j,k]) − sum{(j,i) in Arcs
    }(mu[j,i,k] ∗ CommodityArcFlowContinuous[j,i,k]) = SupplyDemandContinuous[i,k];
s.t. IntegerCommodityGeneralizedMassBalances{k in IntegerCommodities, i in Nodes}:
    sum{(i,j) in Arcs}(CommodityArcFlowInteger[i,j,k]) − sum{(j,i) in Arcs}(mu[j,i,k] ∗
    CommodityArcFlowInteger[j,i,k]) = SupplyDemandInteger[i,k];

s.t. ResourceAvailabilityConstraintsUp{(i,j) in Arcs}: sum{k in ContinuousCommodities
    }(CommodityArcFlowContinuous[i,j,k]) + sum{k in IntegerCommodities}(rho[i,j,k] ∗
    CommodityArcFlowInteger[i,j,k]) <= UpperCapacity[i,j];
s.t. ResourceAvailabilityConstraintsLow{(i,j) in Arcs}: LowerCapacity[i,j] <= sum{k in
    ContinuousCommodities}(CommodityArcFlowContinuous[i,j,k]) + sum{k in
    IntegerCommodities}(rho[i,j,k] ∗ CommodityArcFlowInteger[i,j,k]);

s.t. IndividualCommodityCapacityConstraintsContinuousUp{k in
    ContinuousCommodities, (i,j) in Arcs}: CommodityArcFlowContinuous[i,j,k] <=
    CommodityUpperCapacity[i,j,k];
s.t. IndividualCommodityCapacityConstraintsIntegerUp{k in IntegerCommodities, (i,j) in
     Arcs}: CommodityArcFlowInteger[i,j,k] <= CommodityUpperCapacity[i,j,k];
s.t. IndividualCommodityCapacityConstraintsContinuousLow{k in
    ContinuousCommodities, (i,j) in Arcs}: CommodityLowerCapacity[i,j,k] <=
    CommodityArcFlowContinuous[i,j,k];
s.t. IndividualCommodityCapacityConstraintsIntegerLow{k in IntegerCommodities, (i,j)
    in Arcs}: CommodityLowerCapacity[i,j,k] <= CommodityArcFlowInteger[i,j,k];
```

Figure 6.11.: A general min-cost-flow model - Formulation definitions

a reference for the generalized formulations contained in the model, Ahuja et al. [AMO93] is mentioned once again. In comparison to the second min-cost-flow model from Subsection 6.1.2, two commodity sets `ContinuousCommodities` and `IntegerCommodities` are defined. The split of these commodity sets reflects the separated treatment of the two flow variable collections `CommodityArcFlowContinuous` and `CommodityArcFlowInteger`. As outlined in the last subsection, this separated treatment is due to a different treatment of commodity flow goods in the model, where continuous commodities can, e.g., be imagined as liquids that can be treated uniformly in terms of some transport capacity, whilst integer commodities represent some items of different size. The different size of the integer commodities is modeled by multiplication with the additional parameter `param rho{Arcs, IntegerCommodities} >= 0` and the integrality requirement of `CommodityArcFlowInteger` reflects the assumption of the indivisibility of the respective goods. The separated treatment of the commodities and flows enforces the construction of implicit set-unions for the declarations of the parameters `Cost, mu, CommodityLowerCapacity` and `CommodityUpperCapacity`. Though procedural elements in AMLs such as AMPL could in principle be applied to generate the respective elements iteratively, without indexing, the usage of indexing constitutes a well established practice in AMLs. The required union-statements in AMPL will be mirrored by union-set specifications in the ontology formalism. The latter will then not be derived to an explicit set declaration but will lead to implicit set-expressions inside the indexings. More information will follow in the text below.

The description of the model shall now be continued with the declaration of the two groups of network balance constraints, see, Figure 6.11. The two statements for `ContinuousCommodityGeneralizedMassBalances` and `IntegerCommodityGeneralizedMassBalances` introduce network balance constraints for each group of the commodity flow variable collections. The formulation type in both cases is `NetForms#GainLossGeneralizedMultiCommodityBalance` and has been presented in Figure 6.4 of Subsection 6.1.2. The type provides a multicommodity pendant of the gain-loss balance type `NetForms#GainLossGeneralizedSingleCommodityBalance` that was discussed in the context of the single commodity balance type hierarchies in Subsection 6.1.1. Hence, subsumption can be exploited when editing the second min-cost-flow model towards the representation of the third model. The respective generalized balance statements in the model also introduce the usage of a parameter `mu` that allows to model a gain or loss of flow goods in the incoming arcs. The resource availability constraints in a general form, as discussed in the last subsection, can also be found for bounding the respective capacities both from above and from below. Similar to that, the objective of the model is a target statement for the type `NetForms#LinearMultipleCommoditySetsFlowCostGoal` from the last subsection. Multiple commodity- and arc-individual flow bounding constraints are contained in the model. The four respective groups of constrains result from the separate treatment of integer and continuous commodities as well as the consideration of upper and lower bounds.

The ontology formulation of the third min-cost-flow model shall be explained now. The modeling of implicit set-unions will be exemplified first.
Figure 6.12 provides a specification of the two commodity sets `ContinuousCommoditySet` and `IntegerCommoditySet` as well as an implicit union-set `UnionSetCommodities`. The implicit union-set individual of the type `SPI#UnionSet` is marked as `SPI#NonInstantiable true` such that no

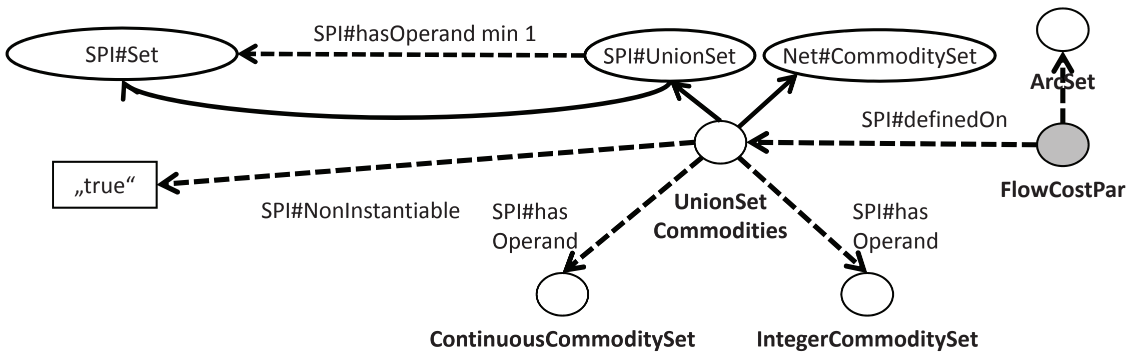


Figure 6.12.: Specification of implicit set unions for multiple commodity sets

set-declaration statement will be generated for the union-set individual in the AML Derivation. `UnionSetCommodities` is related to the two elements of the set union by axioms using the `SPI#hasOperand` property. In the result, a parameter such as `FlowCostPar` can be specified to be defined on the union set and hence a union set-expression such as `ContinuousCommodities union IntegerCommodities` will be generated in the indexing of the AML declaration of the parameter `Cost`, see, Figure 6.10. As `UnionSetCommodities` is of type `Net#CommoditySet`, the respective cardinality restriction from the flow cost parameter mdo-type is satisfied. When viewing the model formulation of the third model as an extension of the second one, an adaption of the commodity set specifications is a first step to perform. In order to do so, the old `CommoditySet` individual can be renamed to `IntegerCommoditySet`. The renaming of the individual can, e.g., be performed by refactoring methods in ontology editors such as Protégé. In addition to that, the new individuals for the continuous commodities and the union of commodities need to be created, related and typed. Other specifications making use of the old `CommoditySet` which now is the `IntegerCommoditySet` have to be adapted whend their indexing is not intended to relate to the set of integer commodities. In case of the parameter `Cost` from Figure 6.12, the respective `SPI#definedOn` axiom has to be adapted to one with the new object `UnionSetCommodities`.

The next specifications to discuss are the assertional specifications for the upper resource availability constraints (`ResourceAvailabilityConstraintsUp` in the AMPL model). According to the type specifications for `NetForms#MultipleFlowParsResourceAvailabilityConstraintsUp` from Figure 6.8, a constraint type individual `ResourceAvailablityUp` is to be related to minimum one expression individual and some data entities. The respective specifications can be found in Figure 6.13. As mentioned above, the model formulation in the ontology can be seen as the result of working the second

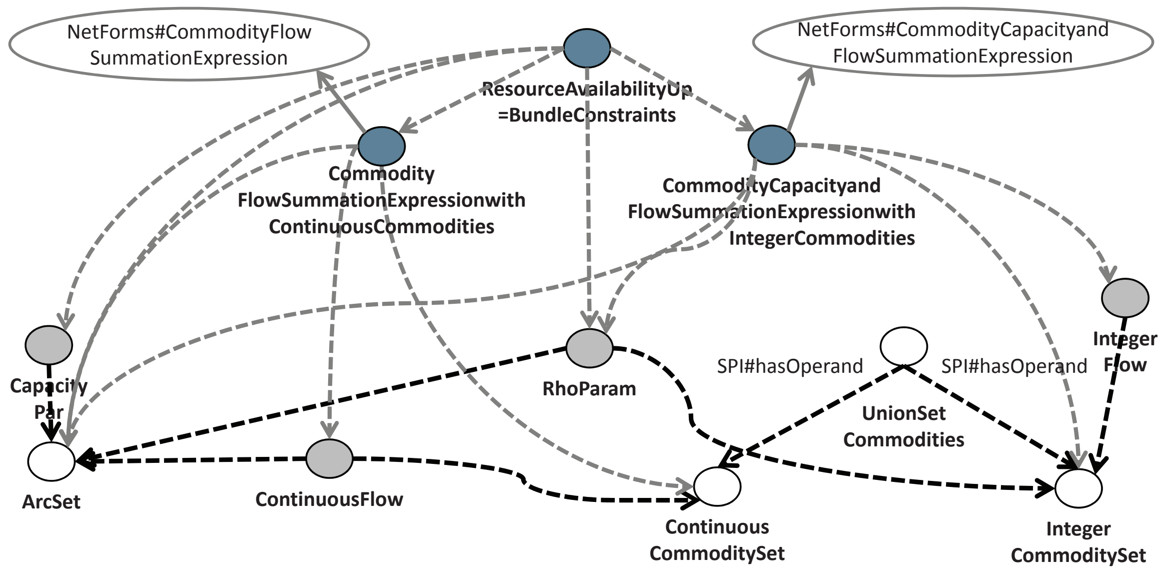

NetForms#CommodityFlow SummationExpression
NetForms#CommodityCapacityand FlowSummationExpression
ResourceAvailabilityUp =BundleConstraints
Commodity FlowSummationExpressionwith ContinuousCommodities
CommodityCapacityand FlowSummationExpressionwith IntegerCommodities
Integer Flow
Capacity Par
RhoParam
SPI#hasOperand
SPI#hasOperand
UnionSet Commodities
ArcSet
ContinuousFlow
Continuous CommoditySet
Integer CommoditySet

Figure 6.13.: Instance level specifications for the upper resource availability constraints

min-cost-flow model towards the one of this subsection. Therefore, the old `BundleConstraints` individual with its specifications can be used to build the upper resource availability constraints individual. The first steps are a refactoring in form of a renaming to `ResourceAvailabilityUp`, as well as an adaption of the formulation type. Afterwards, further individuals need to be related. These individuals possibly need to be created, or their specifications need to be adapted in a recursive fashion. The result to be achieved after multiple steps then has the following form: Two expression individuals `CommodityFlowSummationExpressionwithContinuousCommodities` and `CommodityCapacityandFlowSummationExpressionwithIntegerCommodities` represent the two inner sums of the target statement, where only the latter individual is connected to the individual `RhoParam` accoring to the specification for the expression (sub-) type `NetForms#CommodityCapacityandFlowSummationExpression`. The respective expression type allows to derive the term `sum{k in IntegerCommodities} (rho[i,j,k] * CommodityArcFlowInteger[i,j,k])` with a product of the respective parameter. Both inner summation individuals refer to different commodity sets and flow variable collections. The respective specifications are visualized by grey arrows in the figure. Then, also some data entities need to be related, occasionally including the steps to create, type and relate the data entities themselves further. Considering, e.g., `RhoParam`, it would be likely that such an individual does not exist in the model before the specification of a resource availability constraint and hence a recursive specification of the data individual itself would be required. The adaptions for the flow variable collection individuals in contrast to that could have already been performed in the context of adapting another formulation entitie's specification, or directly after splitting the commodity sets for the intended model extension. Included steps would have been a renaming of the old `FlowVar` individual to `IntegerFlow` and the creation, typing and relation of a new individual `ContinuousFlow`.

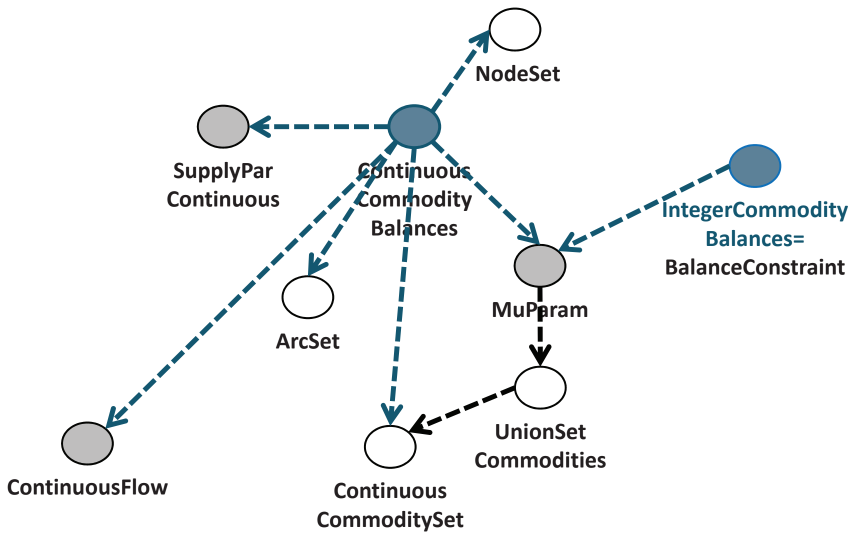


Figure 6.14.: Adaptions in the assertional specifications of the balance constraints when extending the ontology specification towards the third min-cost-flow model

Of further interest are the assertional specifications for the network balance constraints. As in comparison to the second model, a gain/loss generalized formulation type is used, the respective specification for the existing balance individual needs to be adapted by adding a `OM#requires` axiom with the gain and loss parameter individual `MuParam` as an object. The individual can again be present in the model or will be created in a recursive fashion. If the balance constraint individual represents the integer commodities, a refactoring of the name to `IntegerCommodityBalances` is indicated. On the assertional side of the data entity specifications, the commodity set should already represent the set of integer commodities, as the first operation in adapting the ontology specification for the third model would be a split of the commodity sets. As the latter steps create a balance constraint that only covers the integer commodities, another individual for the continuous commodities needs to be created. This balance constraint then requires complete new assertional specifications and a type definition as it is indicated in Figure 6.14 by blue/green arrows. A complete ontology specification of the min-cost-flow model of this subsection can be found on the digital material accompanying this thesis.

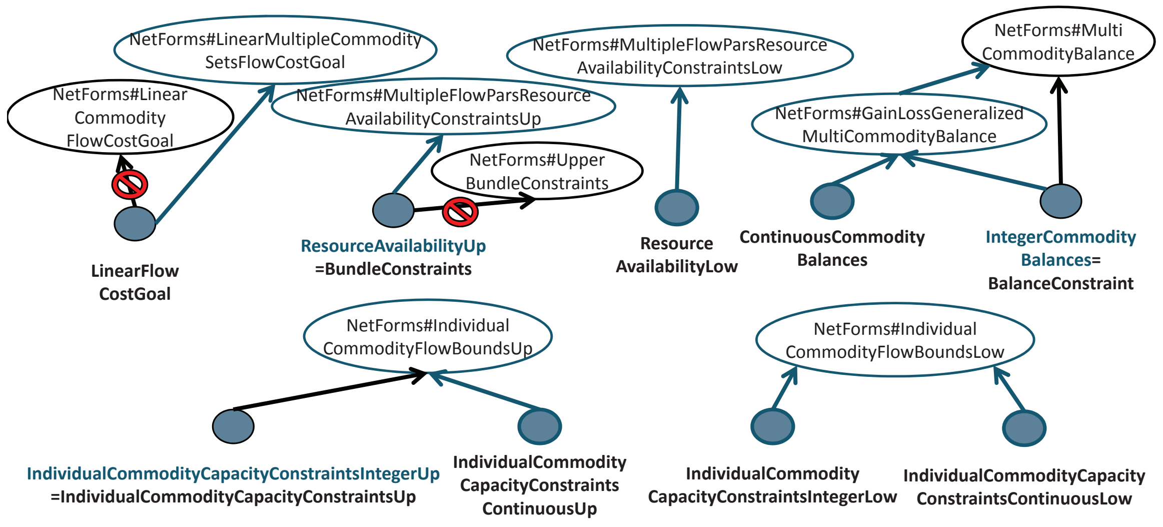

NetForms#LinearMultipleCommodity SetsFlowCostGoal
NetForms#MultipleFlowParsResource AvailabilityConstraintsLow
NetForms#Multi CommodityBalance
NetForms#Linear Commodity FlowCostGoal
NetForms#MultipleFlowParsResource AvailabilityConstraintsUp
NetForms#GainLossGeneralized MultiCommodityBalance
NetForms#Upper BundleConstraints
LinearFlow CostGoal
ResourceAvailabilityUp =BundleConstraints
Resource AvailabilityLow
ContinuousCommodity Balances
IntegerCommodity Balances= BalanceConstraint
NetForms#Individual CommodityFlowBoundsUp
NetForms#Individual CommodityFlowBoundsLow
IndividualCommodityCapacityConstraintsIntegerUp =IndividualCommodityCapacityConstraintsUp
IndividualCommodity CapacityConstraints ContinuousUp
IndividualCommodity CapacityConstraintsIntegerLow
IndividualCommodityCapacity ConstraintsContinuousLow

Figure 6.15.: Formulation type changes when extending the ontology specification towards the third min-cost-flow model

In addition to the adaptions of the assertional model knowledge, entity type adaptions and specifications need to be carried out for the old and new individuals. In case of the formulation type specifications, Figure 6.15 provides an overview. New specifications are visualized in blue/green. There are five newly created formulation individuals, namely a lower resource availability constraint, a continuous commodity balance and multiple individual commodity flow bounds. Type changes for existing individuals can be observed. Here, the balance constraint individual `IntegerCommodityBalances` is of special interest. The type of the existing individual is changed to a subclass by an additional type-axiom, which allows it to reuse the existing specifications and just extend them by relating a parameter as it has been discussed above. For three of the formulation types from the image, there are each two individuals with different instance level specifications in the model. This further enforces the idea of reuse.
Three formulation types occur in the model that make use of the decoupling of expression types. The three formulation types are for the objective as well as for the upper and lower resource availability constraints. As every AMPL statement consists of two inner summations, one would expect six respective expression individuals. But as the type definitions in Figure 6.16 show, only four expression individuals are required. This is due to the similarity of the upper and lower resource availability constraints which allows it to reuse the respective subexpressions.
For the meta domain ontology type specifications, Figure 6.17 shows the respective

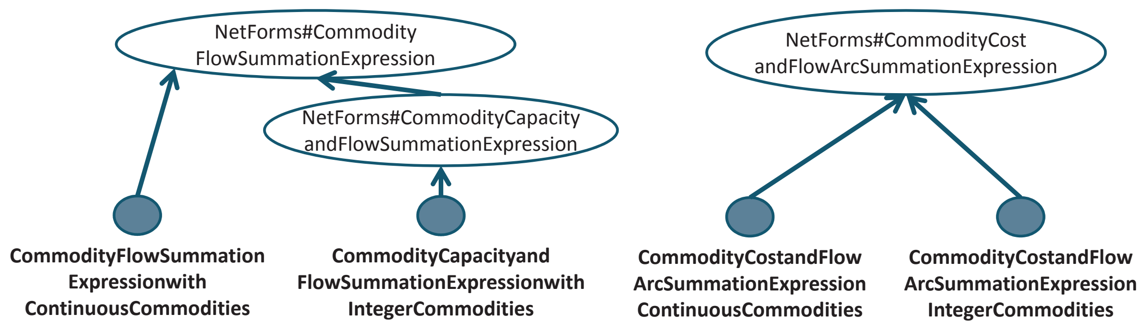

NetForms#Commodity FlowSummationExpression
NetForms#CommodityCapacity andFlowSummationExpression
NetForms#CommodityCost andFlowArcSummationExpression
CommodityFlowSummation Expressionwith ContinuousCommodities
CommodityCapacityand FlowSummationExpressionwith IntegerCommodities
CommodityCostandFlow ArcSummationExpression ContinuousCommodities
CommodityCostandFlow ArcSummationExpression IntegerCommodities

Figure 6.16.: Expression type definitions for the third min-cost-flow model

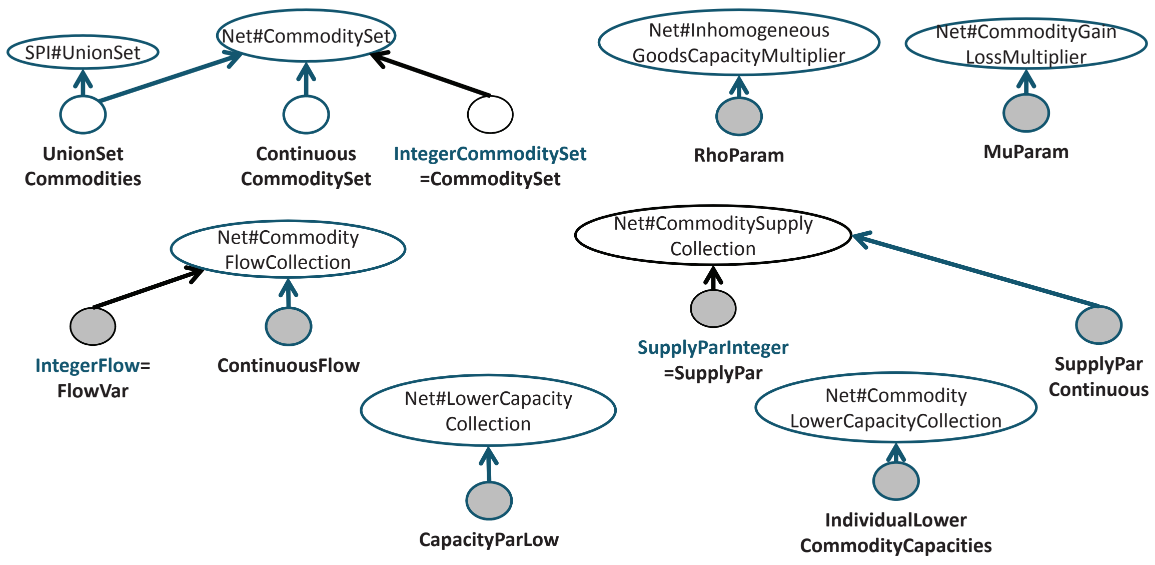

SPI#UnionSet
Net#CommoditySet
Net#Inhomogeneous GoodsCapacityMultiplier
Net#CommodityGain LossMultiplier
UnionSet Commodities
Continuous CommoditySet
IntegerCommoditySet =CommoditySet
RhoParam
MuParam
Net#Commodity FlowCollection
Net#CommoditySupply Collection
IntegerFlow= FlowVar
ContinuousFlow
SupplyParInteger =SupplyPar
SupplyPar Continuous
Net#LowerCapacity Collection
Net#Commodity LowerCapacityCollection
CapacityParLow
IndividualLower CommodityCapacities

Figure 6.17.: MDO type changes when extending the ontology specification towards the third min-cost-flow model

type definitions. When working the second model towards the third one, no deletions of mdo type axioms take place, but rather are just additional individuals created. The respective new individuals and their specifications are once more visualized by blue/green ellipsoids and arrows.

Concluding Remarks:
Let us now conclude the consideration of the third min-cost-flow model. The specifications of this subsection demonstrated the reusage of formulation types through multi-instantiation in several cases. Furthermore, the flexible types for network flow goals and resource availability types from the last subsection were demonstrated in a consistent model context. Union-set specifications were

introduced as a novel feature. The adaption of a formulation entity to a subclass of its former type was demonstrated for network balance constraints. If the type adaption is performed manually, an automated procedure may again look for existing individuals for the gain and loss parameter and relate them in case of an existing and unique individual. But full automatization of this adaption step seems hard to achieve in every model context. E.g., if no such parameter exists in the model, an individual has to be created the new and a `SPI#definedOn` axiom with the correct commodity set individual as an object has to be generated. In the presented example from Figure 6.14, this individual would refer to the union-set `UnionSetCommodities` that should have been introduced to the model before. A rule that prefers such set-unions over basic sets in the automatized step could be written, but its application nonetheless would just provide a guess for the model formulation. If multiple individuals for a gain/loss parameter exist in the model, choosing the correct one could be complicated though a matching of the individual specifications might be applied. Nonetheless, a model formulation system can not know a modeler's intention without a respective input in the model formulation process.

6.2. Discussion of MDO and Formulation Ontology Type Design

This section shall discuss the ontology design of model entity types and the strucure of the respective ontologies. Section 5.3 of Chapter 5 introduced vocabularies that are required to specify model entity types. Besides some simple examples in Chapter 5, the specification of exemplary reusable types from the network flow domain was a topic of the first section of this chapter. The presented formulation and mdo types were specified in terms of certain cardinality class restrictions and were partially grouped into subsumption hierarchies. This section will explain which cardinality restrictions should be used when defining a type, as well as what kind of embedding into the ontology structure is to be considered. At first, the cardinality restrictions in the desing of formulation and meta domain ontologies will be concluded in short:
All examples of formulation types, including the ones that will follow in the next section, only use cardinality class restrictions of the following form:

- The cardinality class restriction for the property `OM#requires` is `exactly n` if the object is a data entity type (no expression). In the typical examples given so far, the value of `n` was one. An example of a specification with `n=2` required node-sets will be discussed in Subsection 6.3.2. The value of `n` can also be set to zero in order to explicitely forbid specifications.
- The cardinality class restriction for the property `OM#requires` is `min 1` if the object is a `OM#ExpressionEntity`. The expression entity types themselve are defined with sharp cardinality restrictions of the form `OM#expRequires exactly m` with `m` typically being set to one.

On the side of the data entity types that are organized in meta domain ontologies, the following applies:

- Cardinality class restrictions of the form `exactly n` are used with the property `SPI#definedOn` where in the typical examples, `n` is set to one. Not every data entity type allows for such definedOn's. In case of `SPI#SimpleParameter`, they are explicitely forbidden, i.e., `n` is set to zero.
- Cardinality class restrictions of the form `max 1` are used with the property `SPI#WithinCartProductTwoSame`.
- Cardinality class restrictions of the form `min 1` are used with the property `SPI#hasOperand`.

Interpretation of the cardinality class restriction design:
Recapitulate the single commodity balance hierarchies from Subsection 6.1.1 as well as the multicommodity flow types introduced in the subsections following thereon. Reusable formulation types were defined by class restrictions with mdo types in their range, as well as that the mdo types themselves had to be defined by class restrictions. Two respective vocabularies `NetForms` and `Net` were introduced. In both ontologies, subsumption was used as a technique to extend father types in subtypes by inheritance of the cardinality restrictions, possibly specializing their range or adding new class restrictions. The inheritance concept was also exploited in a multi-inheritance fashion. But inheritance was not always possible. As different mdo-conceptualizations for demand/supply or single and multiple commodities introduced isolated classes in the mdos, the distinction propagated to the formulation types. The distinctions in the mdos were also due to a modeling principle that prefers sharp cardinality restrictions, and hence sharp type definitions, over flexible ones. This is motivated by the idea to lay a foundation for automatizations in the model formulation, especially, the automated recommendation of formulations. The possibilities to analyze type definitions are both required and enforced by the design decisions. The design principles that are concluded in this section lead to a definition of many different formulation types. This makes the automated treatment of modeling knowledge necessary. On the other hand, as it has already been mentioned, these automation capabilities are explicitely desired and hence imply a design with explicit knowledge of a "type interface".

An exception from the modeling with cardinality restrictions of the form "`exactly n`" was given by the flexible types for resource availability constraints and a network flow objective in Subsection 6.1.3. There, it was possible to relate a generic number of expressions, each leading to different inner summation terms. Nonetheless, the flexible types still have many specifications prescribed on class level that should support their automated finding by a recommendation/matching formalism.

As the next section will demonstrate, the organization of entity types by (multi)-inheritance is not only restricted to definitions in a single ontology. Specialized domains such as *water networks* may introduce subtypes of general network types. Specialized domain ontologies may also import standard vocabularies and extend them by inheritance. The reusage of types can be enforced by providing more and more flexible formalizations in subdomains. If models use the same constraint type for more than one individual, the reusability of formulation types can also be exploited. This is further motivated by the fact that certain formulation types encapsulate multiple AML statements in larger blocks. Hence, an instantiation of these types may save the error-prone work of specifying multiple related statements, e.g., for the linearization of a curve in AMLs. The latter example of linearization and smoothing will be discussed in Subsection 6.3.3 of the following case study section.

6.3. Considerations for a Water Network Case Study

In this section, further formulation type formalizations will be introduced in the context of another case study. The application domain of water distribution systems will be investigated together with planning tasks in this domain. A network design problem will serve as a basis for discussing the formalizations. Formulations from another planning task of the domain will be discussed based upon the initial observations for the network design problem. Important examples cover types for linearizing and smoothing nonlinear pipe headloss constraints, as well as such ones that model the short-term activation and deactivation of pumps in different fashions, i.e., such that there is a choice between an integrality-requiring linear, and a continuous, nonlinear, formulation. At first, a short introduction to the planning of water distibution systems and the underlying mathematical models and equations will be given. This provides the basis for discussing the ontology formalizations in the remainder of this chapter. Chapter 7, which follows on this section, then provides a model and numerical results for the problem of pipe renewal planning. The latter problem can be seen as related to the problem of pipe network design that will be treated within the case study of this section.

6.3.1. Introduction to Water Distribution Systems Planning

Distribution networks for water constitute an important part of our infrastructure. The supply of water in drinking quality has to fulfill different requirements on availability and quality, especially in unforeseen demand situations such as fire or mass events. To that end, the analysis and design of water distribution systems (WDS) constitutes an active area of research for people from diverse fields such as engineering and the computer sciences.
A water distribution system is typically represented by a network model that

distinguishes six types of components as there are suppy/reservoir nodes, tank nodes, general junction nodes (with demands), pipe arcs and two special types of arcs for modeling network pumps and valves. An example network is visualized in Figure 6.18.

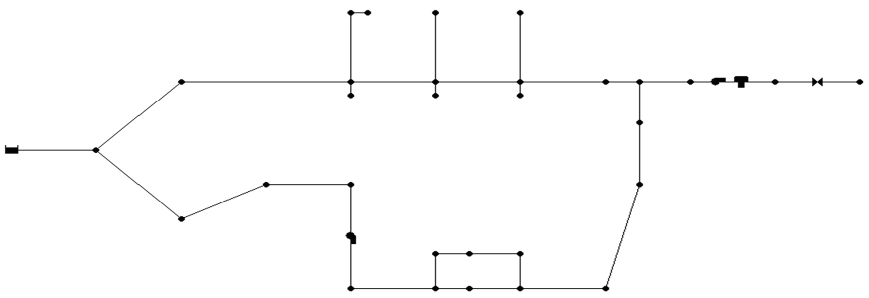


Figure 6.18.: Exemplary drawing of a WDS (generated with EPANET 2, [Ros00])

Besides the modeling of the distribution network's structure, the network's state at a certain point of time has to be represented by some values which are determined by parameters and physical laws. Hydraulic variables of interest are the arc's flow values and the nodal heads. Their status at a certain point of time is influenced by the network's demand and supply scenario, the boundary heads as well as the pipe characteristics to be determined in terms of pipe length, diameter and roughness.
Mass Balance:
The first typical equations to be included in a mathematical system for describing network hydraulics model the conservation of mass. Considering a network model at a static point of time, the mass balance at the junction nodes of the network is formalized in terms of the volumentric flow rates through the pipes. For the set of network pipes P, denote by $Q_{i,j}$ the value of flow through a pipe (i,j) at a static point of time. In many models for water network optimization, this flow is unrestricted in sign. A reference orientation of the network pipes is assumed by modeling the pipes as directed arcs, which allows it to represent flow into the pipe direction by positive values and vice versa, to represent flow in direction contrary to the reference orientation, by negative values. Considering the set J of junction nodes inside a network with positive demand values $D_k \geq 0, k \in J$, the mass balance at a static state reads

$$\sum_{(i,k)\in P} Q_{i,k} - \sum_{(k,j)\in P} Q_{k,j} = D_k \quad \forall k \in J. \tag{6.7}$$

This representation assumes water as a fluid due to its incompressibility. The mass balance equation in a constraint form as above represents an example for the demand-oriented balance type `NetForms#SingleCommodityBalanceDemandOriented` from Subsection 6.1.1.

Pipe Friction caused Headloss:

The second system of equations to be included in mathematical models for simulation and optimization is the pipe friction caused headloss in standard pipes (no pumps or valves). Denoting by $H_i, i \in J$ the nodal head at a junction node, there are different models for determining the headloss between two nodes (i, j) constituting a network pipe. Besides the formula of Hazen-Williams, which is often used in the anglo-american area and provides a valid approximation in the case of turbulent flow, the formula of Darcy-Weisbach in principle provides a valid framework for computing pipe friction based headlosses for different types of flow. The formula of Darcy-Weisbach in SI units will be used in the examples of this thesis. The classical formula is represented in terms of pipe velocities and can, e.g., be found in [KCLH08]. But as volumetric flow rates are the standard variables in optimization models for WDS related problems, the following equivalent representation will be used. The following formula can, e.g., be found in [BGS09]:

$$H_i - H_j = r_{i,j} \cdot Q_{i,j} \cdot |Q_{i,j}| \quad \forall (i, j) \in P, \tag{6.8}$$

where the coefficient $r_{i,j}$ aggregates the pipe "resistance" information and one has

$$r_{i,j} = \lambda_{i,j}(Q_{i,j}) \cdot \frac{8 \cdot l_{i,j}}{\pi^2 \cdot d_{i,j}^5 \cdot g}. \tag{6.9}$$

The factor $\lambda_{i,j}(Q_{i,j}), (i, j) \in P$ denotes the so-called friction factor. Its determination is dependent on he character of flow, i.e., laminar or turbulent, and furthermore the hydraulical pipe characteristic (smooth pipe versus rough pipe). With typical distribution network scenarios lying in the so-called transient area, the law of Prandtl-Colebrook provides the accurate formula for computing the friction factor. But due to its dependency on the pipe speed and hence flow rate, in this thesis, the law of Prandtl-Kármán, which provides an accurate approximation for high pipe speeds and roughness parameters of the material, will be used. The law of Pradtl-Kármán in a variant that has been resolved for the friction factor reads

$$\lambda_{i,j} = (2 \log_{10}(\frac{k_{i,j}/d_{i,j,t}}{3.71}))^{-2}, \tag{6.10}$$

where $k_{i,j}$ denotes the roughness coefficient of the pipe material in $[m]$.

Based upon the network model with its components and the associated physical laws, the simulation of network hydraulics is a key task for network analysis. In 1936 the major Hardy Cross developed a method for the consecutive computation of flows and heads in looped pipe networks which is known today as the Hardy Cross method [Cro36]. Nower contributions enlarge the applicability to a broader range of network topologies and increase the performance such as, e.g., the method of Todini and Pilati [TP88]. The hydraulic simulation of a water distribution system has become a key feature in the analysis and design of this kind of systems. But the planning of a water distribution systems can also be supported by optimization. A classification of optimization problems can, e.g., be found in the book of Boulos, Lansey and Karney [BLK06]. The latter book also provides an excellent reference for the general analysis and design methods of WDS. The examples of this thesis will basically be concerned with the design of the network in terms of pipe dimensions. A model for the related task of planning the renewal of pipes will also be presented and solved in Chapter 7. This is to be accompanied by a short literature review on pipe network design, renewal planning and optimization in WDS in Section 7.1. As the latter considerations would blow up the discussion of the ontology formalizations, it was decided to put them into an extra chapter. This section will now present a basic case study model in the next subsection. Ontology formalizations will be discussed in the subsections following thereon.

6.3.2. The Case Study Model

In Germany, most households and industrial consumers are connected to the public water distribution system. The planning of new networks is mainly due to new housing developments and therefore constitutes a rare task. In contrast to that, the renewal and extension planning of existing networks has gained a lot of attention in the last years. This is due to the fact that water usage in Germany has considerably decreased since the early 90's. In contrast to that, the distribution networks that were designed before the drop are overdimensioned due to an estimated, increasing demand. Overdimensioned networks lead to lower pipe speeds and, possibly, stagnation, which requires costly countermeasures such as pipe flooding in order to keep the high level of water quality. With the consecutive deterioration of pipes, pipe renewal planning under the aspect of minimizing pipe dimensions has gained attention. A side-effect of this dimension reduction is the saving of pipe material costs. When planning the renewal of pipes, a first proposal for decisions is typically being made by solving a pipe network design problem. This kind of problem will be introduced in what follows and is concerned with the determination of a target network's pipe dimensions with minimized pipe diameter costs.

This subsection introduces a model for the problem of pipe network design as a starting point for a case study in ontology formalizations for water distribution systems. Ontology formalizations will also be dicussed in this subsection, but the main part of the type discussion will be presented in the following subsections. The model presented in this subsection is a slightly modified variant of a Mixed Integer Nonlinear Programming (MINLP) model presented in the article [BDL^{+}06]. The case study variant of the model is given in Figures 6.19 and 6.20. The model presented here differs from the one given in the literature reference by the following points:

- The bounds for the pipe heads are given directly as head values instead of parameters that are computed out of pressure values and an elevation.
- The Darcy-Weisbach equations are used to represent the pipe headloss instead of the Hazen-Williams formula. The model of this subsection uses an unsmoothed variant, whilst the following subsection will introduce a type that represent a smoothing of the curve around zero. Though the exact formulas are different, the smoothing will be applied in a fashion similar to the paper reference.
- The Darcy-Weisbach equations will be split into two goups of constraints such as it has been introduced in Subsection 6.3.1. By this, another nonlinear restriction for computing the "resistance coefficient" is introduced. Both constraint groups together yield a nonlinear, nonconvex curve which is comparable to the Hazen-Williams formula of [BDL^{+}06]. An "Area Parameterization" for reducing the nonlinearity as it is suggested in the paper reference is not applied.
- Instead of a single source node, multiple source nodes and respective restrictions will be allowed. This makes it possible to demonstrate another specificational feature.

The reason for the modifications is to provide a consistent framework for the presentation of the ontology types in this section. In addition to that, the formulation of the model as it is given here can serve as a basis for understanding the renewal planning model of the next chapter.
Coming back to Figures 6.19 and 6.20, the AMPL model can be described as follows: A simple network model with junction and source nodes is defined in the first four declarative statements. Only standard pipes (no pumps and valves) are considered. Variables of the model are the nodal heads `H`, pipe flow rates `Q` and the the pipe diameters `d` that are used to represent the computed network design. Further variables are introduced for the resistance coefficient `r` as well as that some binaries for modeling increments of pipe diameters `X` are introduced. The objective of the model is to minimize the overall cost of pipe material. To that end, the unit cost of pipe material is modeled by a polynomial function that depends on the diameter. Multiplication by the pipe length and summation over all pipes yields the objective function. The details of the pipe diameter polynomial which in fact is a fit of discrete cost data will be further discussed in scope of the renewal planning model in Section 7.3. Coming back to the case study model from this subsection, standard demand-oriented balance equations (6.7) as well as an unsmoothed variant of the

```
#Basic water network case study model as a variant of Bragalli et al.

set Nodes;
set JunctionNodes within Nodes;
set Sources within Nodes;
set Pipes within {Nodes , Nodes};
param dnum;              #Number of available pipe diameters
param degP;              #Degree of fitted cost−function polynomial
param PI;                #Pi
param g;                 #Earth gravitation [m/s^2]
param Diameters{1..dnum};      #Set of commercially available pipe diameters in[m]
param pcoeff{0..degP};         #Coefficients of fitted cost−function polynomial
param Ck{Pipes};                       #Pipe roughness coefficient in [m]
param L{Pipes};                        #Length of a pipe in [m]
param Demand{JunctionNodes} ; #Demand at a junction node [m^3/s]
param vmax{Pipes};                      #Maximal pipe velocities   [m/s]
param dmin{Pipes};                      #Minimal pipe diameter    [m]
param dmax{Pipes};                      #Maximal pipe diameter    [m]
param QSmin{Sources};         #Minimal source inflow     [m^3/s]
param QSmax{Sources};         #Maximal source inflow     [m^3/s]
param Hmin{JunctionNodes};    #Minimal head of junction node [m]
param Hmax{JunctionNodes};    #Maximal head of junction node [m]
param HSources{Sources};      #Initial head at source node [m]

var H{Nodes} >=0;        #Head at nodes [m]
var Q{Pipes};            #Flow through pipes [m^3/s]
var r{Pipes} >=0;        #Resistance coefficient of pipes for DW
var d{Pipes}    ;        #Variable diameter of pipes [m].
var X{Pipes,0..(dnum−1)} binary; #Pipe diameter increment switching
```

Figure 6.19.: The water network case study model - Data definitions

Darcy-Weisbach headlossformula (6.8) are introduced in Figure 6.20. The computation of the resistance coefficient is modeled by a highly nonlinear curve in accordance to equation (6.9). The modeling of network hydraulics is continued by some bounds on the pipe speed. Two groups of constraints are required in order to model the unrestricted sign of the flow variables. The pipe speed bounds constitute quadratic restriction in the pipe diameter variables. The model presented here allows for multiple source nodes. Respective restrictions model the constant head at such nodes, lower and upper bounds on the inflow into the network as well as a nonnegativity condition for the flow variables that represent the outflow of sources. Coming back to the general network, junction node heads and pipe diamters are

```
#Objective
minimize Costs: sum{(i,j) in Pipes}(L[i,j] * sum{e in 0..degP}(pcoeff[e]*d[i,j]^e));

#Mass balance at junctions.
s.t. Massbalance{x in JunctionNodes}: sum{(i,x) in Pipes}(Q[i,x])
− sum{(x,j) in Pipes}(Q[x,j]) = Demand[x];

#Standard headloss, unsmoothed
HeadlossDW{(i,j) in Pipes}: H[i] − H[j] = r[i,j] * Q[i,j] * abs(Q[i,j]);

# Restriction for computation of "Resistance Coefficient". log10 has basis 10
s.t. ResistanceCoefficientPipes{(i,j) in Pipes}: r[i,j] = (8 * L[i,j]) /
(PI^2 * g * (d[i,j])^5) * (2 * log10((Ck[i,j]/ d[i,j] ) / 3.71))^(−2);

# Upper bounds for pipe speed
s.t. PipeSpeedBoundsupPos{(i,j) in Pipes}:
Q[i,j] <= vmax[i,j] * (PI/4) * d[i,j]^2;
s.t. PipeSpeedBoundsupNeg{(i,j) in Pipes}:
(−1) *Q[i,j] <= vmax[i,j] * (PI/4) * d[i,j]^2;

# Constant initial heads at sources
s.t. SourceHeadsConstant{s in Sources}: H[s] = HSources[s];
# Lower and upper bounds on the outflow of sources
s.t. SourceFlowBoundslow{s in Sources}: QSmin[s] <= sum{(s,j) in Pipes}(Q[s,j]);
s.t. SourceFlowBoundsup{s in Sources} : sum{(s,j) in Pipes}(Q[s,j]) <= QSmax[s];
#All outflows nonnegative by a set−intersection
s.t. SourceFlowNonneg{(s,j) in ({Sources,Nodes} inter Pipes)}: Q[s,j] >= 0;

# Bounds on heads at junction nodes
s.t. HeadBoundslow{n in JunctionNodes}: Hmin[n] <= H[n];
s.t. HeadBoundsup{n in JunctionNodes} : H[n] <= Hmax[n];

#Lower and upper bounds on the diameters
s.t. PipeDiameterBoundslow{(i,j) in Pipes}: dmin[i,j] <= d[i,j];
s.t. PipeDiameterBoundsup{(i,j) in Pipes}: d[i,j] <= dmax[i,j];

# Discrete set of commercially available pipe diameters by increments
s.t. PipeDiametersincremental{(i,j) in Pipes}: d[i,j] = Diameters[1]
+ sum{ mu in 2..dnum}((Diameters[mu] − Diameters[mu−1])* X[i,j,mu−1]);
s.t. Incrementalsalloneupto{(i,j) in Pipes, mu in 1..(dnum−2)}:
X[i,j,mu] >= X[i,j,mu+1];
```

Figure 6.20.: The water network case study model - Formulation definitions

bounded by further constraints. The last two groups of constraints model the discrete set of commercially available pipe diameters. As only such a restricted set of pipes is available in practice, the computed values of the continuous diameter variables should only take values from the respective discrete set. This is achieved by an incremental modeling as it was proposed in [BDL$^+$06]. The pipe diameters are summed up by increments which are being activated by respective binaries. The binaries are coupled in a monotonically decreasing order such that a consistent sum is guaranteed.

Numerical results for the case study model discussed in this subsection will not be provided within this thesis directly. But two points should enforce the practicability and relevance of the model. At first, the underlying original model from Bragalli et al. [BDL$^+$06] has been solved and investigated in the paper reference. Furthermore, the pipe renewal planning to be presented and solved in the next chapter contains (besides others) all the constraints and objective formulations from above in an extended form. Finally, the model given in this subsection has been checked for correctness of syntax and static semantics by an execution of the AMPL command `model`.

Ontology formalizations for the entities of the model will be explained in the remainder of this section. To that end, Subsection 6.3.4 further investigates the modeling of the mass balance constraints in the context of the water network domain. The formulation type `NetForms#SingleCommodityBalanceDemandOriented` has already been mentioned as the suited formulation type from the balance type hierarchies, but of further interest will be the operations in model formulation when the balance equations need to be integrated with pipe network typical constraints such as the headloss equations. The headloss equations themselves are of central importance for mathematical models in water networks. Different formulations may introduce linearizations and smoothings. To that end, a formulation type hierarchy that provides respective variants will be discussed in the next subsection. Another type formalization will be discussed right now, in the remainder of this subsection. Further ontology specifications and types for the case study model can be looked up in the digital material accompanying this thesis.

Exemplary formalizations shall be presented now for the restrictions that are concerned with the source nodes. The respective AMPL statements from Figure 6.20 are

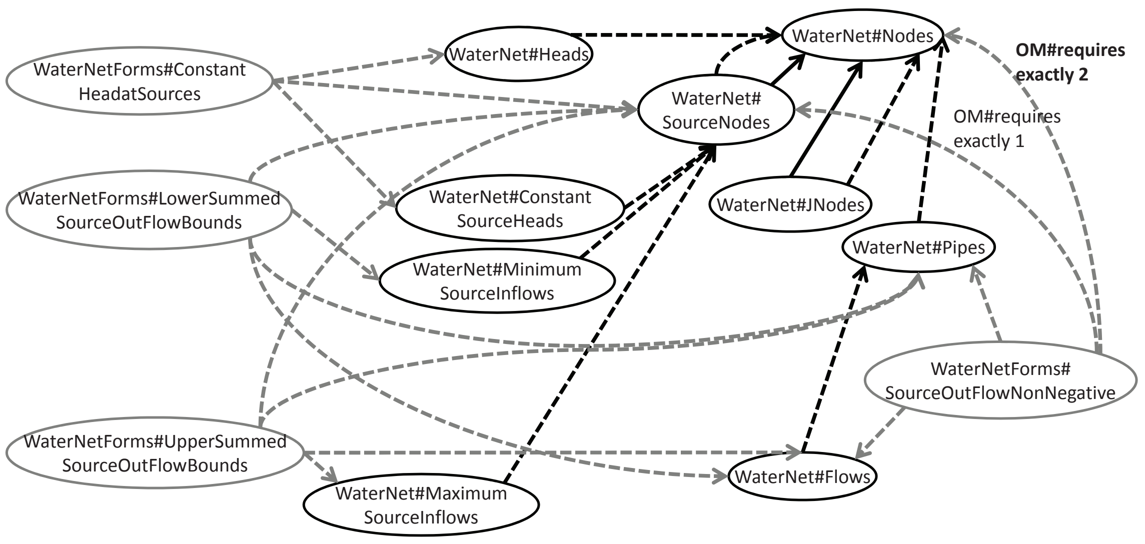

WaterNet#Nodes
WaterNet#Heads
WaterNetForms#Constant HeadatSources
OM#requires exactly 2
WaterNet# SourceNodes
OM#requires exactly 1
WaterNetForms#LowerSummed SourceOutFlowBounds
WaterNet#Constant SourceHeads
WaterNet#JNodes
WaterNet#Pipes
WaterNet#Minimum SourceInflows
WaterNetForms# SourceOutFlowNonNegative
WaterNetForms#UpperSummed SourceOutFlowBounds
WaterNet#Flows
WaterNet#Maximum SourceInflows

Figure 6.21.: Definition of waternet source restriction types

```
     s.t.  SourceHeadsConstant {s in Sources}:  H[s] = HSources[s]
                             s.t.  SourceFlowBoundslow {s in Sources}:
                              QSmin[s] <= sum{(s,j) in Pipes}(Q[s,j]);
                              s.t.  SourceFlowBoundsup {s in Sources}:
                              sum{(s,j) in Pipes}(Q[s,j]) <= QSmax[s];
    s.t.  SourceFlowNonneg {(s,j) in {Sources,Nodes} inter Pipes}:
                                                         Q[s,j] >= 0;.
```
(6.11)

The formulation types for the latter statements can be found in Figure 6.21. The `SourceHeadsConstant` are reflected by the type `WaterNetForms#ConstantHeadatSources`. As the type provides a typical water network formalization, also the mdo-concepts stem from an ontology for water networks with the prefix `WaterNet`. The `WaterNet` vocabulary provides subtypes of `Net` classes as well as novel types. Three data conceptualizations for the head variables, constant head values at sources and the source nodes themselves are required for the formulation type by class restrictions of the form `OM#requires exactly 1`. Similar specifications are made for the lower and upper source flow bounds, reflected by the formulation types `WaterNetForms#LowerSummedSourceOutFlowBounds` and `WaterNetForms#UpperSummedSourceOutFlowBounds`. Both formulation types require both `WaterNet#SourceNodes` and `WaterNet#Pipes` in order to build the outer indexing expressions as well as the indexings for the inner summations. The pipe indices for the inner summations use the outer indice `s` for referencing the source nodes and allow to select only pipes that "leave a fixed source".

Finally, the formulation type that provides a novel observation in the ontology design of formulation types is `WaterNetForms#SourceOutFlowNonNegative` with the target statement `SourceFlowNonneg` from above. In order to explicitely index over all pipes that have a source node as a starting node, a set-expression with the three sets `Sources, Nodes` and `Pipes` is build. The sets are reflected by the `WaterNet` mdo-classes `WaterNet#Nodes`, `WaterNet#SourceNodes` and `WaterNet#Pipes`. Respective axioms with the `OM#requires` property were created for the formulation type specification. But as the class `WaterNet#SourceNodes` is a subclass of `WaterNet#Nodes`, the cardinality restriction for `WaterNet#Nodes` has the cardinality value two. The necessity to model cardinality restrictions with values higher than one was mentioned in Section 6.2. Here, an example how such higher values originate is given. When required mdo-types stand in a subclass relation, the cardinality of a subtype also has to be counted in the restriction of the supertype. Without doing so, a contradiction would be introduced to any model that provides the intended, distinct individuals for both the super- and the subclass. Therefore, respective knowledge about subclasses needs to be considered when defining a new formulation type.

6.3.3. Pipe Headloss and Linearization Types

This subsection introduces reusable formulation types at the example of the Darcy-Weisbach equation(s) for pipe headloss (6.8). Different solution approaches and solvers might require different formulations and approximations of the original curve. The latter is motivated by the fact that the Darcy-Weisbach equation(s) represent a nonlinear curve that is not twice continuously differentiable in the origin. To that end, different approaches for smoothing and linearizing the constraint formulation will be discussed and encapsulated into reusable formulation types within this subsection. The types will, as in the case of network balance equations, be organized in a hierarchy, allowing for explicitely formalized choices in the model formulation. The formulation types themselves encapsulate medium sized blocks of AML definitions such that a reusage of these formulations should really save work. Finally, the perspective for automatization in the model formulation will be discussed. Furthermore, it should be emphasized that the here proposed type hierarchy of smoothing and linearization types can be seen as prototypical for other technical formulas. Whenever certain nonlinear curves are to be used as constraints in an optimization model, the provision of smoothed and linearized variants might be appropriate in order to suitedly support the process of model formulation.

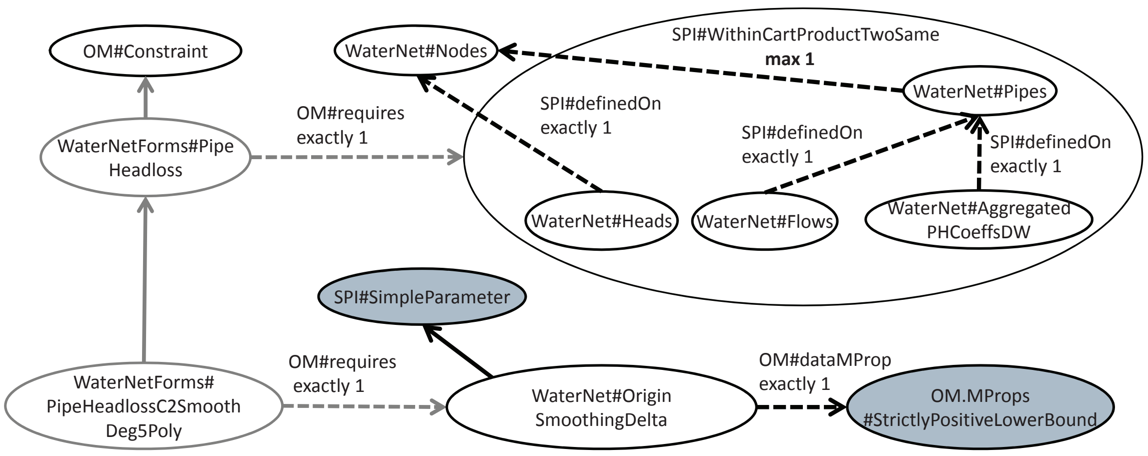


Figure 6.22.: Definitions of smoothed and standard headloss types

Figure 6.22 shows formulation type definitions for the standard pipe headloss formulation as well as a smoothed variant `WaterNetForms#PipeHeadlossC2SmoothDeg5Poly`. The standard pipe headloss constraint in the AMPL formulation of the case-study model (Figure 6.20) reads

```
    HeadlossDW{(i,j) in Pipes}:  H[i]-H[j]=r[i,j]*Q[i,j]*abs(Q[i,j]);.
```

The respective formulation type `WaterNetForms#PipeHeadloss` requires exactly one of `WaterNet#Pipes, WaterNet#Heads, WaterNet#Flows` and `WaterNet#AggregatedPHCoeffsDW`. The latter class represents the multiplicative "resistance coefficient" `r`. The second formulation type in Figure 6.22 models a variant of the standard pipe headloss formulation. As the standard formulation is not twice continuously differentiable around the origin, the type `WaterNetForms#PipeHeadlossC2SmoothDeg5Poly` introduces a smoothed variant of the curve. Details about the smoothing will be given in the context of the renewal planning model in Section 7.3. For the moment, it should suffice to know that the resulting curve differs from the standard formulation in a range `deltaHeadloss` of flow values around the origin. In this interval, the curve is approximated by a smooth polynomial of degree five. A target AMPL statement in the context of the case-study model then reads

```
                         param deltaHeadloss > 0;
                      HeadlossDW{(i,j) in Pipes}:
     H[i] - H[j] = if(abs(Q[i,j]) > deltaHeadloss)
                then r[i,j] * Q[i,j] * abs(Q[i,j])
   else (r[i,j] * (0.375 * deltaHeadloss * Q[i,j]
           + (3 / (4 * deltaHeadloss)) * Q[i,j]^3
      - (1 / (8 * deltaHeadloss^3)) * Q[i,j]^5));.
```

(6.12)

The positive parameter `deltaHeadloss` is introduced at first. The formulation uses logic expressions by the case distinction `if then else`. The logics behing the piecewise curve decalaration will be resolved into a MIP formulation by AMPL using standard MIP modeling techniques. The formulation type `WaterNetForms#PipeHeadlossC2SmoothDeg5Poly` that delivers the target statement can be defined as a subclass of `WaterNetForms#PipeHeadloss`. By this, all type requirements are inherited and only the additional requirement of exactly one `WaterNet#SmoothingDelta` has to be added to the type definition. The respective mdo-type models the positive parameter `deltaHeadloss`. An additional mathematical property definition is prescribed for any individual of the mdo-type `WaterNet#SmoothingDelta` in order to enforce the positive value restriction.

Besides smoothing, linear approximations of the pipe headloss curve are of interest for applying standard MIP techniques to the solution of water distribution related optimization models. The L01 linearization of separable, nonlinear functions provides an established MIP modeling technique for linearizing and representing nonlinear constraints in a MIP model. Padberg [Pad00] reviews the technique and gives a proof that it yields better polyhedral formulations than another established linearization and representation technique usually referred to as SOS2. A standard L01 linearization, applied to the pipe headloss curve in the flow variables `Q`, can be found in Figure 6.23. Before explaining the technique, it is important to mention that linearization would have to be applied to all nonlinear formulations of a model when aiming to solve it by mixed integer programming solvers. In the case study model, e.g., the `ResistanceCoefficientPipes` restriction that gives the curve for the "resistance coefficient" `r`, as well as the quadratic restrictions for the pipe speed bounds and the polynomial objective function, would also need to be linearized. Furthermore, the variable `r` occurs in a product term in the pipe headloss curve of the flow variables `Q`. To that end, the case study model is highly nonlinear such that the here represented techniques are only of exemplary character. Nonetheless, by making diameter decisions `d` constant and switching to another model, e.g., for planning the operation of pumps, or the technical planning of further assets such as tanks, the respective techniques could be applied in a practical fashion to obtain a linear model.

Let us now continue by introducing the linear modeling of the pipe headloss constraints in Figure 6.23: The AMPL modeling introduces three parameter declarations. A nonnegative, integer parameter `kL01` for modeling the step-number of the L01 linearization is introduced at first. Then, two parameters `a` and `b` are defined to model grid points of the curve `Q[i,j] * abs(Q[i,j])`. The parameter `r[i,j]` is not part of the curve linearization and will be multiplied to the respective terms in order to yield a representation of the pipe headloss that is linear whenever `r[i,j]` is constant. The grid points will be used to build interval lengths and (approximative) gradients in what follows. The resulting curve will be stepwise linear on the intervals. In order to represent the resulting curve in a mixed zero-one programming fashion, two variables `zL01` and `yL01` are introduced. Whilst `zL01` models parts of the interval lengths, the binaries `yL01` are required in two restrictions `ziFullSegmentLengthTill` and `ziSegmentLengthBound` to model the accounting of lengths and value gains/decreases of a linear segment in a way such that all segments are fully accounted in a consecutive fashion and only the last such segment constributes its values in a partial fashion. This is reflected by the two constraints `FlowRepLinL01` and `HeadLossRepLinL01` that represent the respective values of flow and head gain/loss by a sum over the segment contributions. Together with the initial `Boundz01` on the first interval length variable `zL01[i,j,1]`, the modeling introduces three parameter-, two variable- and five constraint-declarations as a whole. The respective modeling work for writing the statements in an AML model can be seen as quite costly and error-prone in contrast to the few ontological specifications when reusing a formulation type.

Figure 6.24 provides and overview on the formulation type definition for the linearization type `WaterNetForms#PipeHeadlossLinearizedL01NonUniform` that yields statements such as explained above. The mdo-requirements of the type are grouped into two parts in the figure, as there are the standard water network requierements of exactly one of `WaterNet#Pipes, WaterNet#Heads, WaterNet#Flows` and `WaterNet#AggregatedPHCoeffsDW`, which are the same as for the standard formulation type `WaterNetForms#PipeHeadloss`, and then further requirements for data of the linearization. The formulation type `WaterNetForms#PipeHeadlossLinearizedL01NonUniform` can be defined as a subclass of the type `WaterNetForms#PipeHeadloss` and hence, the respective waternetwork mdo-requirements can be inherited from that type. When changing the pipe headloss formulation in a model to the subtype, the respective definitions of individuals and their relations can be taken over as they are. Furthermore, another L01-linearization type, which is a further father of `WaterNetForms#PipeHeadlossLinearizedL01NonUniform`, will be defined below,

```
param kL01 > 0, integer;
param a{0..kL01};
param b{0..kL01};
var zL01{Pipes, 1..kL01} >= 0;
var yL01{Pipes, 1..(kL01 − 1)} binary;
s.t. FlowRepLinL01{(i,j) in Pipes}: Q[i,j] = a[0] + sum{l in 1..kL01}(zL01[i,j,l]);
s.t. HeadLossRepLinL01{(i,j) in Pipes}: r[i,j] ∗ ( b[0] + sum{l in 1..kL01}( ((b[l] − b
    [l−1]) / (a[l] − a[l−1])) ∗ zL01[i,j,l] )) = H[i] − H[j];
s.t. Boundz01{(i,j) in Pipes}: zL01[i,j,1] <= a[1] − a[0];
s.t. ziFullSegmentLengthTill{(i,j) in Pipes, l in 1..(kL01−1)}:
zL01[i,j,l] >= (a[l] − a[l−1]) ∗ yL01[i,j,l];
s.t. ziSegmentLengthBound{(i,j) in Pipes, l in 1..(kL01−1)}:
zL01[i,j,l+1] <= (a[l+1] − a[l]) ∗ yL01[i,j,l];
```

Figure 6.23.: AMPL statements for a non uniform L01 linearization of the pipe headloss in a MIP

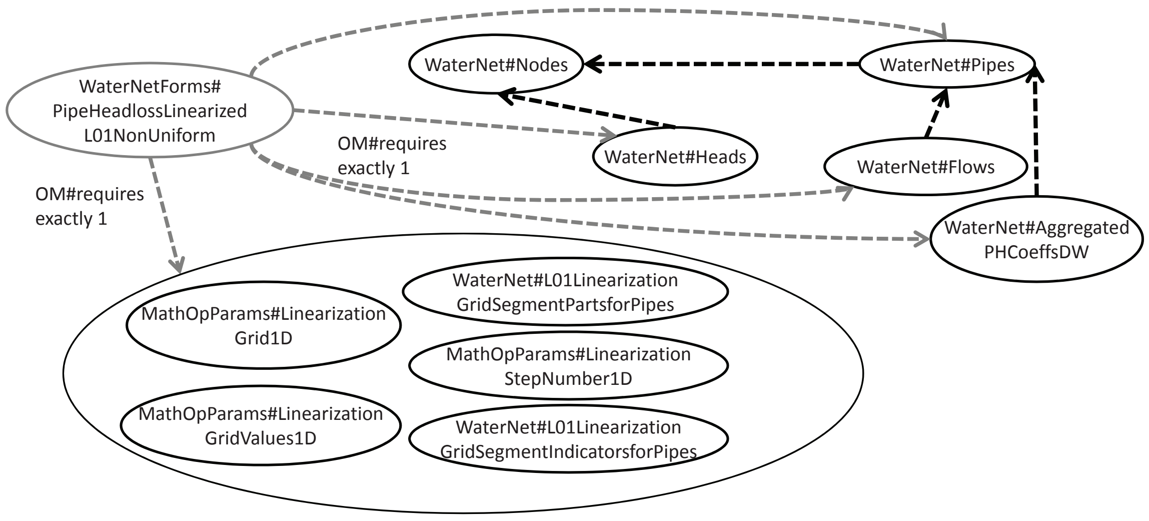


Figure 6.24.: Type definition for the non uniform L01 linearization of the pipe headloss

such that a whole hierarchy of formulation types originates.

Let us now continue with the consideration of the left-open mdo-requirements for the type `WaterNetForms#PipeHeadlossLinearizedL01NonUniform`: An external vocabulary `MathOpParams` yields three classes `MathOpParams#LinearizationStepNumber1D`, `MathOpParams#LinearizationGrid1D` and `MathOpParams#LinearizationGridValues1D` of which exactly one individual is required each. The respective mdo-types represent the three parameter definitions of kL01,a and b in the examplary AMPL statements. To that end, both MathOpParams#LinearizationGrid1D and MathOpParams#LinearizationGridValues1D are defined on exaclty one MathOpParams#LinearizationStepNumber1D. This fact can be looked up, together with the other mdo-definitions, in Figure 6.25. Two further required concepts are each exactly one of WaterNet#L01LinearizationGridSegmentPartsforPipes and WaterNet#LinearizationGridSegmentIndicatorsforPipes. In that order, they represent the variables zL01 and yL01 that are required for the modeling. Again, both mdo-types are defined on exactly one MathOpParams#LinearizationStepNumber1D and exactly one WaterNet#Pipe.

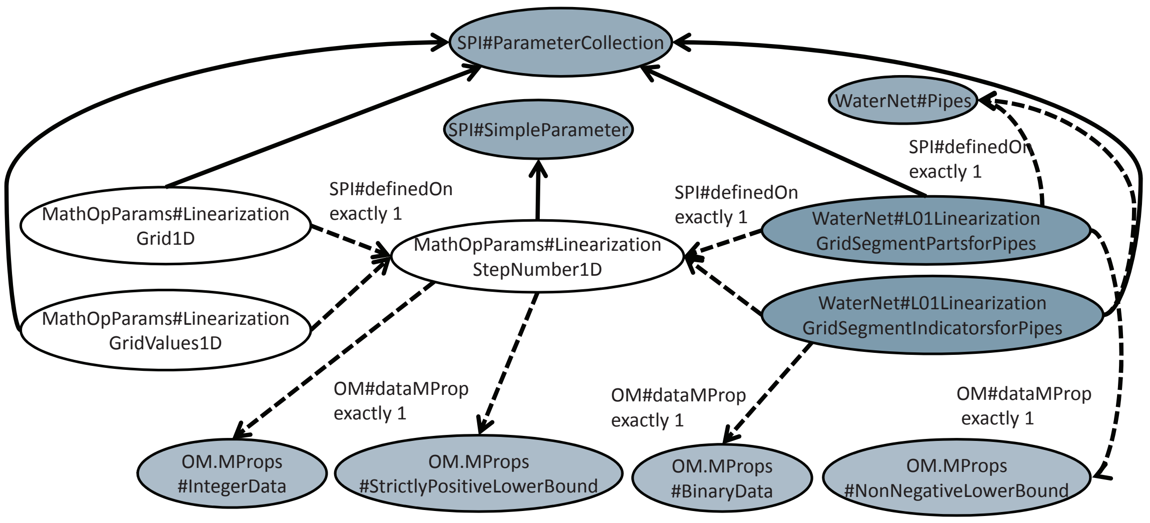

SPI#ParameterCollection
WaterNet#Pipes
SPI#SimpleParameter
SPI#definedOn exactly 1
SPI#definedOn exactly 1
SPI#definedOn exactly 1
MathOpParams#Linearization Grid1D
MathOpParams#Linearization StepNumber1D
WaterNet#L01Linearization GridSegmentPartsforPipes
MathOpParams#Linearization GridValues1D
WaterNet#L01Linearization GridSegmentIndicatorsforPipes
OM#dataMProp exactly 1
OM#dataMProp exactly 1
OM#dataMProp exactly 1
OM.MProps #IntegerData
OM.MProps #StrictlyPositiveLowerBound
OM.MProps #BinaryData
OM.MProps #NonNegativeLowerBound

Figure 6.25.: MDO definitions for the non uniform L01 pipe headloss linearization type

The mdo definitions in Figure 6.25 furthermore contain required mathematical definitions on class level, e.g., for the integrality and positive value requirement of the MathOpParams#LinearizationStepNumber1D individual. Before concluding the representation of the first L01 linearization type, a remark should be made on the grid values: As the data values for the parameters a and b represent samples from the curve $Q * |Q|$ in the flow variable Q, the respective generation of the values could be done in an automated fashion by a further service. Before computing points on the curve, the value kL01+1 of grid points as well as the lower and upper bounds for the linearization need to be determined. The rough idea would be to allow an extra module to generate the values for a,b and kL01 depending on estimated flow value bounds and accuracy requirements of the approximation. This step could be performed a priori the instantiation of the model with data. This remark should suffice for the scope of this thesis, let us now continue with another L01 linearization type that, in contrast to the latter remark, automates the computation of the grid

```
param kL01 > 0, integer;
param aStartEnd{0..1};
param stepWidth = (aStartEnd[1] − aStartEnd[0]) / kL01;
param aComp{l in 0..kL01} = if l = 0 then aStartEnd[0] else aComp[l−1] + stepWidth;
param bComp{l in 0..kL01} = if l = 0 then aStartEnd[0] ∗ abs(aStartEnd[0])
 else aComp[l] ∗ abs(aComp[l]);
var zL01{Pipes, 1..kL01} >= 0;
var yL01{Pipes, 1..(kL01 − 1)} binary;
s.t. FlowRepLinL01{(i,j) in Pipes}: Q[i,j] = aComp[0] + sum{l in 1..kL01}(zL01[i,j,l]);
s.t. HeadLossRepLinL01{(i,j) in Pipes}: r[i,j] ∗ ( bComp[0] + sum{l in 1..kL01}( ((
    bComp[l] − bComp[l−1]) / (aComp[l] − aComp[l−1])) ∗ zL01[i,j,l] )) = H[i] − H[j];
s.t. Boundz01{(i,j) in Pipes}: zL01[i,j,1] <= aComp[1] − aStartEnd[0];
s.t. ziFullSegmentLengthTill{(i,j) in Pipes, l in 1..(kL01−1)}:
zL01[i,j,l] >= (aComp[l] − aComp[l−1]) ∗ yL01[i,j,l];
s.t. ziSegmentLengthBound{(i,j) in Pipes, l in 1..(kL01−1)}:
zL01[i,j,l+1] <= (aComp[l+1] − aComp[l]) ∗ yL01[i,j,l];
```

Figure 6.26.: AMPL statements for a uniform L01 linearization of the pipe headloss

values by respective AMPL statements.

The second L01 formulation type is exemplified by target AMPL statements in Figure 6.26. In contrast to the definitions of Figure 6.23, the grid value parameters `a` and `b` are replaced by *computed parameters* `aComp, bComp` and an additional parameter collection `aStartEnd`. The parameter `stepWidth` represents a further computed paremeter. The values of computed parameters are determined out of other parameter values in AML environments such as AMPL when a model is instantiated with data. To that end, the only data-values that are to be provided externally for the declarations in the figure are for the parameters `kL01` and `aStartEnd`. The values represent the number of grid points (minus one, see, e.g., the declaration of `aComp`) as well as the start (`aStartEnd[0]`) and end (`aStartEnd[1]`) of the interval on which the curve will be approximated. The respective linearization will be based upon a uniform grid. This should also become clear from the introduction of a uniform `stepWidth` and the computation of the grid values in the parameter definitions of `aComp` and `bComp`. Besides the usage of the computed parameter values `aComp, bComp` and the respective interval bounds `aStartEnd` in the constraint and variable declarations, no further differences to the AMPL declarations for the non uniform L01 linearization exist.

A look into the formulation type definitions for `WaterNetForms#PipeHeadlossLinearizedL01Uniform` in Figure 6.27 reveals that the uniform L01 linearization can be defined as a supertype of the non uniform one. In difference to the sole consideration of the non uniform type in Figure 6.24, the

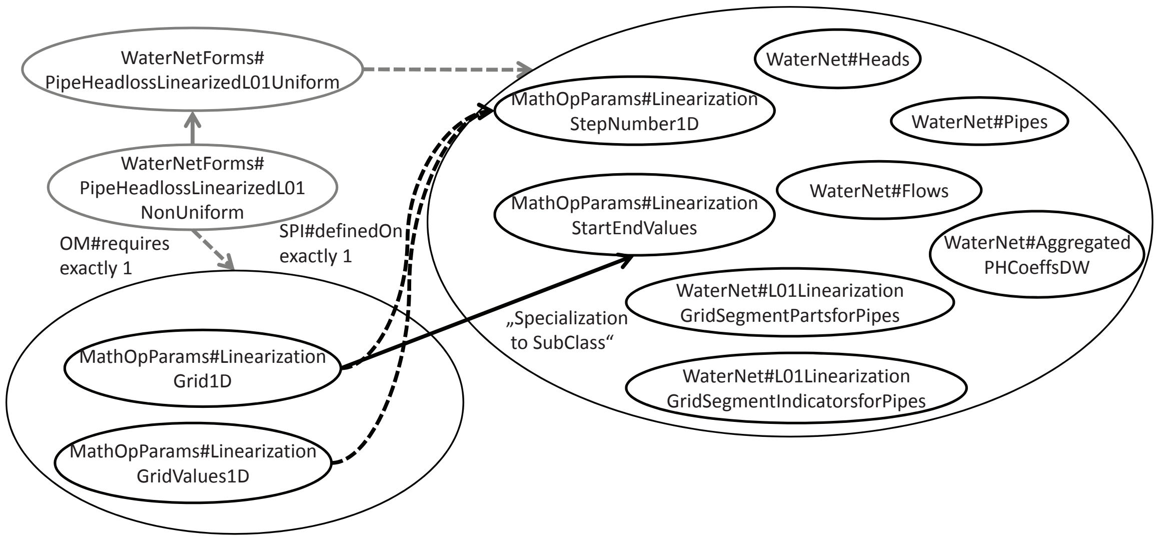


Figure 6.27.: The uniform L01 pipe headloss linearization type as a parent of the non uniform one

waternet mdo-requirements for pipes, heads, flows and resistance coefficients are now shifted upwards to the father type.
`WaterNetForms#PipeHeadlossLinearizedL01Uniform` in similarity to its subtype requires exactly one of `WaterNet#L01LinearizationGridSegmentPartsforPipes` and `WaterNet#GridSegmentIndicatorsforPipes` for the variable declarations of `zL01` and `yL01`. To that end, also these type requirements are shifted upwards to the supertype. A difference occurs in the representation of the discretization parameters for the L01 linearization. The requirement of the `MathOpParams#LinearizationStepNumber1D` can also be moved to the supertype but a requirement of exacly one individual of the novel type `MathOpParams#LinearizationStartEndValues` occurs for the supertype. The respective class restriction is then refined by the non uniform subtype to the range `MathOpParams#LinearizationGrid1D` in order to yield the usual requirement for the grid segment values. The `MathOpParams#LinearizationGridValues1D` are only required by the subtype as the uniform grid supertype treats grid points as computed parameters. This also explains the aforementioned refinement of `MathOpParams#LinearizationStartEndValues` to `MathOpParams#LinearizationGrid1D` when changing from the super to the subtype: The start and end values of the interval for the L01 linearization need to be provided anyhow, but the respective whole grid is only required for the non uniform subtype. The supertype treats the grid implicitely as a computed parameter.

To complete the consideration of the formulation type `WaterNetForms#PipeHeadlossLinearizedL01Uniform`, it should be mentioned that the latter type provides a more simple formulation type than its non uniform subtype with respect to the complexity of required ontology specifications and instance data provision. Fewer mdo-individual specifications and data values are required to be provided when instantiating the type. Nonetheless, if instance data is provided in an automated fashion, as it was mentioned above for the non uniform type, the latter aspect becomes less important and a usage of `WaterNetForms#PipeHeadlossLinearizedL01NonUniform` might be indicated.

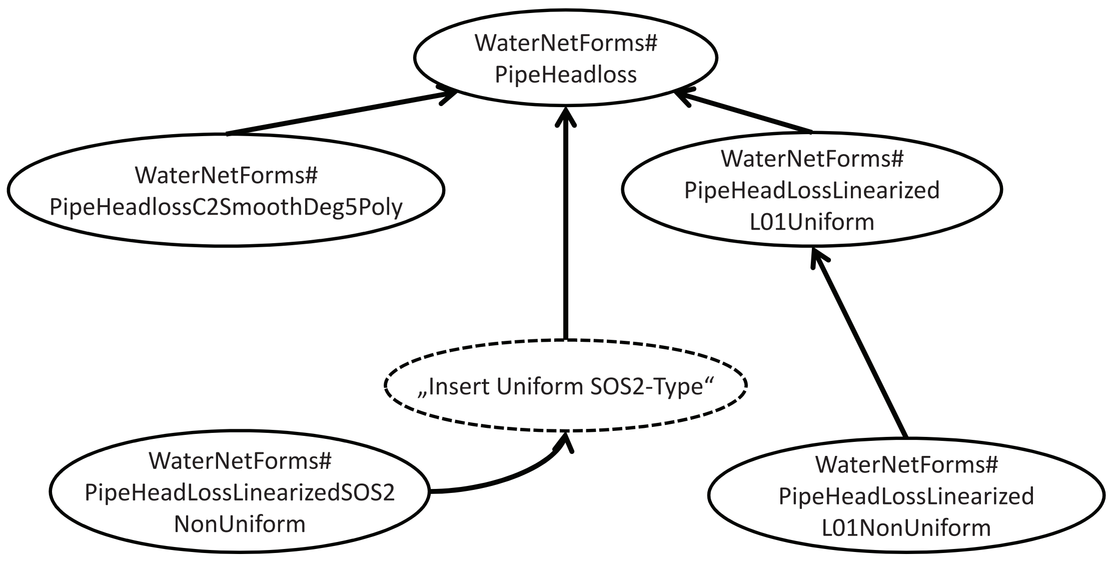


Figure 6.28.: Hierarchy of pipe headloss smoothing and linearization types

The preceding considerations of this subsection introduced four formulation types for the pipe headloss constraints that varied in their differentiability and linearity properties. The L01 linearization types were further distinguished by uniform and non uniform grid approaches, where the uniform grid was argued to be specificationally simpler. Figure 6.28 concludes the consideration of pipe headloss types by structuring them into a subsumption hierarchy. Type requirements are not visualized for reasons of simplicity, but should be clear from the preceeding considerations. The standard pipe headloss type `NetForms#PipeHeadloss` defines the class restrictions for pipes, heads, flows and resistance coefficients and the subtypes inherit and possibly adapt or extend the class restrictions. A new formulation type `WaterNetForms#PipeHeadlossLinearizedSOS2NonUniform` as well as a placeholder for a uniform grid variant of that type can also be found in the hierarchy. The SOS2 linearization provides an alternative to the L01 linearization techniques by the introduction of so-called SOS2 variables. SOS2 linearizations can be compared to the L01 pendants from a formulation type perspective. But the linearization with SOS2 variables provides a polyhedral formulation that is not as good as the L01 formulation (,see, again Padberg [Pad00]). Nonetheless, many MIP solvers support SOS2 variable declarations and an efficient treatment of these in their algorithms. To that end, SOS2 formulation types, though not further discussed here, might be of certain use to some model formulation approaches.

Let us now conclude this section by discussing the automation capabilities of the pipe headloss formulation types and their hierarchy according to the use case "Formalized choices in modeling" from Section 4.1. The types from this subsection reflect the pipe headloss formulation example considered in the use case. The goal of providing formalized modeling knowledge in the form of choices and type interfaces is achieved by the type hierarchy. Furthermore, the provided formulation types might help saving costs and reduce errors in model formulation as they provide medium-sized building blocks of reusable AML statements. Especially, the linearization types were discussed under the latter two points. What remains to discuss are the capabilities for automation in the adaption of the ontology specification when changing the type of a pipe headloss formulation, as well as the consequences for the automated generation of a runnable system. The first subtype example `WaterNetForms#PipeHeadlossC2SmoothDeg5Poly` from this subsection provided a smoothing of the pipe headloss equation. Consider the fixed case study model with the standard type `WaterNetForms#PipeHeadloss` in an ontological representation. When changing the type of the pipe headloss equation individual to the subclass, all existing axioms stay valid according to the class restrictions. The only missing specification is a new relation to an individual of `WaterNet#OriginSmoothingDelta` as well as a strictly positive lower bound property for the latter. A respective individual for the smoothing around the origin could be added and typed automatically in the case where there is no such individual in the model. In addition to that, the mathematical property specification can also be added automatically, leading to a consistent specification for the new smoothed pipe headloss type. The problem that could occur with this has already been described for the min-cost-flow models: If one or more individuals for an origin smoothing parameter were added to the model before, the modelers intention which individual is to relate can only be guessed. Creating and relating a new individual might not always be correct as well as that relating a single existing individual appears logical but is not provably correct. In case of two or more individuals, matching could be applied, but nonetheless, automated model formulation decisions only provide guesses.

With a consistent constraint declaration, the methods of Section 5.6 can be applied to analyze the model in terms of mathematical properties. If the pipe headloss is the only not twice constinuously differentiable function, the respective properties of the model and the postconditions of the model instantiation service will change after a type adaption. This would then allow more solvers to be included in the composition procedure, especially those which have the C2-property of the input as a precondition. Before the system is brought to execution, the value of the

`WaterNet#OriginSmoothingDelta` data could simply be set to a standard value such as 10^{-6}. Alternatively, a more detailed configuration that goes beyond the scope of this thesis could be performed by an exra module before the instantiation of the model with data.

In a similar fashion to the smoothing, a type change from `WaterNetForms#PipeHeadloss` to a linerization type such as `WaterNetForms#PipeHeadlossLinearizedL01Uniform` can be considered. In the end, the resulting model could have become linear which would allow more solvers, especially those with a linearity precondition, to be included in the template composition step. A more complex task would be the automation of the specificational adaptions. This time, two new parameter individuals would need to be introduced, or related automatically, without knowing the modelers exact intention. The generation of data values for the respective number of steps and the interval bounds would again be something to be performed by an extra module a priori the instantiation of the model with data.

6.3.4. Model Formulation Considerations

Model formulation was introduced in Section 3.2 as a mechanism for model creation in the model management lifecycle. When formulating models in the ontology representation, different capabilities could be exploited to support model formulation. These capabilities are focussed around the knowledge that is contained in the ontology representation and the reusable model entity types. Type knowledge in form of class restrictions can, e.g., be exploited to conclude required specifications that are needed to keep a model consistent. More specific, ontological consistency in this context would include correct and complete specifications and relations of model entities.

Futhermore, type knowledge might be exploited in order to recommend certain operations, e.g., changing a type or adding a new individual of a certain type. If a system for model formulation is configured to allow for guesses, performing steps in model formulation can be automated as it was discussed for the preceding examples. Different such scenarios were discussed in the preceding sections. This subsection shall conclude these considerations and give a further example of a use case in model formulation, in which a model with a single constraint declaration is extended by another constraint declaration from a subdomain. In the end of this subsection, the use case of adding a new formulation entity to an ontology model will be investigated from a process perspective.

An important example for support in the model formulation was given in the context of the multicommodity min-cost-flow model from Subsection 6.1.2 where, besides others, the idea to recommend the change of a formulation type and further data specifications, based upon a matching of formulation type class restrictions, the actual model specifications, and some change in a set conceptualization for commodities, was discussed.

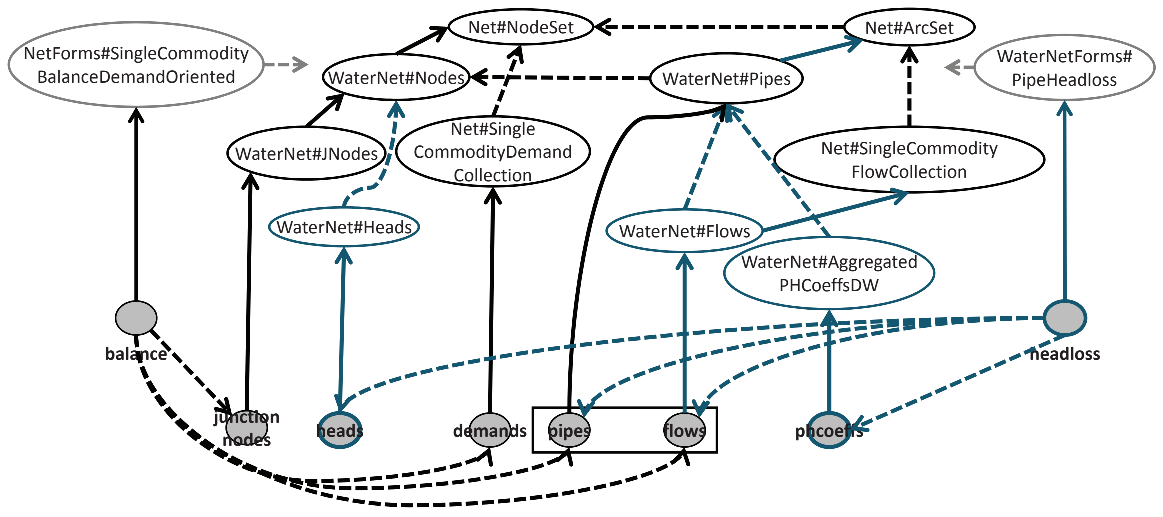
When a formulation type is changed or a new formulation entity is added to a model, the specification of the entity as well as those of all related ones need to be adapted according to ontology class restrictions. The latter scenario was discussed for the multicommodity flow model in Subsection 6.1.2, as well as for its extension in Subsection 6.1.4. In the latter model, especially a type change to a subclass was performed and the type change required the additional relation of a required parameter individual. Further examples were also given in the last subsection for the pipe headloss smoothing and linearization types where, besides the specificational consistency according to class restrictions, the consequences for the model properties and model composition were discussed.

If a conclusion is to be drawn from the examples seen so far, one can say that the retrieval of ontology model consistency in terms of class restrictions is an important task in all model formulation operations. A respective process will be discussed at the example of adding a new formulation entity to a model further below. For the moment it is just repeated that there is a tradeoff between the automatization of the retrieval of such a specificational consistency and the intention of a modeler that has to be interrogated and entered in a manual fashion. I.e., getting a consistent model according to class restrictions might be possible, but the resulting model might not be that what the modeler wanted. To that end, respective tools for model formulation in the ontology representation should provide means to configure the degree of automation allowing in principle, to interrogate the modeler in conflict situations. Furthermore, it should be stated that recommending formulations and providing consistency is not everything in model formulation. There are always steps remaining that need to be performed by a modeler such as, e.g., the declaration of data entities as variables or the specification of mathematical properties for the data entities such as, e.g, integrality requirements.

Certain operations for the model formulation in the ontology representation have cristalized out of the preceding examples. E.g., it was illustrated how new model entities can be added to a model. This requires a type definition as well as a creation of further specifications according to the class restrictions of the entity. As already mentioned, the respective specifications can make a recursive treatment of the specifications of related mdo-individuals necessary. Besides the addition of new elements, old elements might be adapted in terms of type changes. This then typically requires the consistency-retrieval of the ontology model specifications according to class restrictions. Together, *restoring specificational consistency in an ontology represented model* can be identified as an important process.

At every time in the model formulation, a modeler might extend specifications of model entities that are not covered by class restrictions, e.g., declaring a variable or specifying a mathematical property. When elements get deleted, required adaptions in the ontology model get extended by a different orientation. I.e., the affected element does not only imply checking the meaningfulness of objects in the axioms where it occurs as a subject, but vice-versa do all individuals have to be checked that have the element as an object. This thesis does not provide a complete characterization of model formulation operations. Rather will the case of adding a new formulation entity and restoring specificational consistency be treated below. At first, an additional example will be given.

Model Formulation Example: Balance extended by Headloss



Net#NodeSet
Net#ArcSet
NetForms#SingleCommodity BalanceDemandOriented
WaterNet#Nodes
WaterNet#Pipes
WaterNetForms# PipeHeadloss
WaterNet#JNodes
Net#Single CommodityDemand Collection
Net#SingleCommodity FlowCollection
WaterNet#Heads
WaterNet#Flows
WaterNet#Aggregated PHCoeffsDW
balance
headloss
junction nodes
heads
demands
pipes
flows
phcoeffs

Figure 6.29.: Extending a model with a balance constraint by a headloss constraint

In this example, a modeler with the abstract goal to generate an ontology represented version of the case study pipe network design model from Subsection 6.3.2 starts his modeling work with some initial specifications. These specifications simply just cover the network model in mind that consists of network nodes with the two subsets of junction- and source-nodes, as well as the network pipes. As the model is to be generated for water networks, the modeler generates individuals for all the latter entities together with water network typical types from the vocabulary `WaterNet` and their further specifications such as subset relations. Besides the definition of the network set elements, no further specifications are added to the model. I.e., no parameters or variables are defined at first. The direct next step now is the introduction and specification of a network balance constraint model entity. Therefore, before coming to the declaration of the headloss equations that were outlined for this example, the specifications for the network balance constraints will be considered.

As just said, the next step is the introduction of the first constraint group coming into mind: The network balance constrains. As the modeler knows that he is dealing with water networks, he considers to add the constraint group in a form that uses the typical conceptualization of positively counted water network demand as well as a single flow good of water. Before introducing the respective parameter- and variable- collections, he first of all generates an individual of the type `NetForms#SingleCommodityBalanceDemandOriented` within the ontology model. By now, he would have the possibility to create and relate the required data individuals on his own. But he decides differently and uses functionality from an extra ontology model formulation module that allows him to partially automatically create and/or relate required data individuals according to the type definition. As the type definition for `NetForms#SingleCommodityBalanceDemandOriented` has two cardinality class restrictions for exactly one of `Net#ArcSet` and `Net#NodeSet`, the respective individuals are related automatically by respectively generated axioms. Since the latter mdo-individuals were added to the model in a manual fashion before, their specificational correctness in terms of the class restrictions is to be checked again. The individuals were already typed to the water network types `WaterNet#Pipes` and `WaterNet#JNodes` which are subclasses of the required network domain superclasses and hence introduce no contradictions. Therefore, the partially automated specification procedure is continued with the generation, typing and further recursive relation of the required individuals for `Net#SingleCommodityFlowCollection` and `Net#SingleCommodityDemandCollection`.

Both individuals are created and related as objects of the axioms for the `balance` individual, but as the types of the respective `NetForms#SingleCommodityBalanceDemandOriented` `OM#requires` class restrictions are not water network specific, they are first of all directly typed to the `Net`-domain types mentioned above. In case of the individual `flows`, a respective adaption of the type to a subdomain type for water networks will be required when extending the model by other constraint formulations. But before the addition of a new formulation entity will be discussed, the consideration of the automated balance individual specifications should be concluded. As already mentioned, two individuals, `flows` and `demands` for the flow variables of water and the water demands at junction nodes, need to be created the new and their specifications will have to be created in a recursive fashion. The mdo-types are given by respective `Net`-domain types and are at first not specialized types from the subdomain `WaterNet`. `SPI#definedOn` axioms for pipes and junction nodes will have to be added in an automated fashion. In case of the pipes for the flow-individuals, there is exactly one `Net#ArcSet` individual in the model that has a type definition for the subclass `WaterNet#Pipes`. The respective axiom should be generated in an automated fasion. In the case of the demand, a decision has to be made, which of the three `Net#NodeSet` individuals for nodes, junction nodes and sources should be chosen. This can not be resolved in an automated fashion with any meaningful guess and so the modeler would have to be interrogated. He should then answer the dialogue by choosing the individual for junction nodes.

After the declaration of the mass balance equations, the next step could be the aforementioned introduction of the pipe headloss constraints. The situation is visualized in Figure 6.29. The modeler might start the process by creating a new indvidual `headloss` of the standard headloss formulation type `WaterNet#PipeHeadloss` in the model ontology. Then again, model formulation functionality could be used to extend the specification of the pipe headloss individual and the affected other data individuals in an automated fashion. As `WaterNet#PipeHeadloss` requires four individuals from the `WaterNet`-domain, the following should happen: At first, the required `WaterNet#Pipes`-individual `pipes` could be related in an automated fashion. This step should work without complications as `pipes` is the only `WaterNet#Pipes` individual in the model. Then, an individual of `WaterNet#Flows` is required. As there is no such individual in the model, an extended search might conclude that an individual `flows` of the superclass `Net#SingleCommodityFlowCollection` is contained in the model. The automated procedure for model formulation hence might decide to adapt the type of `flows` to the subclass and relate it by an `OM#requires`-axiom in an automated fashion. Now, two further axioms are required for the single individuals of `WaterNet#Heads` and `WaterNet#AggregatedPHCoeffsDW`. In case of the head individual, such an individual could again be created and typed in an automated fashion, but for its `SPI#definedOn` axiom of a `WaterNet#Nodes`, the choice between the indviduals for all nodes, junction nodes and sources would need to be resolved by the user again. Finally, an individual `phcoeffs` for the resistance coefficient type would have to be created and specified further. As no such individual exists in the model and the only `SPI#definedOn` class restriction can be resolved to the single existing individual `pipes` in the model, an automated specification should work out well in this case.

With the relation of the four required mdo-individuals as well as the recursive and mostly automated adaption/generation of their specifications, the definition of the `headloss` individual is finished as far as a model formulation module can consider it. What remains open and needs to be performed by the modeler are further specifications such as the definition of the individuals `heads, flows` and `phcoeffs` as variables of the model. Again, this is something that lies in the intention of the modeler and cannot be guessed or concluded by a model formulation component. In addition to that, mathematical properties such as the non-negativity of the head and resistance coefficient variables will have to be entered manually. In case of the resistance coefficient of type `WaterNet#AggregatedPHCoeffsDW` one could think about adding an additional data restriction to the mdo-type that enforces a respective specification. Then again, based upon the class knowledge, a model formulation component could add such a specification in an automated fashion. But nonetheless, for the head-variables such a specification might not be correct in every model context.

In the description of the model formulation example, some further aspects were neglected so far. E.g., the definition of the `OM#EName` axioms and their values was left out of the consideration. Therefore, it is mentioned that also this step should allow for automatization by a model formulation component. Furthermore, the example with its setting also provided only one realization of a model formulation process under many possible ones. In the example above, the modeler could, e.g., have decided to complete the specifications of the data model before the definition of the first formulation entity. This would have had consequences on the automated specifications when adding the formulation entities. E.g., the `SPI#definedOn`-properties of the related mdo-individuals would have already existed when defining the model formulation entities. In a different approach, the model might be defined witout any pre-existing data model. In this case, all the types for a mass balance constraint might, e.g., first of all be created and typed automatically for the domain `Net`, until directly after the automated specifications, the modeler decides to adapt them to water network types manually. This would, in case of the balance equations, lead to a scenario similar to the first ontology specification example of this thesis in Subsection 5.3.1, Figure 5.5.

Model Consistency Retrieval Process Considerations:
This subsection shall now be concluded by the characterization of a process for restoring consistency of ontology model specifications according to class restrictions. In order to keep things simple, the consideration will be restricted to the case where a new formulation entity is added to the model and by this causes a need for adapting specifications. When a new formulation entity is created, the formulation type should be defined by the modeler in a first step. Then, according to the cardinality class restrictions of the chosen type, different `OM#requires` specifications need to be created. These specifications can be entered manually or created in an automated fashion, as different examples of the preceding sections discussed it, but the central point is the order of processing the `OM#requires` specifications in combination with the further mdo-specifications and adaptions they imply. As multiple `OM#requires` specifications originate from the choice of a formulation type, the required adaptions can not be processed all at once. Hence, they need to be stored in a data structure for later processing. This data structure changes dynamically during the process and new elements will be added to it, e.g., when the specification of an object of an axiom for a currently processed `OM#requires` specification needs to be adapted itself. The latter case occurs, e.g., when a related mdo-individual is created the new or when it exists in the model but its type is being changed.

Assuming now that the necessary adaptions are processed in a depth-first order, i.e., by first following the path of mdo-specifications such as, e.g., `SPI#definedOn`, down to standard sets or simple parameters before returning to the next `OM#requires` class restriction of the primary formulation entity, the data structure to represent the required specifications within the process can be identified as a stack of model entity and type pairs. Every of the latter pairs on the stack has a reference to the cardinality class restrictions that are required to check when processing it. These class restrictions themselves imply further pairs to be pushed onto the stack when processing them. E.g., a cardinality class restriction might require the creation and specification of a new mdo-individual as an object. When this individual is created and typed, a pair with new class restrictions itself originates. The latest element that was pushed onto the stack represents the current entity to be processed. When processing a class restriction itself, iteration might be necessary in case of cardinality classifiers such as `min 1` or `exactly n` with `n > 1`. I.e., according to the ontology type design considerations from Section 6.2, a standard cardinality class restriction would be of the form `p exactly 1 mdoobject` with `p` being a property such as `OM#requires` or `SPI#definedOn`, but also specifications with `p exactly 2 mdoobject` or `p min 1 expressionobject`, the latter case dealing with expressions, might occur. In the latter cases, the class restriction processing would have an iterative form, still including the necessary recursion mentioned above, until the cardinality class restrictions are satisfied and, optionally in case of `min` or `max`, a modeler states completeness.

The typical situation when adding a new formulation entity to a model and creating its specification is visualized from a snapshot perspective in Figure 6.30.

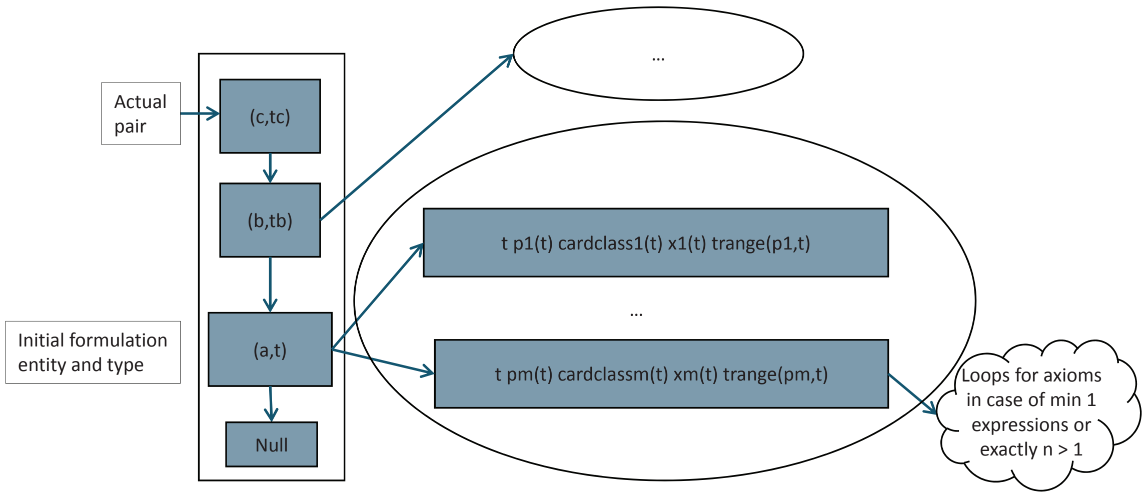

...
Actual pair
(c,tc)
(b,tb)
Initial formulation entity and type
(a,t)
Null
t p1(t) cardclass1(t) x1(t) trange(p1,t)
...
t pm(t) cardclassm(t) xm(t) trange(pm,t)
Loops for axioms in case of min 1 expressions or exactly n > 1

Figure 6.30.: Recursive character of model formulation specifications for retrieving an ontology model's consistency after adding a new formulation entity

As indicated in the figure, the initial formulation entity lies on the bottom of the stack and after finishing its specification, the process terminates. During the process, different mdo-entity and type pairs might be added to the stack together with the respective class restrictions to check. As mentioned above, checking a single cardinality class restriction itself can be a complex process. As such, a formalization of the whole formulation entity addition and ontology model consistency retrieval process was figured out as a very complex task during the work on this thesis. To that end, the textual and graphical process considerations of this subsection should suffice for the scope of this thesis. This subsection should now be concluded by a very rough discussion of the algorithmic properties of the DFS-approach.

As the ontology graph structure of ontology represented optimization models has a partial order with formulation entities occuring only as root nodes or subjects in respective axioms, and other axioms propagating down to sets and simple parameters as leave nodes and objects, a DFS-like processing of the specificational adaptions should both be complete and correct. Completeness would mean that all required adaptions will be processed, whilst correctness would mean that the specifications of the optimization model are consistent in an ontological way after termination. I.e., no contradiction to the class restriction knowledge exists in the model after the use case process has terminated. The termination of the process itself depends on its concrete realization and the interaction with the modeler. I.e., if it can be guaranteed that the iterative treatment of the `min 1` expression specifications will terminate at some point, termination of the process itself should be guaranteed. Nonetheless, the discussed process provides only a single use case in the model formulation that would typically be accompanied by operations that are not that formalized at all. E.g., the typing of data entities as variables was mentioned as a typical user interaction before that can not be completely captured by ontology class restriction focussed model formulation processes. To that end, termination is not an important property for the model formulation process itself. In the end, on the level of formulating complete models, a modeler would always have to decide by himself, when a specification is completed.

The exemplary consideration of a recursive consistency retrieval process as outlined above should suffice for the scope of this thesis. For sure, the outlined process itself is not the only possible approach to such a consistency retrieval in model formulation.

6.3.5. Types and Formulation Choices concerning Integrality Requirements

This subsection is intended to discuss another example of a choice in the formulation of a model, where reusable types provide the necessary formalization. The essence of the example is the choice between a linear type that requires the introduction of certain integral, especially binary, variables, and a continuous variant that in contrast introduces nonlinearities. In order to stay consistent with the surrounding water network setting of this section, the activation and deactivation of pumps was chosen as a specific form of the exemplary modeling situation. The latter topic occurs in operative planning models for water distribution systems.

As the scheduling of pumps is an operative task, it would be untypical to consider it in the context of the pipe network design case study model from Subsection 6.3.2. To that end, the discussed modeling fragments will be taken from another model that can be found in the literature. This subsection will only discuss the relevant modeling excerpts and not the complete model. As a reference, the reader is referred to the article [BGS09] that deals with the formulation and solution of a NLP model for the operative planning in large drinking water networks. The modeling fragments considered here are concerned with the short-term activation and deactivation of pumps in the operative planning shedule that runs over a discrete time-scale. The motivation for trying to forbid the short-term activation and deactivation of pumps is given by the fact that respective frequent switching can reduce the service life of the pumps as well as that it can be costly from an energy perspective. The authors mention the two tasks to forbid both short-term activation and deactivation together with a third task, the problem of "alternating discharge", which will not be considered here. For the problem of forbidding or at least reducing the short-term pump activation and deactivation, a novel quadratic and smooth NLP formulation is proposed in [BGS09] and compared to a standard MIP formulation which uses binary pump activity indicator variables. The two formulations will be presented right now, followed by the introduction of the respective reusable formulation types. Finally, the application scenarios in the model formulation will be discussed.

At first, a standard linear formulation using binary indicator variables will be presented. The formulation forbids the short-term activation and deactivation of pumps by fixing the activity status of a pump for `K` periods in a row in case of changes. Both the activation and the deactivation can be treated by one group of constraints each. The formulation excerpt in AMPL can be found in Figure 6.31. For a literature reference, the reader is referred to [BGS09], but care has to be taken due to an adaption of the group of deactivation constraints by the author of this thesis. As it will be explained below, a respective multiplicative factor of two in the right-hand side of the constraint group was identified to be mistakeably wrong and has thus been adapted to a factor of one.

The MIP formulation introduces the binary indicator variables `Y` for all pumps at all times of the discrete planning horizon `0..T`. This number can thus be quite huge although pumps typically rarely occur in comparison to normal pipes in a network. The parameter `K` controls the number of periods for which a pump's activation status is constant in case of a change. Typically, `K` is set to the value three in order to freeze the satus for two subsequent periods after an activation or deactivation in a respective period. The constraint group `PumpSwitchingLimitActivationwithBinaries` enforces a pump that is being activated in period `t+1` to stay on for the periods until time `t+K`. Vice versa, `PumpSwitchingLimitDeactivationwithBinaries` enforces a pump to stay off for the respective number of periods by letting the right-hand side take a zero value in

```
set Nodes;
set FullPipes within {Nodes,Nodes};
set Pumps within FullPipes;
param T > 0;
param K > 0 integer;
var Y{Pumps,0..T} binary;
s.t. PumpSwitchingLimitActivationwithBinaries{(i,j) in Pumps, t in 0..(T−K)}:
K ∗ (Y[i,j,t+1] − Y[i,j,t]) <= sum{l in 1..K}(Y[i,j,t+l]);
s.t. PumpSwitchingLimitDeactivationwithBinaries{(i,j) in Pumps, t in 0..(T−K)}:
sum{l in 1..K}(Y[i,j,t+l]) <= K ∗ (Y[i,j,t+1] − Y[i,j,t]) + K;
```

Figure 6.31.: Forbid short-term pump activation and deactivation by MIP constraints

case of a deactivation in period `t+1`. More specific, right-hand side values greater equals `K` are enforced in case of activated or state-preserving pumps, and the value of zero only results in case of a deactivation. Note, that in the original reference [BGS09], the right-hand side term `Y[i,j,t+1] - Y[i,j,t]` has a factor of `2*K`, leading to a right-hand side value of `-K` in case of a deactivation. As the left-hand side is a sum of the respective 0-1 indicator variables, the constraint would thus become infeasible. For this reason, the formulation was adapted to a rigth-hand side multiplicative factor of one by the author of this thesis. Finally, it should be mentioned that both of the latter constraint groups are linear.

The NLP formulation for the pump switching scenario has been developed in [BGS09] and does not require the introduction of any binary variables but rather formulates on the volumetric flow rates `Q` through the pumps. It is restricted to keeping the activity status constant for two or three periods which respectively means to freeze the status for one or two periods after the period in which a change took place. To that end, the parameter values `K=2,3` are instantiated by the formulation excerpt in Figure 6.32. The four constraint groups are divided into two parts for activation and deactivation. In case of freezing the pump activity for two periods subsequent the period in which a change took place (`K=3`), two constraint groups, including also the one with the name-part "`OnePeriod`", are required. The same holds for the deactivation constraints. The activation constraints are linear, whilst the deactivation constraints are nonlinear but smooth. As the authors mention, the activation constraints are certain by fobidding the short-term pump activation in any flow-scenario, whilst the deactivation constraints are only stated to be "reliable [BGS09]".

The constraints underlying the formulation excerpts in Figure 6.32 are derived in [BGS09]. For an intuitive interpretation of "how they work", the reader might note that flow in pumps is always nonnegative in sign and that certain bounds are given on the multiplicative factors of the constraint formulations. In the constraint goup

```
set Nodes;
set FullPipes within {Nodes,Nodes};
set Pumps within FullPipes;
param T > 0;
param alpha >0;
param cDeact > 0, <= 0.5;
param cOne > 0.5, < 2;
param cTwo > 1/3, < 2;
var Q{FullPipes, 0..T};
s.t. PumpActivationOnePeriodForbiddenNLP{(i,j) in Pumps, t in 0..(T−2)}:
(cOne + 1) ∗ Q[i,j,t] + (cOne − 2) ∗ Q[i,j,t+1] + (cOne + 1) ∗ Q[i,j,t+2] >= 0;
s.t. PumpActivationTwoPeriodsForbiddenNLP{(i,j) in Pumps, t in 0..(T−3)}:
(cTwo + 1) ∗ Q[i,j,t] + (cTwo − 1) ∗ Q[i,j,t+1] + (cTwo − 1) ∗ Q[i,j,t+2] + (cTwo + 1)
    ∗ Q[i,j,t+3] >= 0;
s.t. PumpDeactivationOnePeriodForbiddenNLP{(i,j) in Pumps, t in 0..(T−2)}: Q[i,j,t+1]
    − cDeact ∗ (Q[i,j,t] + Q[i,j,t+2] − sqrt((Q[i,j,t] − Q[i,j,t+2])^2 + alpha^2)) >= 0;
s.t. PumpDeactivationTwoPeriodsForbiddenNLP{(i,j) in Pumps, t in 0..(T−2)}:
Q[i,j,t+1] − cDeact ∗ (Q[i,j,t] + Q[i,j,t+3] − sqrt((Q[i,j,t] − Q[i,j,t+3])^2
+ alpha^2)) >= 0;
```

Figure 6.32.: Forbid short-term pump activation and deactivation for one and two subsequent periods by quadratic NLP constraints as in [BGS09]

`PumpActivationOnePeriodForbiddenNLP`, the activation of a pump is, e.g., characterized by flow rates `Q[i,j,t]` of zero and `Q[i,j,t+1]` greater zero such that the negative sign of the `Q[i,j,t+1]`-term enforces also `Q[i,j,t+2]` to be positive. The next, the formulation types for the latter two formulations of the short-term pump activity modeling should be discussed.

Encapsulation of the two latter formulation artifacts is achieved by the formulation types `WaterNetForms#IntegralPumpSwitchingConstraintsActivation-DeactivationOperativeTime` and `WaterNetForms#ContinuousPumpSwitchingConstraintsActivation-DeactivationMaxTwoPerOperativeTime` which each contain the respective constraint groups for both activation and deactivation of pumps from the latter examples. The first type can be used for a generous number of periods in which the pump status stays constant after a change, whilst the second type freezes the state for two periods after the first period in which the change took place, i.e., the pump operation is constant for three periods. This is done by integrating all four constraint groups from Figure 6.32.

`WaterNetForms#IntegralPumpSwitchingConstraintsActivationDeactivation-OperativeTime` requires exactly one of `WaterNet#Pumps` and

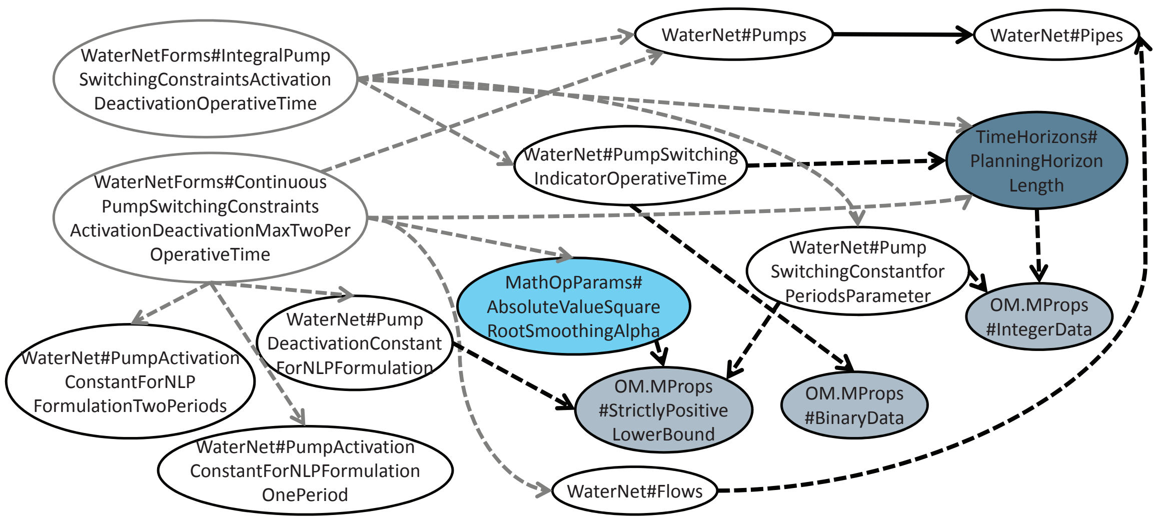

WaterNet#Pumps
WaterNet#Pipes
WaterNetForms#IntegralPump SwitchingConstraintsActivation DeactivationOperativeTime
TimeHorizons# PlanningHorizon Length
WaterNet#PumpSwitching IndicatorOperativeTime
WaterNetForms#Continuous PumpSwitchingConstraints ActivationDeactivationMaxTwoPer OperativeTime
WaterNet#Pump SwitchingConstantfor PeriodsParameter
MathOpParams# AbsoluteValueSquare RootSmoothingAlpha
OM.MProps #IntegerData
WaterNet#Pump DeactivationConstant ForNLPFormulation
WaterNet#PumpActivation ConstantForNLP FormulationTwoPeriods
OM.MProps #StrictlyPositive LowerBound
OM.MProps #BinaryData
WaterNet#PumpActivation ConstantForNLPFormulation OnePeriod
WaterNet#Flows

Figure 6.33.: Ontology definitions for the two pump switching formulation types

`TimeHorizon#PlanningHorizonLength`, yielding the set of pumps and the length of the planning horizon. Furthermore, exactly one individual of the mdo-type `WaterNet#PumpSwitchingConstantforPeriodsParameter` is required in order to have the parameter `K` available. The core requirement is given by exactly one of `WaterNet#PumpSwitchingIndicatorOperativeTime` which represents the binary indicator variables `Y`. This mdo-type has a further class restriction `OM#dataMProp exactly 1 OM.MProps#BinaryData` which explicitely represents the fact that the indicator data will always be of binary value nature.

Compared to the latter, the NLP formulation type `WaterNetForms#Continuous-PumpSwitchingConstraintsActivationDeactivationMaxTwoPerOperativeTime` has more extensive type requirements. This is due to the fact that a formulation such as in Figure 6.32 requires further respective parameters, such as `cDeact, COne` and `CTwo` that are represented by the three types in the lower left corner of Figure 6.33. Futhermore, a parameter `alpha` that is used in the quadratic and smoothed formulation of an absolute value is required for the formulation of the deactivation constraints. The respective type requirement is given by the type `MathOpParams#AbsoluteValueSquareRootSmoothingAlpha`. The NLP formulation type does not require any binary indicators or the respective value of `K` since it is fxed to `K=3`. Nonetheless, as the NLP formulation uses the volumetric flow rates through the pumps, exactly one individual of `WaterNet#Flows` is required. The type requirements for pumps and the length of the planning horizon are the same as for the integral formulation type.

As there is the beneficial missing of the binary indicator requirement in the NLP formulation type together with some exclusive specific parameter requirements that the MIP-type does not have, a subtype relation can not be established for the two latter formulation types. Also, fixing the parameter `K` in the integral MIP-type to value three would not help. To that end, one gets two formulations with some type requirement similarities where the types capture some large and complicated statement blocks for reuse. Using either one of the formulations by types is an example of a formalized choice though it is not that explicit as the two types are no siblings in a subclass hierarchy. In the model formulation, the two types might be applied as follows:

- By Formulation Recommendation: As both types require pumps and operative timesteps, the introduction of such sets and parameters to a model could lead to an automated recommendation of multiple types containing the two as "pump specific" ones. Nonetheless, it would be up to the user to choose between a formulation that has a requirement for an mdo-concept that explicitely introduces binary variables, or a nonlinear one.
- By Semantic Search: If a user knows that he is looking for constraints that formulate on pumps, he can formulate a semantic query for a respective ontology class, i.e., a formulation type. Filtering out all the solutions that explicitely require an "integral" mdo-type afterwards could yield the NLP formulation type as a single result.

6.4. Soft Constraints

This section presents the possibilities to exploit typical MIP-modeling techniques within the ontology model formulation approach at the example of *soft constraints*. Soft constraints provide an opportunity to weaken typical MIP restrictions, such as upper bounds or capacities, by allowing a violation which is being penalized in the objective. In the remainder of this section, the design of special soft constraint formulation types and their usage will be discussed. As reusable types for constraint formulations can only capture the constraint modeling operations, a further part of the discussion will be designated to the specification and adaption of goal/objective formulations. Soft constraint modeling is generally applicable in the MIP-modeling context, so the formalizations of this section will be given in a pattern-like way without a concrete application domain such as, e.g., water networks. Furthermore, the suggested types and ontology concepts are not part of the implementations accompanying this thesis.

The soft constraints modeling approach consists of multiple partially optional steps to be applied to an excerpt of a mathematical program. In order to give a short introduction, you may consider the following three prototypical MIP-constraints and their transformation into a soft constraint form:

$$\sum_{j\in J} a_j x_j \leq b \Rightarrow \sum_{j\in J} a_j x_j - u \leq b, \tag{6.13}$$

$$\sum_{j\in J} a_j x_j \geq b \Rightarrow \sum_{j\in J} a_j x_j + v \geq b, \tag{6.14}$$

$$\sum_{j\in J} a_j x_j = b \Rightarrow \sum_{j\in J} a_j x_j - u + v = b. \tag{6.15}$$

$$u, v \geq 0 \tag{6.16}$$

The initial constraints contain some parameters $b, a_j \in J$ and variables $x_j \in J$. Their soft constraint variants introduce variables u and/or v with non-negativity requirements. These variables are used to allow for the violation of the original constraints. In case of an upper bounding constraint, the variable u allows to exceed the upper bound, whilst in case of a lower bound, a variable v allows to undercut the respective constraint. In case of an equality constraint, both variables can be used.

Introducing additive terms and nonnegative variables u, v is only a first part of the soft constraints modeling technique. A second part that also takes place within the constraint system would be the upper bounding of the variables u, v by some parameters U, V in respective bounding constraints. This can be used to ensure that the absoulute violation of constraints does not exceed a certain level. But the introduction of such bounds is optional. Of more importance is the punishment of constraint violations within the objective function. This "third part" of the modeling technique reveals an important consequence for the formulation type based modeling in the ontology formalism: In accordance to the ontology type design principles from Section 6.2, modeling fragments will be encapsulated into new types rather than that a configuration of more generous placeholders for single types or whole models takes place. By this, the modeling of soft constraints will lead to new formulation types that capture the modeling statements of the adapted constraints together with the new variables u, v and optionally their upper bounds. But what can not be encapsulated into a constraint formulation type is the respective adaption of the objective. This adaption will be introduced right now and its modeling within extra goal-types will be a topic further below.

To complete the consideration of the objective modeling, consider the case of a minimization problem. Any additive term in the variables u, v that increases the value of the objective monotonically within these variables is a penalization of the respective usage of the variables. As such, a convenient way is to introduce a proportionality factor $c > 0$ for every such variable and add an additive term $+cu$ or $+cv$ to the objective. Care has to be taken in case of a maximization objective. In order to penalize the violation of constraints in that case, a decrease of the objective function has to be achieved and the respective factors $c < 0$ are to be accounted with a negative sign when using the convention $+cu$ or $+cv$ for the additive terms.

Let us now start with the modeling of reusable formulation types by exploiting the constraint formulation types that encapsulate the first two parts of the soft constraint modeling technique from above. Figure 6.34 gives an overview on the type definitions.

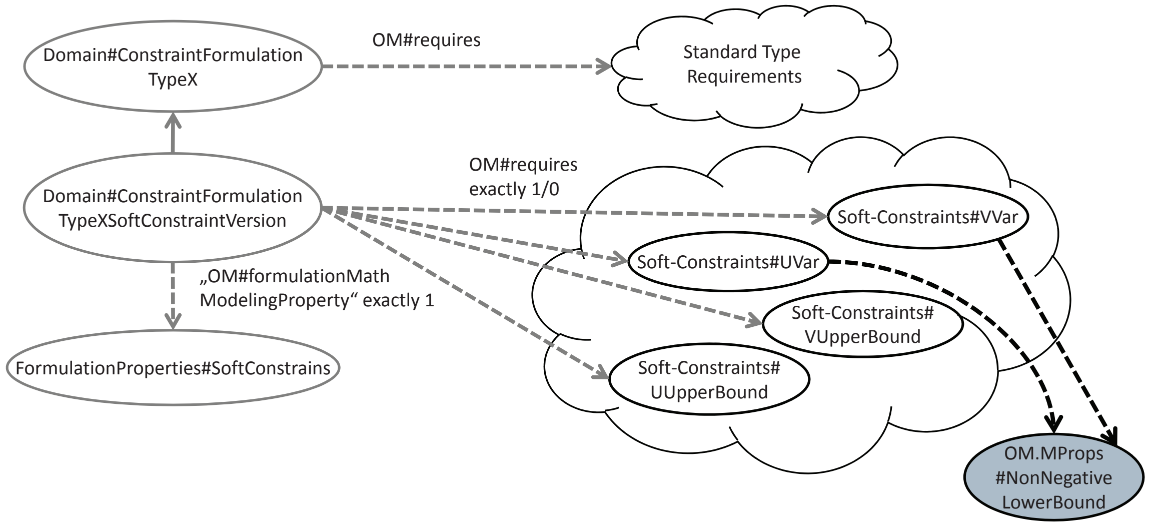

Domain#ConstraintFormulation TypeX
OM#requires
Standard Type Requirements
Domain#ConstraintFormulation TypeXSoftConstraintVersion
OM#requires exactly 1/0
„OM#formulationMath ModelingProperty“ exactly 1
FormulationProperties#SoftConstrains
Soft-Constraints#VVar
Soft-Constraints#UVar
Soft-Constraints# VUpperBound
Soft-Constraints# UUpperBound
OM.MProps #NonNegative LowerBound

Figure 6.34.: Ontology definitions for soft constraints' constraint types

A soft constraint variant should be provided by a subtype of an originating constraint formulation type as there is no need to delete any preceding type requirements. By this, the old type requirements can be inherited and only need to be extended by the additional requirements for the variables u, v as well as their optional upper bounds. The mdo-concepts for the respective type requirements stem from an extra vocabulary `Soft-Constraints`. Their usage is to be prescribed in an individual way by every soft constraint type. Soft constraint variants for upper bounding might require exactly one `Soft-Constraints#UVar` used for the exceedance of upper bounds, but would typically not use and require any `Soft-Constraints#VVar`. Furthermore, the usage of upper bounding constraints for the new variables and the derivation of respective statements from the type is optional and can be handeled differently by any type conceptualization and associated derivation implementation. Special attention has to be paid to a possible indication of the "soft constraint" property of a formulation. In order to perform a semantic search for soft constraint variants in the model formulation, a concept `FormulationProperties#SoftConstraints` might be of interest. The concepts can be added to a formulation individual by an ontology axiom using the property `OM#formulationMathModelingProperty` and its usage can be prescribed on class level by a respective cardinality restriction. By this, the semantice formulation type provides a simple pattern for a semantic search of soft constraint variants of a constraint.

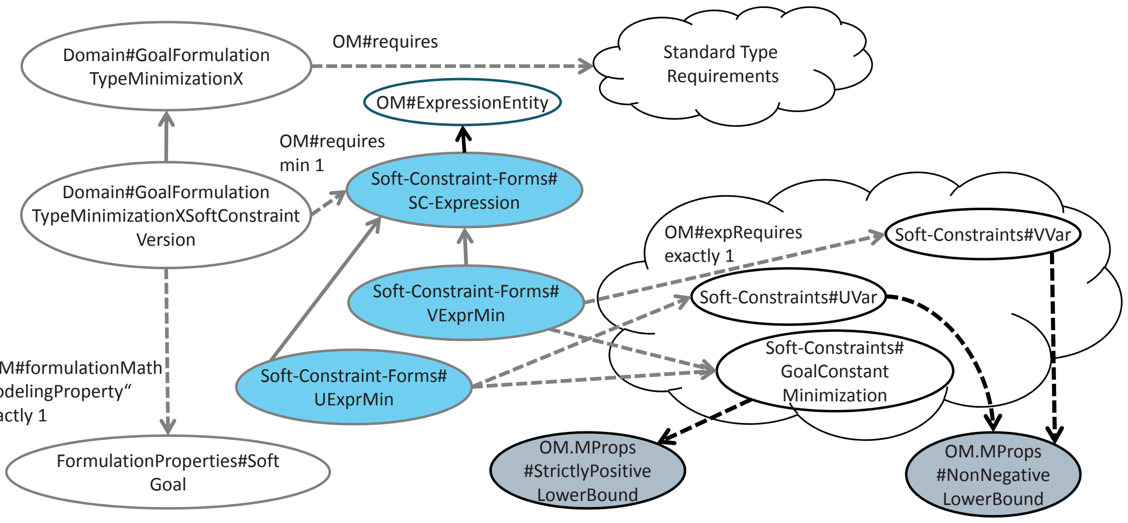

Domain#GoalFormulation TypeMinimizationX
OM#requires
Standard Type Requirements
OM#ExpressionEntity
OM#requires min 1
Soft-Constraint-Forms# SC-Expression
Domain#GoalFormulation TypeMinimizationXSoftConstraint Version
OM#expRequires exactly 1
Soft-Constraints#VVar
Soft-Constraint-Forms# VExprMin
Soft-Constraints#UVar
„OM#formulationMath ModelingProperty“ exactly 1
Soft-Constraint-Forms# UExprMin
Soft-Constraints# GoalConstant Minimization
FormulationProperties#Soft Goal
OM.MProps #StrictlyPositive LowerBound
OM.MProps #NonNegative LowerBound

Figure 6.35.: Ontology definitions for soft constraints' goal types

In order to provide the penalization of constraint violations in the objective, adaptions in the goal type and instance specification have to take place. As different soft constraints, which all have to be punished within one single objective, can be used in a model, the goal formulation types themselves should be designed in a flexible way. This can be done by providing a subtype of every goal type. The conceptualization of subtypes also allows it to stay consistent with the ontology formalizations and design principles that were given in this thesis.

The soft constraint goal type uses `OM#ExpressionEntity` subtypes in order to be flexible enough to allow for a generic extension of the objective expression by different penalization terms. Expressions were introduced in Subsection 6.1.3 in the context of certain min-cost-flow constraints. A soft constraint capable goal can use a generic, positive number of such expressions, depending on the number and character of soft constraints used in a model. As soft constraint objectives are conceptualized as special subtypes, the requirement of `min 1 Soft-Constraint-Forms#SC-Expression` can be stated, which also is consistent with the design considerations from Section 6.2. As one does not know whether and how many exceedance or shortfall soft constraint formulations will be used with an objective, the requirement of the expression type `Soft-Constraint-Forms#SC-Expression` from the extra vocabulary `Soft-Constraint-Forms` represents a non-instantiable type which is to be refined to the subtypes `Soft-Constraint-Forms#UExprMin` or

`Soft-Constraint-Forms#VExprMin`. These instantiable expression types are then specific to their usage by having the respective requirements for variables and parameters using the property `OM#expRequires` together with sharp cardinality restrictions of the form `exactly 1` and furthermore encapsulating the right "sign-requirements", e.g., for minimization. The minimization specific penalization scaling parameters are encapsulated in the class `Soft-Constraints#GoalConstantMinimization`. As in the case of constraint types, the soft constraint capability of a goal type could be indicated by an extra axiom using the `OM#formulationMathModelingProperty` property.

To conclude this section, one can say that providing the soft constraints modeling technique via reusable types is possible whereupon the necessity to adapt both constraints and objectives complicates the model formulation. When planning to reformulate an existing constraint or group of constraints as soft constraints, a modeler might first of all choose a respective soft constraint subtype. As the type is a subtype, the respective adaptions for the constraint individual only concern the type change and the specification and relation of the additional variables and parameters. The relation of a unique individual to indicate the soft constraint property can be automated. What the modeler should not forget about is to adapt the goal specification by adding respective expression entities with their specifications and possibly changing the type of the objective. When performing these adaptions, also the direction of the optimization has to be taken into account.

6.5. Conclusion

Within this chapter, multiple examples of formulation- and mdo-types were presented. Besides their specifications in terms of cardinality class restrictions, whole subsumption hierarchies of types were discussed, enabling *formalized choices in modeling*. Capabilities for automatization in the model formulation were illustrated together with a discussion of formulation type reusage. Based upon the various examples, also a characterization of the class restriction design for single types was given in terms of cardinalities.

The example formulations were presented in the context of two domains, namely network flow problems and water distribution systems. For the network flow problems, two consecutive generalizations of a standard min-cost-flow problem were discussed in terms of commodities and a split of the integral and continuous commodity flow goods. The example models led to the formalization of different parallel subsumption hierarchies for network flow mass balance types as well as a very general representation of resource availability constraints that makes use of flexible expression specification capabilities within the ontology approach. The case study for water networks introduced an application domain whose formalizations can partially be grouped under the ones of the network flow domain. The integration of a mass balance constraint individual with constraints for pipe headloss in Subsection 6.3.4 provided an illustrative example for the interplay of the different formalizations as well as for an exemplary workflow in model formulation. Pipe headloss formulations were investigated in detail in Subsection 6.3.3 and provided insights into formalized choices in modeling as well as that they demonstrated how larger groups of AML statements can be encapsulated for reuse. More specific, polynomial smoothing and different linearizations were presented and respective type formalizations were brought into a hierarchy. Automatizations in ontology specifications for retrieving an ontology model's consistency were discussed as well as that the perspective for the analysis of model properties and the service composition was illustrated. In Subsection 6.3.5, another modeling decision was formalized by two types for pump switching constraints in models for water distribution systems. The formulations came from the literature where they had been applied to a pump operation planning problem. A key feature was the choice between the linear but integral and the continuous but nonlinear formulation. Finally, Section 6.4 explained how the modeling technique of soft constraints can be applied in the formulation of ontology represented optimization models by the aid of respective types.

Concerning the idea of reusing formulation types in the formulation of ontology represented model's, different examples were presented that might allow for saving costs in terms of the time for formulating a model. Furthermore, reusing formulations in ontology types should avoid mistakes in the specification of AML statements. Besides others, multiple individuals of the same types in the third min-cost-flow model as well as the huge statement blocks for the pipe headloss linearizations were mentioned as examples. Furthermore, a decoupling of expression structure was demonstrated for the flexible types of resource availability constraints and a network flow objective. Inside the latter types, different formalizations of expression types can be reused as well as that the general principle of subsumption is exploited. The subype relation for expression and formulation types was characterized as only refining old and adding new class restrictions. This can lead to only a small number of required operations when changing types in an existing model formulation.

The support of tasks in the model formulation was illustrated by different examples. The initial creation of a commodity-set in a min-cost-flow model specification led to the consideration of *model formulation recommendation*, a technique that was only outlined by examples and should be investigated further in the future research. The basic idea was to recommend formulation types together with data model adaptions, where the ontology class restrictions match both with the types of individuals in the current model specification as well as with the intended adaptions of the data model and types, e.g., the "multiple commodities" in the considered example. Besides the formulation recommendation, the formalized choices in modeling that were reprented by different subsumption hierarchies provided another case of support in the model formulation. E.g., for the pipe headloss types, the generation of different model variants and systems to be solved in an iterative process (or in parallel) was highlighted as something to automatize.

The central model formulation task that occurred in the different examples was the retrieval of an ontology model's consistency according to the class restrictions of the concerned types. A necessity for specificational adaptions can be triggered by different model formulation operations such as, e.g, the creation of a new formulation entity. Besides general process considerations in Subsection 6.3.4, different automatizations for creating the required specifications were discussed. These automatizations, besides others, covered the cases where specificational axioms and especially new objects occurring in the axioms were added automatically to a model, as well as that existing objects were related in axioms, possibly including a change of their type. Unfortunately, all these latter methods have no access to the modeler's real intention. Hence, a decision has to be made whether a modeler should trust automated "guesses" that are based on some rules, or whether automatization should not be exploited and the modeler should be interrogated.

Concerning the requirements from Section 4.2, this chapter demonstrated different new features for their coverage:

- **Model Formulation by abstract Types**: The usage of abstract types as a central representational element has already been discussed in Section 5.7. In this chapter, many concrete examples of formulation types were given and the automatization of different operations in model formulation was illustrated. Especially, the retrieval of an ontology model's consistency was discussed from a process perspective. Different examples illustrated that reusing types can be simple and can save specificational work. E.g., when changing to a subclass type, the adaptions in a specification only concern the differing class restrictions. In addition to that, huge blocks of statements that are encapsulated in formulation types save work compared to AML specifications. Though there are limitations in automatizing the model formulation process, this requirement can now be stated as covered.

- **Model Semantics**: The coverage of this requirement has already been discussed in Section 5.7.

- **Automated Modifications and Recomposition**: A detailed example for the coverage of this requirement was discussed in scope of the pipe headloss linearization types in Subsection 6.3.3. As the example illustrates, subsumption hierarchies of formulation types provide *formalized choices in modeling*. The choices can be exploited for automated modifications and a recomposition of the system. The recomposition is based on an automated model analysis and service generation (recapitulate Section 5.6 of the last chapter). In this chapter, the perspective to iteratively recompose systems after changes was highlighted by the pipe headloss example.

- **Models as Services and Composition**: The coverage of this requirement has already been discussed in Section 5.7.
- **System Integration**: The coverage of this requirement has also been discussed in Section 5.7.
- **Service Enabling**: Furthermore, the coverage of this requirement has already been discussed in Section 5.7.
- **Compatible with (semantic) Web Technologies**: And finally, also the coverage of this requirement has been discussed in Section 5.7. This chapter considered the OWL 2 design of concrete formulation- and mdo-type ontologies and hence introduced no new technologies or architecures that could contradict the coverage of the requirement.

7. Consolidation: A MINLP Model for Pipe Renewal Planning

Besides the formulation type focussed considerations, Section 6.3 gave a short introduction to the planning of water distribution systems and important parts of the physical network model including underlying physical laws. During the work on this thesis, a Mixed Integer Nonlinear Programming (MINLP) model for the problem of pipe renewal planning was formulated and solved. The model catches up important fragments of the models in [BDL$^+$06] and the case study variant in Section 6.3, but further extends them to a multi-year planning horizon with different network configurations to be considered over multiple periods a day. To that end, it represents a complex new model for a practical application. This section aims at presenting this model and some numerical results on its solution by two different MINLP solvers.

Concerning the connection to the preceding chapters, the reader will find some variants of the types from Section 6.3 and Subsection 6.1.1 in the model formulation, as well as that the case study domain of drinking water networks will be exemplified and motivated further. Furthermore, a general planning approach in the domain will be discussed. The planning approach corresponds well to the development process and architecture of optimization systems that was outlined in this thesis in Section 5.1. This chapter furthermore provides a part of the evaluation for the considerations of the case study in Section 6.3 by showing the relevance of concepts from the framework of this thesis. Besides that, the numerical results might be of some interest for practitioners doing optimization in water distribution systems.

The next section provides a short literature review on the associated planning problem. Section 7.2 then gives a problem definition and discusses the model solution in the scope of the broader approach mentioned above. Section 7.3 finally presents the MINLP model to be solved. The results will be given in Section 7.4. There are in fact two results: At first, numerical results for an AMPL formulation of the model with an older version of the solver Bonmin will be presented. Then, an implementation of the model under AIMMS which is solved with the solver BARON in an up to date version will be given. The results under AIMMS/BARON were achieved by L. Brinkmeyer based upon the mathematical model formulation developed by the author of this thesis and given within this thesis in Section 7.3. The AIMMS/BARON model and results are documented in the bachelor thesis [Bri15] and will only be cited in the relevant excerpts within this chapter.

7.1. Literature Review: Pipe Renewal Planning and Optimization

Planning the renewal of pipes in a water distribution network integrates the two ideas of planning a target network design and determining a strategy how to achieve this design in a cost efficient, long-term planning process. There are not many publications dealing with this integrated view. This section gives a short literature review on different optimization approaches whose common result is a network with modified pipes. The static problem of pipe network design, which is already known from the MINLP model in [BDL^{+}06] that was modified in Subsection 6.3.2, represents the standard problem underlying most of the approaches.
A further relevant approach that is also used in practice for planning the development of German municipal drinking water networks is described in the diploma thesis [Hen08]. The author presents an integrated approach that is based upon a two-staged planning process. At first, an existing network's pipe dimensions are minimized in a cost-efficient way by a simulation-based heuristic. The second planning step then introduces either a Mixed Integer Programming (MIP) approach or a genetic algorithm to determine an optimal target network topology. Technical assets such as valves are included in the planning procedure for an optimal target network. The approach is evaluated on different networks of realistic size and character. Strategies how to determine a plan for achieving the target netwok structure over a long-term planning horizon are described.
Whilst the integrated view has not been regarded too often, the study of the network design problem for drinking water networks has been carried out under different viewpoints with different optimization methods. Since the nonlinear network hydraulics have to be considered in all kinds of approaches that vary from MIP over MINLP to meta-heuristics, also different model reformulations and linearizations can be observed.
Sherali and Subramanian [SSL01] present a Branch-and-Bound approach with a guaranteed gap to the global optimum. They start with a split-pipe design (to be explained in what follows) formulation of the nonlinear network design problem. Based upon a convex combination, the nonlinear headloss equations in Hazen-Williams form are being reformulated in a linear way, where the nonlinearity in the model now is contained in a formulation of the link flows as a function of a new variable. This nonlinear constraint is being relaxed by a linear outer approximation such that the resulting lower bounding problems can be solved efficiently by LP methods. An upper bounding heuristic is presented and the whole approach is being evaluated on the standard network design test problems (which are actually of very small size) and a more realistic Blacksburg test network. The guaranteed solutions that are within $10^{-4}\%$ of optimality provided a novelty in MIP-based approaches for the pipe network design problem at that time.
The idea of split-pipe designs, i.e., the determination of pipe diameters by assigning portions on the length of a pipe for each available diameter was exploited by many

approaches relying on the Linear Programming Gradient method (LPG). See, e.g., [KS89] for an improved variant. Here, the solution process of the pipe network design problem is split into two phases. In a first phase, network heads and optimal portions of pipe diameters on the length segment are computed for a fixed flow scenario. Due to the split-pipe formulation the computation can be performed in a linear fashion by solving a LP model. In a second stage, the flow is changed due to the so-called gradient of the objective function (GOF). The approach in principle lacks on the guarantee of reality suited solutions (,i.e., no splitting of diamter portions exists in optimal, realistic solutions) and the general guarantee of global optimality. Hence, partitioning approaches such as discussed above for [SSL01], or (Mixed-Integer) Nonlinear Programming Approaches, constitute important parts of the newer research.
Besides that, meta-heuristics have been, and are still, applied to the drinking water network design problem.
Dandy et al. [DSM96] provide an improved genetic algorithm (GA) for the pipe network design problem. They improve the standard GA performance by using a variable power scaling of the fitness function in combination with an adjacency mutation operator and a gray coding instead of a binary one. Their results are provided on the standard New York Tunnels Problem.
An approach based upon Tabu Search is given by [dR04]. The choice of suited pipe diameters from a discrete set of commercially available dimensions is accompanied by the characterization of neighbourhood solutions. Here, neighbourhood solutions are characterized by changing exactly one diameter of one pipe at a time. The results present some improvements on the small sized standard design problem networks from the literature.
Novel approaches integrate the integrality and nonlinearity requirements of pipe diameters and headloss relations by formulating as Mixed Integer Nonlinear Programming (MINLP) models. As already mentioned before, Bragalli et al. [BDL$^+$06] present a MINLP model formulation for the network design problem. The nonlinear headloss equations are included in the model formulation in a locally smoothed variant in order to fulfill NLP solver requirements. The discrete choices on commercially available pipe diameters are modeled by continuous diameter variables that are restricted to the discrete set of available diameters by some diameter increment constraints. Consequently, the discrete cost function is fitted to a smooth variant for the principally continuous diameter variables. The model is solved using different MINLP codes that are interfaced to the modeling environment AMPL [FGK03]. The results are presented on the standard test networks from the literature as well as the real world fossolo network near Bologna and compare well to the results of heuristic or MIP approaches. As the authors highlight, the intractability of MIP solutions due to a high number of binaries wehn formulating linearizations can be avoided by MINLP approaches.
To that end, MINLP approaches constitute a promising area of actual WDS

planning research. Whilst NLP techniques might be successfully applied for the continuous and 0-1 decisions in pump activity planning [BGS09], the treatment of integral decisions in the planning of pipe network designs, including pipe dimensions, cannot be neglected. The developments in MINLP solver technology reflect this development. The solver Bonmin [PLA$^+$08] provides different methods such as a MINLP Branch-and-Bound and outer approximations in a hybrid scheme that unfortunately cannot guarantee global optimality for nonconvex problems. The major reason is the relying on the solution of possibly non convex NLP relaxations by local NLP solvers. The outer approximation solver Baron [MN05] uses as different approach that creates convex LP relaxations whose solutions converge to a global optimum even for non convex problems.

7.2. Multiperiod Renewal Planning: Problem Statement and Approach

As it was already mentioned in Subsection 6.3.2, water distribution systems in Germany are characterized by overdimensioned networks. Since renewal measures are necessary building measures, the idea to develop plans based upon the estimated times of renewals is a target to be investigated within this chapter.
The basic idea is to develop an integrated model whose result is a multi-year pipe renewal strategy. Since multiple time-steps in combination with large network sizes and detailed hydraulic computations will introduce a high complexity into the model, the long-term vision is to integrate the developed model into the following iterated solution approach that has been investigated in the dissertation [Hal15] for the problem of tank planning:

- Reduction: The first step constitutes of an iterated removal and aggregation of network pipes and nodes. For the goal of changing the size of instances, sequences of pipes and pipes in parallel are aggregated to single pipes as well as that tree-structures and end-nodes are removed. Some of the techniques may introduce errors to the network hydraulics behaviour whilst others produce virtual but equivalent network formulations. Descriptions of such techniques and the mathematics behind can, e.g., be found in [BGS09, TD99].
- Instantiation: An abstract optimization model is instantiated with the instance data for the reduced network.
- Optimization: The optimization instance is solved by a suited algorithm, e.g., for MINLP problems.
- Simulation: The optimization solution, which is based upon the reduced network, is validated by a hydraulic simulation on the detail level of the originating network. Different demand scenarios are evaluated.

- Modification: If the solution process did not produce a satisfying and valid solution, it will be reiterated with a new configuration. In principle, the reduction method itself, the optimization model behind the instantiation, and the solution algorithm implemented in a solver, can be modified.

A visualization can be found in Figure 7.1. For a technical realization further data transformation steps would be required. The approach compares well to the execution workflow and iterated development process of the optimization systems in this thesis as it was presented in Section 5.1.

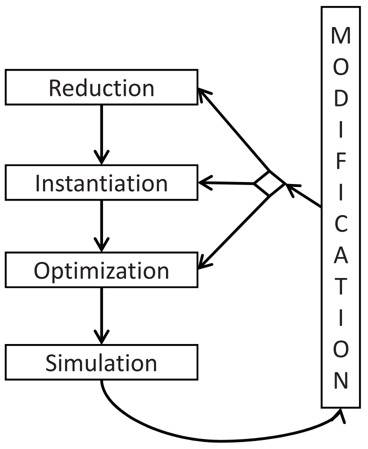


Figure 7.1.: A process for water optimization applications. Visualized process is in similarity to an approach presented in [Hal15]

The modification step has to be further explained in terms of the changes it will introduce to the next iteration of the optimization process:

- Reduction: Change the level of aggregation by reconfiguring the network reduction algorithm and recomputing the reduced network.
- Instantiation: Change the abstract model by changing formulations or adding/removing constraint groups and compiling a new instance.
- Optimization: Change the optimization algorithm or its configuration.

The first step towards a realization of this planning process was to develop and study an integrated standard model formulation used in the optimization step. This formulation is very complex and can be solved up to medium-sized practical instances. Nonetheless, support for the reduction and modification techniques above might allow for solving large-scale instances with high quality/level of detail in the future.

Speaking of requirements, the following aspects are to be integrated into the general model formulation:

1. Planning horizon and time-coupling of building decisions: A multi-year planning horizon with a scalable number of discrete timesteps for building measures has to be considered. A special subset of pipes that is to be renewed in the planning horizon has to be marked up and these pipes are to be annotated with a minimum and maximum time parameter for the renewal. The decisions on renewing a pipe have to be coupled over time such that for every pipe to renew, a building measure happens exactly once within the bounds for the renewal.

2. Discrete diameters: Diameters have to take values from a discrete, prescribed set of commercially available pipe diameters.

3. Nonlinear diameter cost function and goal: The goal of the model should be to minimize pipe diameters and material cost according to the monotonically increasing pipe diameter cost function. Since this function is nonlinear and in principle occurs for every pipe at every possible time of building, a suited way of modeling that reduces binaries and nonlinear effects has to be found. The cost function must be represented in a way such that solvers, e.g. for NLP, can work with it.

4. Budget: A simple declining budget at every time in the planning horizon has to be adhered to.

5. Demand scenario, periods and tanks: The model has to be accurate in the way that every developed network variant in the planning horizon is valid for a simple period simulation. I.e., tank fillings and at least two periods for day and night have to be respected.

6. Nonlinear headloss: The pipe friction caused headloss should be included in an exact, nonlinearized way. The assumption of rough pipes is acceptable such that the Prandtl-Kármán model for the friction factor can be applied. This introduces a further nonlinearity into the model due to the diameter dependency. The network hydraulics are to be determined uniquely due to the input boundary parameters for heads and inflow rates.

7. Bounds and source inflows: Nodal heads are to be bounded by minimum and maximum parameters and arc flow rates are to be implicitely bounded by maximum pipe speeds. The fact that diameters come from a discrete and bounded set should be mapped by the model as well as that the source outflows are to be bounded from above. Source outflows in addition to that are only allowed to flow into one outgoing direction. Tanks should have a maximum filling characterized by their maximal head parameter.

The overall goal can be concluded as to find a renewal strategy in form of a decision when to renew and how to dimension the pipes. Thereby, the strategy should lead to a cost minimal plan in terms of minimized pipe material costs and diameters.

7.3. MINLP Model

This section presents the nonconvex MINLP Model for the pipe renewal problem. At first, the network model is introduced.
The general network model distinguishes sources, tanks and junction nodes as node types, whilst arcs only occur in the form of normal pipes. Due to complexity and solvability reasons, it was decided to leave out pumps and valves, i.e., it is assumed that all network valves are opened and that there are no pumps inside the distribution network. The mathematical formulation of the network model uses the following sets, where a certain subset BP of pipes to be renewed within the planning horizon is also marked up explicitely:

- Nodes N to capture all network nodes.
- Junction Nodes JN for nodes through which water flows and at which water can be consumed.
- Sources S for nodes that supply a certain inflow of water.
- Tanks Tk for nodes that can store and hence supply water, but can also consume water from the network to store it for later periods.
- Pipes P through which water can flow.
- BuildPipes BP as the subset of all pipes which are to be renewed within the planning horizon.

From a mathematical perspective, the sets underlie the following subset relations, where it is mentioned that pipes are only allowed to have source nodes as starting nodes:

$$\begin{aligned} J &\subseteq N \\ S &\subseteq N \\ Tk &\subseteq N \\ P &\subseteq (S \cup J) \times J \\ BP &\subseteq P \end{aligned}$$

The whole consideration of the network model underlies two time perspectives. One perspective for the ongoing development of network structure within the planning horizon as well as one set of periods for simulating a concrete network at a fixed point in the planning horizon over multiple hours. Since the development of network structure depends on the bounds for the renewal time of every pipe, additional parameters have to be introduced for every pipe $p \in BP$. To conclude one introduces:

- Renewal times in the planning horizon: $0, \cdots, T$ with $T > 0$.
- Pipe dependent interval of time for a renewal:

$$0 < TR_{i,j}^{start} < TR_{i,j}^{end} \leq T \;\; \forall (i,j) \in BP. \tag{7.1}$$

- Periods and uniform length of periods (in seconds) for modeling a reference day:

$$Per > 0, \delta^{Per} > 0. \tag{7.2}$$

For the Budget requirement the following is introduced:

- Budget at every time of a renewal: $Budget_t \;\; \forall t = 1 \ldots T$.

For the modeling of the pipe diameters and the material cost of a unit pipe length with a certain parameter, the following is introduced:

- the number of commercially availabe pipe diameters: $dnum$.
- the pipe diameter values in $[m]$: $d_i^{co} \;\; \forall i = 1, \ldots, dnum$.
- the cost of a unit length of pipe material for a certain diameter: $Cost(d_i^{co}) \;\; \forall i = 1, \ldots, dnum$. In principle, the function $Cost(\cdot)$ is defined on the discrete set of pipe paraemters. Due to NLP solver requirements it will be necessary to fit it to a continuous, $\mathcal{C}^2$-smooth function, as it will be discussed in what follows

The netwok model has further parameters as shown in table 7.1.
The building decisions will be indicated by the pipe diameter variables for the renewal pipes BP at every time in the planning horizon. Due to its diamter dependency, these pipes further require a variable for the aggregated resistance coefficient in the Darcy-Weisbach formula (6.8). The pipe diameter will be forced to take only values from a predefined set, therefore further 0-1 variables are required. The building measures at a time will further be indicated by a binary variable that also forces a pipe and time dependent cost variable to take a nonzero value. The

Set	Meaning	Symbol	Unit
$\forall tk \in Tk$	Radius of Tanks	RT_{tk}	m
$\forall tk \in Tk, t = 0, .., T$	Initial Head at Tanks	$H^{In}_{tk,t}$	m
$\forall s \in S$	Initial Head at Sources	H_s	m
$\forall s \in S$	Min. Inflow of Source	$QS_{min,s}$	$\frac{m^3}{s}$
$\forall s \in S$	Max. Inflow of Source	$QS_{max,s}$	$\frac{m^3}{s}$
$\forall p \in P$	Length of Pipe	L_p	m
$\forall p \in P$	Roughness of Pipe	k_p	m
$\forall p \in P$	Pipe-Diameter in act. netw	$da_{i,j}$	m
$\forall p \in P$	Max. Flow Velocity in Pipe	$v_{max,p}$	$\frac{m}{s}$
$\forall p \in BP$	Min. Diameter of Pipe	$d_{min,p}$	m
$\forall p \in BP$	Max. Diameter of Pipe	$d_{max,p}$	m
$\forall x \in JN \cup T$	Min. Head at Junction/Tank	$H_{min,x}$	m
$\forall x \in JN \cup T$	Max. Head at Junction/Tank	$H_{max,x}$	m
$\forall x \in JN, t = 0, .., T, per = 1, \dots Per$	Demand at Junction	$D_{x,t,per}$	$\frac{m^3}{s}$

Table 7.1.: Parameters of the MINPL model

hydraulics are besides the resistance coefficient, which is a variable only for the renewal pipes, represented by nodal head and pipe flow variables at all times and periods. A summary is given in table 7.2

Set	Meaning	Symbol	Unit
$\forall x \in N, t = 0, .., T, per = 1, .., Per$	Junction head at time, after per.	$H_{x,t,per}$	m
$\forall p \in P, t = 0, .., T, per = 1, .., Per$	Pipe flow rate at time, in per.	$Q_{p,t,per}$	$\frac{m^3}{s}$
$\forall p \in P, t = 0, .., T$	Pipe resistance at time	$r_{p,t}$	$\frac{m^5}{s^2}$
$\forall p \in BP, t = 0, .., T$	Pipe diameter at time	$d_{p,t}$	m
$\forall p \in BP, \mu = 0, .., dnum - 1, t = 0, .., T$	Pipe diam. increments at time	$X_{p,\mu,t}$	$-$
$\forall p \in BP, t = 1, .., T$	Indicator for pipe renewal at time	$n_{p,t}$	$-$
$\forall p \in BP, t = 1, .., T$	Cost of building meas. at time	$cost_{p,t}$	$-$

Table 7.2.: Variables

In what follows, the model's goal and constraints will be introduced in the order of the requirements from section 7.2.

Planning horizon and time-coupling of building decisions

Each pipe from the set BP is to be renewed exactly once. Therefore, the sum of all building measure indicators over the timesteps in the planning horizon is exactly one, whilst the indicator variables are forced to the value one, whenever a diameter changes. Otherwise, the character of the objective (minimization, see, (7.11) in the following) will force the indicator variables to zero. All diameters start at an initial value:

$$\sum_{t=TR_{i,j}^{start}..TR_{i,j}^{end}} n_{i,j,t} = 1 \qquad \forall (i,j) \in BP \tag{7.3}$$

$$\frac{d_{i,j,t} - d_{i,j,t-1}}{d_{max,i,j}} \leq n_{i,j,t} \qquad \forall (i,j) \in BP, t = 1..T \tag{7.4}$$

$$\frac{d_{i,j,t-1} - d_{i,j,t}}{d_{max,i,j}} \leq n_{i,j,t} \qquad \forall (i,j) \in BP, t = 1..T \tag{7.5}$$

$$d_{i,j,0} = da_{i,j} \qquad \forall (i,j) \in BP \tag{7.6}$$

Discrete diameters

The diameters are forced to take only values from a discrete set by an incremental modeling as it can, e.g., be found in [BDL+06]:

$$d_1^{co} + \sum_{\mu=2..dnum} (d_\mu^{co} - d_{\mu-1}^{co}) \cdot X_{i,j,\mu-1,t} = d_{i,j,t} \tag{7.7}$$

$$\forall (i,j) \in BP, t = 0..T$$

$$X_{i,j,\mu+1,t} \leq X_{i,j,\mu,t} \tag{7.8}$$

$$\forall (i,j) \in BP, \mu = 1..(dnum) - 2, t = 0..T$$

Nonlinear diameter cost function and goal

The first step to include the nonlinear pipe diameter cost into the model is the generation of a twice-continuously differentiable function $Cost(\cdot)$ defined on a bounded interval. To that purpose, the following discrete data known from the Hanoi test network (,see, e.g., [AM05]) will be considered, where the values have been converted to SI units:

Diameter d_i in $[m]$	0.3048	0.4064	0.508	0.6096	0.762	1.016
Cost c_i	45.73	70.40	98.39	129.33	180.75	278.28

Table 7.3.: Diameter cost data

To fit the six data points, an ansatz with a polynomial of degree four that should fit the data $(d_i, c_i)_{i=1,\cdots,6}$ is made. More specific, due to big scale differences of the data, a weighted ansatz of the form

$$Cost(x) = p_{\mathbf{a}}(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4, \tag{7.9}$$

with a vector $\mathbf{a}^t = (a_0, a_1, a_2, a_3, a_4)$ of coefficients that minimize the weighted least-squares objective is made:

$$\min_{\mathbf{a}} \sum_{i=1,\cdots,6} (\frac{1}{d_i} \cdot (p_{\mathbf{a}}(d_i) - c_i))^2. \tag{7.10}$$

A similar approach for a diameter cost function can be found in [BDL+06].
By applying an implementation of the general Levenberg-Marquardt method for nonlinear goal functions from the Octave-Forge packet `optim 1.2.2` one obtains an acceptable result, where the residual lies in the range from `8.95e-005` to `5.23e-003`. A plot of the resulting function can be found in Figure 7.2:

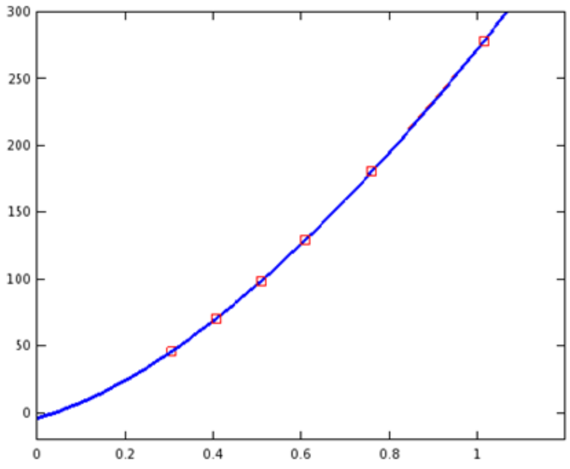


Figure 7.2.: Fitted function $Cost(\cdot)$

With the available function one can now introduce the objective of the model. In order to avoid nonlinearities in the goal function that could stem from products of variable cost terms with 0-1 variables, as well as an occurence of the nonlinear fitted cost function, the the continuous variables $cost_{i,j,t}, (i,j) \in BP, t = 1, \ldots, T$ are introduced. The variables count the cost of a unit length of pipe material of a respective diameter whenever building measures take place. To that end, they need to be multiplied by pipe lengths and summed over all occurences in the objective function. The value of the variables is determined by multiple groups of constraints. The first two groups tie the variables to the value of the nonlinear function $Cost(\cdot)$

in case of a building measure, whilst the third group sets the cost variables to zero when no building measures take place. To that end, the introduction of a big-M parameter is necessary.

$$\min \sum_{(i,j)\in BP, t=1,\ldots,T} L_{i,j} \cdot cost_{i,j,t} \tag{7.11}$$

$$s.t.\ \ cost_{i,j,t} - Cost(d_{i,j,t}) \leq M \cdot (1 - n_{i,j,t}) \quad \forall (i,j) \in BP, t = 1..T \tag{7.12}$$

$$(-1) \cdot cost_{i,j,t} + Cost(d_{i,j,t}) \leq M \cdot (1 - n_{i,j,t}) \quad \forall (i,j) \in BP, t = 1..T \tag{7.13}$$

$$cost_{i,j,t} \leq M \cdot n_{i,j,t} \quad \forall (i,j) \in BP, t = 1..T \tag{7.14}$$

Budget

The budget is modelled by a simple restriction at every time in the planning horizon:

$$\sum_{(i,j)\in BP, t=1,\ldots,T^*} L_{i,j} \cdot cost_{i,j,t} \leq \sum_{T=1,\ldots,T^*} Budget_t \qquad \forall T^* = 1..T \tag{7.15}$$

Demand scenario, periods and tanks

A mass balance constraint (6.7) has to be introduced, where the set of nodes contains all junction nodes and a constraint has to be inserted for every time-step and period. For tank nodes, the mass balance has to be formulated in terms of consistent tank filling levels. These levels are computed by the inflows and outflows from one period to another. To that end, tank radiusses and heights in form of head levels are multiplied to obtain a volume which is made comparable to the average flow rates of a period by dividing through the period lengths:

$$\sum_{(i,x)\in P} Q_{i,x,t,per} - \sum_{(x,j)\in P} Q_{x,j,t,per} = D_{x,t,per} \tag{7.16}$$

$$\forall x \in JN, t = 0..T, per = 1..Per$$

$$\sum_{(i,x)\in P} Q_{i,x,t,1} - \sum_{(x,j)\in P} Q_{x,j,t,1} = \frac{\pi \cdot RT^2}{\delta^{Per}}(H_{x,t,1} - H^{In}_{x,t}) \tag{7.17}$$

$$\forall x \in Tk, t = 0..T$$

$$\sum_{(i,x)\in P} Q_{i,x,t,per} - \sum_{(x,j)\in P} Q_{x,j,t,per} = \frac{\pi \cdot RT^2}{\delta^{Per}}(H_{x,t,per} - H_{x,t,per-1}) \tag{7.18}$$

$$\forall x \in Tk, t = 0..T, per = 2..Per$$

Nonlinear headloss

A pipe headlos relation, as it has been investigated for the ontology formulation case study in Section 6.3.3, has to hold at all times and periods for every pipe. Since the right-hand side of the formulation

$$H_{i,t,per} - H_{j,t,per} = r_{i,j,t} \cdot Q_{i,j,t,per} \cdot |Q_{i,j,t,per}| \quad \forall (i,j) \in P, t = 0..T, per = 1..Per, \tag{7.19}$$

that would be induced by the Darcy-Weisbach representation (6.8) is not twice continuously differentiable at zero, typical NLP solver requirements could not be fulfilled by the latter standard formulation. To that end, the function part $x \mapsto x \cdot |x|$ can be smoothened with a polynomial of the form $g(x) = ax + bx^3 + cx^5$ on a small interval $[-\delta^z, \delta^z]$ around zero. This has been done in [BDL+06] for the similar case of the Hazen-Williams formula. In addition to that, the smoothened formulation without time and period dependencies has been discussed as a formulation type in Section 6.3.3. Continuing with the computation of the polynomial coefficients, in order to enforce a unique solution, one requires a matching of the function values, first and second derivative at both ends of the interval. Due to symmetry one ends up with uniquely determining the three degrees of freedom as $a = 0.375 \cdot \delta^z, b = 3/(4 \cdot \delta^z), c = -1/(8 \cdot (\delta^z)^3)$, see also [Bri15]. Using logic AML constructs that can be resolved by standard MIP modeling techniques in AML environments such as AMPL, one ends up with the formulation

$$\begin{aligned}&\textbf{if } (|Q_{i,j,t,per}| > \delta^z) \textbf{ then } r_{i,j,t} \cdot Q_{i,j,t,per} \cdot |Q_{i,j,t,per}| \\ &\textbf{else } r_{i,j,t} \cdot (0.375 \cdot \delta^z \cdot Q_{i,j,t,per} + 3/(4 \cdot \delta^z) \cdot Q^3_{i,j,t,per} - 1/(8 \cdot (\delta^z)^3) \cdot Q^5_{i,j,t,per}) \\ &= H_{i,t,per} - H_{j,t,per} \;\; \forall (i,j) \in P, t = 0..T, per = 1..Per.\end{aligned} \tag{7.20}$$

Depending on the capabilities of the chosen MINLP solver, either the previously given smoothened formulation or the standard formulation from (7.19) can be added to the model. The concrete choices will be explained in the numerical results of Section 7.4.

The resistance coefficient $r_{i,j,t}$ is determined by the formula ansatz from (6.9) and (6.10), including the Prandtl-Kármán formula. A distinction has to be made for the pipes that are to be renewed within the planning horizon and that hence include the diameter as a variable. Initial heads at sources serve as boundary values for all times:

$$\frac{8 \cdot L_{i,j}}{\pi^2 \cdot g \cdot d^5_{i,j,t}} \cdot (2 \log_{10}(\frac{k_{i,j}/d_{i,j,t}}{3.71}))^{(-2)} = r_{i,j,t} \qquad \forall (i,j) \in BP, t = 0..T \tag{7.21}$$

$$\frac{8 \cdot L_{i,j}}{\pi^2 \cdot g \cdot da^5_{i,j}} \cdot (2 \log_{10}(\frac{k_{i,j}/da_{i,j}}{3.71}))^{(-2)} = r_{i,j,t} \qquad \forall (i,j) \in P \setminus BP, t = 0..T \tag{7.22}$$

$$H_{s,t,per} = H_s \qquad \forall s \in S, t = 0..T, per = 1..Per. \tag{7.23}$$

As mentioned above, for the case $P \setminus BP$, the respective formulas are not treated as constraints but will be used to determine the resistance coefficient $r_{i,j,t}$ as a computed parameter a priori the solution.

Bounds and Source Inflows

Maximum speed bounds need to be added where the speed parameters are transformed to flow rates by multiplication with diameters. First of all, the speed in the pipes with variable diameters is bounded by two quadratic restrictions in each pipe:

$$Q_{i,j,t,per} \leq v_{max,i,j} \cdot (\frac{\pi}{4}) \cdot d_{i,j,t}^2 \quad \forall (i,j) \in BP, t = 0..T, per = 1..Per \tag{7.24}$$

$$(-1) \cdot Q_{i,j,t,per} \leq v_{max,i,j} \cdot (\frac{\pi}{4}) \cdot d_{i,j,t}^2 \quad \forall (i,j) \in BP, t = 0..T, per = 1..Per. \tag{7.25}$$

The nodal head and pipe flow variables as well as all diameters are furthermore bounded by constant maximum values

$$H_{min,x} \leq H_{x,t,per} \leq H_{max,x} \quad \forall x \in JN \cup Tk, t = 0..T, per = 1..Per \tag{7.26}$$

$$Q_{i,j,t,per} \leq v_{max,i,j} \cdot (\frac{\pi}{4}) \cdot da_{i,j}^2 \quad \forall (i,j) \in BP, t = 0..T, per = 1..Per \tag{7.27}$$

$$(-1) \cdot Q_{i,j,t,per} \leq v_{max,i,j} \cdot (\frac{\pi}{4}) \cdot da_{i,j}^2 \quad \forall (i,j) \in BP, t = 0..T, per = 1..Per \tag{7.28}$$

$$d_{min,i,j} \leq d_{i,j,t} \leq d_{max,i,j} \quad \forall (i,j) \in BP, t = 0..T. \tag{7.29}$$

Finally, the summed outflows at sources are forced to stay in their bounds, where each pipe outflow is nonnegative:

$$QS_{min,s} \leq \sum_{(s,j) \in P} Q_{s,j,t,per} \leq QS_{max,s} \quad \forall s \in S, t = 0..T, per = 1..Per \tag{7.30}$$

$$0 \leq Q_{s,j,t,per} \quad \forall (s,j) \in (S \times N) \cap P, t = 0..T, per = 1..Per. \tag{7.31}$$

7.4. Numerical Results

In order to evaluate the MINLP model for pipe renewal planning, two solvers were tested on AMPL and AIMMS formulations of the latter model. Different factors may play a role on the run-time. The complexity of the model increases not only with the network's size which is indicated by the numbers of nodes and arcs, but also with the

number of time-steps in the planning horizon and the uniform number of periods for simulating the network behaviour within each time-step. The time-steps especially multiply the number of many network related variables, making practical instances hughe. Besides nodal heads and pipe flows, the diameter variables for the number of pipes to be renewed within the planning horizon play a major role on the solution time since they are coupled over the times and occur in multiple nonlinear constraints.
The diamter variables are also related to further binary variables for the incremental modeling of the set of discrete, allowed values. Therefore, the number of commerically available pipe diameters also multiplies the number of binaries in the model together with the timesteps and network pipes to be renewed. Further impact factors on the solution time might be the concrete instance parameters such as sharp bounds and the demand scenario. With the latter in mind, the following framework was set up for evaluating the model:

- A constant test network to be described the next was used.
- The time-steps in the planning horizon as well as the number of periods vary with the instances. These are the essential indicators whether finding a renewal strategy over time in a planning horizon of practical length is possible. What also varies with the number of time-steps is the length of the intervals in which pipes can be renewed. For multiple time-steps in the planing horizon it is likely that these intevals increase in length.
- As it was mentioned in the foregoing sections, network size in contrast to time might be reduced by a network reduction. With a fixed network, also a fixed number of building decisions and available pipe diameters can be considered.

The numerical results were obtained with two different solvers on different hardware. This is to be described the next after the introduction of the the network to be considered over time

7.4.1. Test Network

The test network is made up of 54 nodes and 90 pipes. It has a meshed and symmetric structure as you may see in Figure 7.3. Two reservoirs supply the water into the network, where a single tank can be used to store water for later periods. The network data is set to typical values. I.e., the maximum pipe speed bounds are set to $1.5\frac{m}{s}$ and there are uniform pipe lengths of $50m$, except the supply pipes at the reservoirs. The number of available pipe diameters is given by six values from the Table 7.3. Every interior node has a uniform demand of $0.01\frac{m^3}{s}$.

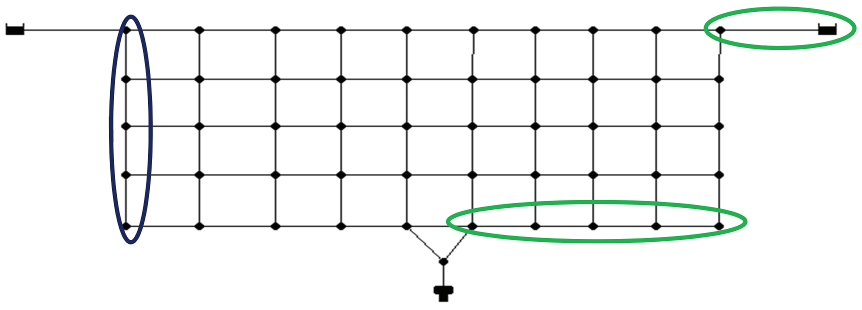


Figure 7.3.: The meshed example network

7.4.2. Results with Bonmin

The first results were computed on a `Intel(R) Core(TM) i5-2520M CPU` with `2.50 GHz` frequency and `64 Bit` memory in late 2012. The used algorithm was the routine `MINLP-BB` of the MINLP solver Bonmin [PLA+08] in version `1.4.2`, performing a pure Branch-and-Bound search based upon NLP relaxations. Since the Branch-and-Bound uses only locally optimal solutions to non convex NLP-relaxations given by the solver Ipopt [WB06], the solutions provided by `MINLP-BB` are not proven to be globally optimal. As the planning goes over time, it is important to mention which build pipes and consecutive intervals in the planning horizon were chosen: A first set of pipes to be renewed in a first time interval of two consecutive steps in the planning horizon (,see, the blue ellipsoid in Figure 7.3) and a set of pipes to be renewed in a later two step range of the time horizon (,see, the green ellipsoid in Figure 7.3) were chosen. The build pipes were the same for all instances and the number of consecutive planning times in which they could be renewed did not vary. The model was formulated in AMPL with a simplified version of the pipe headloss smoothing (7.20) to fit the NLP solver requirements of twice continuous differentiablity. The matching condition of the first order derivative was dropped, leading to a more simple and less nonlinear polynomial of degree three. More concrete, the coefficients $a = 2/3 \cdot \delta^z, b = 1/(3 \cdot \delta^z), c = 0$ from the ansatz of the form $g(x) = ax + bx^3 + cx^5$ on a small interval $[-\delta^z, \delta^z]$ around zero were chosen. As the numerical results with AMPL/Bonmin were not completely satisfactory (see below), no further investigations, e.g., with the fully matching coefficients in (7.20) were performed for the AMPL/Bonmin formulation.

The instances were solved up to an absolute dual gap of 0.1. With an objective value in the range of 10^4 for the best found solution, this value was decided to be acceptable. Since AMPL was used to formulate the model and interface `Bonmin`, the AMPL-Preprocessor was used before applying the solution algorithm. The following table 7.4 supplies the solution times in seconds for the two step solution approach consisting of the steps preprocessing and branch-and-bound. As expected, a strong increase of the solution times can be observed with the number of periods per.

	$Per = 2$	$Per = 4$	$Per = 8$	$Per = 12$
$T = 2$	44	98	235	755
$T = 3$	899	807	3429	5372

Table 7.4.: Bonmin - Solution times in seconds

As the table only lists two rows for the number of time-steps, one first of all ends up with the observation of a drastic increase in the solution times from two time-steps to three time-steps. The solution of a five time-step instance was not possible within a day on the demo machine. For $T = 4$ and $Per = 2$, the best solution was found relatively fast after 23600 nodes, but no significant reduction of the high dual gap could be observed within multiple hours.

As the latter results were not completely satisfactory, it was decided in march 2015 to supervise a bachelor thesis that is concerned with, besides others, an AIMMS/BARON implementation of the latter model as well as the realization of numerical experiments.

7.4.3. Results with Baron

The results under AIMMS/BARON were realized by L. Brinkmeyer and are documented in a detailed fashion in the bachelor thesis [Bri15]. To that end, this subsection will only describe the framework of the experiments, as well as that the solution times using BARON interfaced from AIMMS will be cited. For further information, the reader may consider the latter thesis.

The results were retrieved for the same test network as the results under AMPL/Bonmin. To that end, the pipe lengths, demands, diameters and other technical parameters were also the same. As it was possible to solve instances with more time steps in the planning horizon as in the case of AMPL/Bonmin, the number of consecutive times in the planning horizon during which the build pipes could be renewed was extended with the number of timesteps. Nonetheless, the set of build pipes was the same as in the AMPL/Bonmin case. The demo machine had the following characteristics: Windows 7, 64bit, Intel Core i3-3110M 2x2,4 GHz, 8 GB RAM with an installation of AIMMS Version 4.3.2.3 using BARON version 14. The only modification in the constraints was the usage of the unsmoothed original headloss equation (7.19). In general, the solver BARON implements a polyhedral branch-and-cut approach [MN05] for global optimization allowing to treat general, nonconvex MINLPs with some restriction on the functional expressions used in the constraint formulations. The basic ingredients are polyhedral outer approximations retrieved by recursive functional decompositions. As valid approximations have to be generated out of functional expressions based upon estimation, the constraints have to be formulated with the aid of arithmetical expressions build of supported mathematical functions, using operations such as concatenation, addition or exponation. Since now Baron is capable of treating the absolute value function, no smoothing of the headloss relation was required.

The running times for the instances on a demo machine are to be found in Table 7.5. Not all instances were solved, but all instances for which solution times are denoted were solved to global optimality. Further information can be found in [Bri15].

	$Per = 2$	$Per = 4$	$Per = 6$	$Per = 8$
$T = 2$	24	35	-	65
$T = 3$	-	54	-	330
$T = 4$	45	141	-	5882
$T = 5$	-	-	-	11613
$T = 6$	-	-	6046	-

Table 7.5.: BARON - Solution times in seconds from [Bri15]

7.4.4. Conclusion

To conclude, the applicability of the model to the problem of multi-year pipe renewal planning with MINLP methods is underlined by the AIMMS/BARON results. The network size and meshed structure might be realistic for a practical network that has been processed with techniques of network reduction. The obtained solutions are hydraulically exact due to the nonlinear network hydraulics that were included. A six time-step long planning horizon might be a small but not completely irrelevant number if one thinks, e.g., of a 30 year planning strategy with 5 year long planning steps.

The general approach (Section 7.2) based on network reduction, instantiation, optimization, simulation and modification is of interest both for improving the practicability of the latter model, but also as an example of the optimization system architecure and development approach from Section 5.1. Furthermore, the formulations of the balance constraints (7.16) as well as the pipe headloss equations (7.19) and (7.20) underline the usage of the reusable formulation types from the examples of Chapter 6. Namely, the type `NetForms#SingleCommodityBalancePlanningHorizonandPeriodsDemandOriented` can be used for the standard balance constraints. The types `WaterNetForms#PipeHeadLoss` and the subtype `WaterNetForms#PipeHeadLossC2SmoothDeg5Poly` provide a basis for planning horizon and period extended subtypes that can be used for the headloss equations in the model of this chapter.

The AMPL model, as well as the instance data `.dat`-files for the AMPL/Bonmin experiments, can be found on the digital material accompanying this thesis.

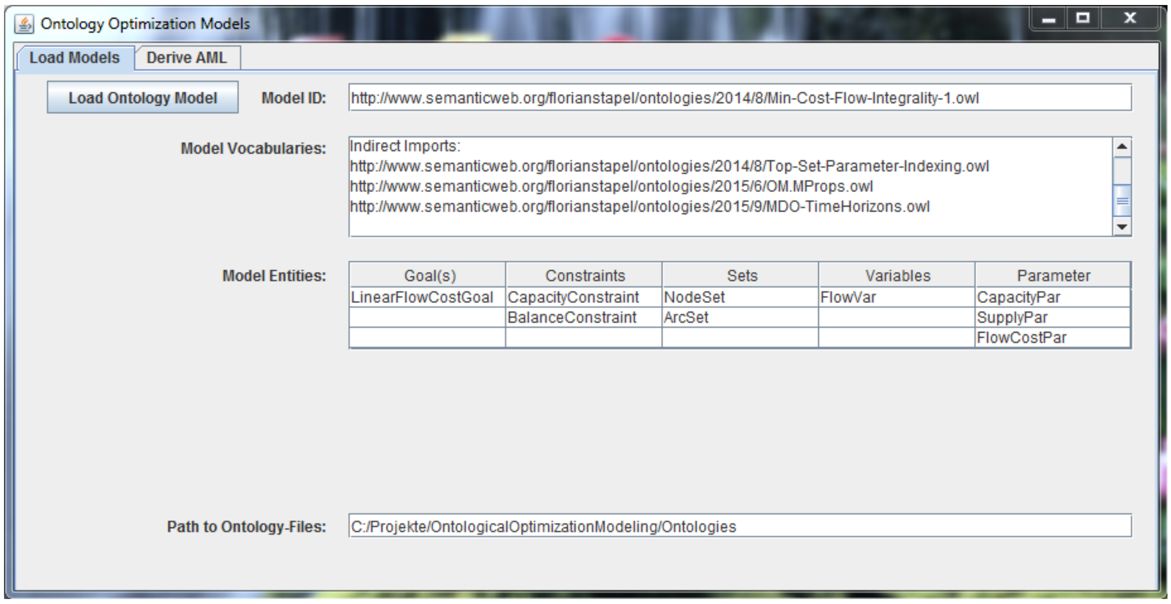
8. Tool Support

This chapter is concerned with a demonstrator application that allows to execute the AML Derivation functionality of the model entity types. Together with the provided type formalizations, queries and derivation implementations, this allows to transform abstract otimization models that were edited in ontology editors such as Protégé into valid AML models in the exemplary language AMPL.

8.1. Funtionality

The application is capable of loading ontology represented optimization models, serialized in OWL-XML format, and performing the statement derivation process as described in Section 5.4 and especially Subsection 5.4.4 for the exemplary target language AMPL. Furthermore, means for validating the derived AML statements against an XML schema document representing valid AMPL grammar productions are implemented. As case study models, all three min-cost flow models from Section 6.1 were implemented in the ontology representation. For the respective types there are vocabulary definitions, queries and derivation implementations which can be found together with the latter ontology models on the digital material accompanying this thesis. The demonstrator is contained as an archive on the digital material. The archive comes without external libraries due to license restrictions. Furthermore, some local paths will need to be adapted in order to run the application on a machine other than the author's demo machine. To that end, an installation without further support might be complicated. If you are interested in getting a "live demo" of the demonstrator or installing it based upon the data on the digital material, you may therefore first contact the author of this thesis. Further information on the digital material accompanying this thesis can be found in Appendix B.

After launching the application, the `Load Models` tab is activated. By clicking on the `Load Ontology Model` button, a dialogue is activated by which the user can choose an ontology optimization model from a path on the file system. Example models can be found under the path `Ontologies/OptimizationModels` on the digital material but should be copied to a respective path on a demo machine. In this section, the standard min-cost-flow model that has been introduced as an AMPL model in Section 5.2 and was specified in the ontology representation in Subsection 5.3.5, will be used as an example. The respective file name is `Min-Cost-Flow-Integrality-1.owl`. After clicking the `Öffnen` button, the window with the `Load Models` tab should look as in Figure 8.1.



Model ID: http://www.semanticweb.org/florianstapel/ontologies/2014/8/Min-Cost-Flow-Integrality-1.owl

Model Vocabularies:
Indirect Imports:
http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-Parameter-Indexing.owl
http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl
http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TimeHorizons.owl

Model Entities:

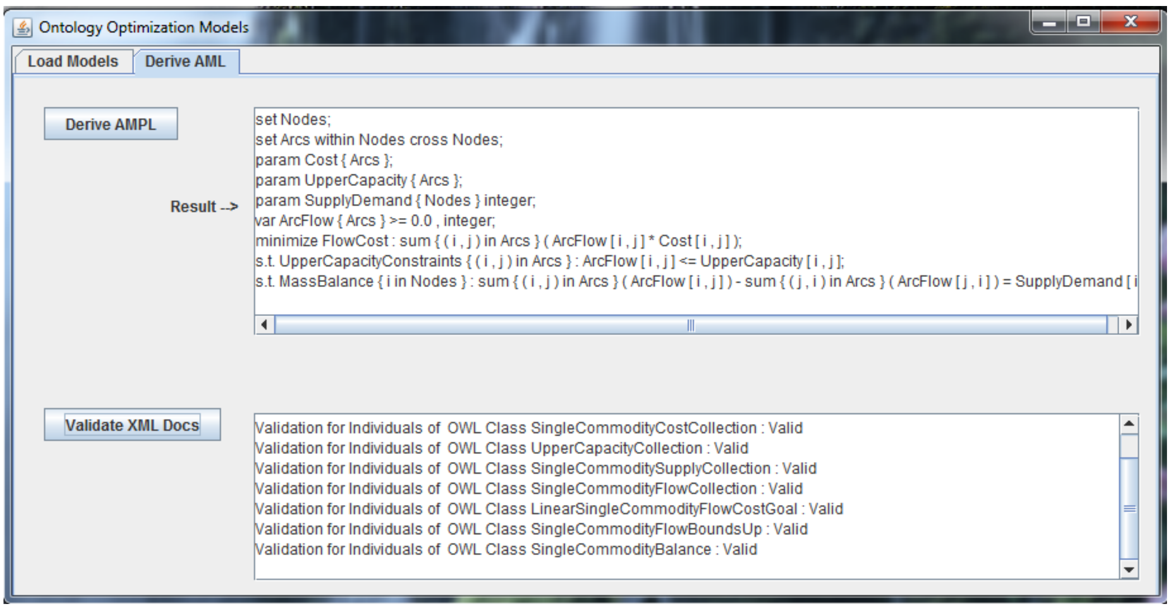
Goal(s)	Constraints	Sets	Variables	Parameter
LinearFlowCostGoal	CapacityConstraint	NodeSet	FlowVar	CapacityPar
	BalanceConstraint	ArcSet		SupplyPar
				FlowCostPar

Path to Ontology-Files: C:/Projekte/OntologicalOptimizationModeling/Ontologies

Figure 8.1.: Screenshot of the Load Models Tab

Some information about the loaded ontology optimization model can be obtained from a text area and a table. A text area labeled `Model Vocabularies` gives information about the directly and indirecly imported ontologies. The loaded ontologies represent the vocabularies introduced within this thesis. The most important such vocabularies, `OM`, `OM.MProps` and `SPI` were introduced in Section 5.3 (recap. also Figure 5.4). Formal definitions of the latter vocabularies can be found in the appendix and are contained in the digial material accompanying this thesis. The table labeled `Model Entities` shows the model entities of the optimization model structured according to the five concepts for goals, constraints, sets, variables and parameters from the `OM` ontology. There are nine model entities contained in the considered example model that represent a single objective/goal, two constraints, two sets, a single family of flow variables and three families of parameters.

By changing to the `Derive AML` tab, the core functiality of deriving a valid AMPL model can be executed right now. A click on the `Derive AMPL` button starts the derivation process for the whole ontology represented model. The resulting XML files containing the AMPL statements will be stored in the `.../Ontologies/Statement-Derivation/` subfolder of the ontology folder installed on the target machine. A serialization of the file contents yields the AMPL model `.mod`-file content and is displayed in the text area right to the `Result->` label. A click on `Validate XML Docs` validates the results of the statement derivation against an XML schema. The results for every model entity type contained in the model are displayed in the lowest text area field. After executing the latter steps, the results should look as in Figure 8.2.



```
set Nodes;
set Arcs within Nodes cross Nodes;
param Cost { Arcs };
param UpperCapacity { Arcs };
param SupplyDemand { Nodes } integer;
var ArcFlow { Arcs } >= 0.0 , integer;
minimize FlowCost : sum { ( i , j ) in Arcs } ( ArcFlow [ i , j ] * Cost [ i , j ] );
s.t. UpperCapacityConstraints { ( i , j ) in Arcs } : ArcFlow [ i , j ] <= UpperCapacity [ i , j ];
s.t. MassBalance { i in Nodes } : sum { ( i , j ) in Arcs } ( ArcFlow [ i , j ] ) - sum { ( j , i ) in Arcs } ( ArcFlow [ j , i ] ) = SupplyDemand [ i
```

```
Validation for Individuals of  OWL Class SingleCommodityCostCollection : Valid
Validation for Individuals of  OWL Class UpperCapacityCollection : Valid
Validation for Individuals of  OWL Class SingleCommoditySupplyCollection : Valid
Validation for Individuals of  OWL Class SingleCommodityFlowCollection : Valid
Validation for Individuals of  OWL Class LinearSingleCommodityFlowCostGoal : Valid
Validation for Individuals of  OWL Class SingleCommodityFlowBoundsUp : Valid
Validation for Individuals of  OWL Class SingleCommodityBalance : Valid
```

Figure 8.2.: Screenshot of the AML Derivation Tab

8.2. Design

Whilst the latter section has explained the scope and usage of the demonstrator application, the classes and methods used to provide the functionality shall be discussed within this section. This is of special importance for users that want to define new types of model entities in the ontology representation. Whilst ontological specifications for these types can be performed in Protégé with the aid of the ontologies provided with this thesis, the implementation of the derivation functionality for generating valid AMPL statements is besides the ontological type definition due to the creation of a new SPARQL-DL query and a java class capable of transforming the query results into statements. Information about how to design a type and implement the derivation and query has been given in Section 5.4. This section shall now, besides others, explain how to insert the respective derivation implementation into the project and where to look for helping functionality to use for the implementational work.

The design will be presented in the form of two class diagrams showing selected classes, attributes and methods. At first, in order to illustrate the design behind the general demonstrational process of loading and deriving an ontology model to AMPL, the reader may take a look at Figure 8.3.
The class `OptimizationOntologyManager` provides the core functionality to load and transform an ontology represented model. In order to do so, further objects of other classes are created inside the respective methods. The constructor can be called with a `File` object representing the location of an ontology optimization model. The

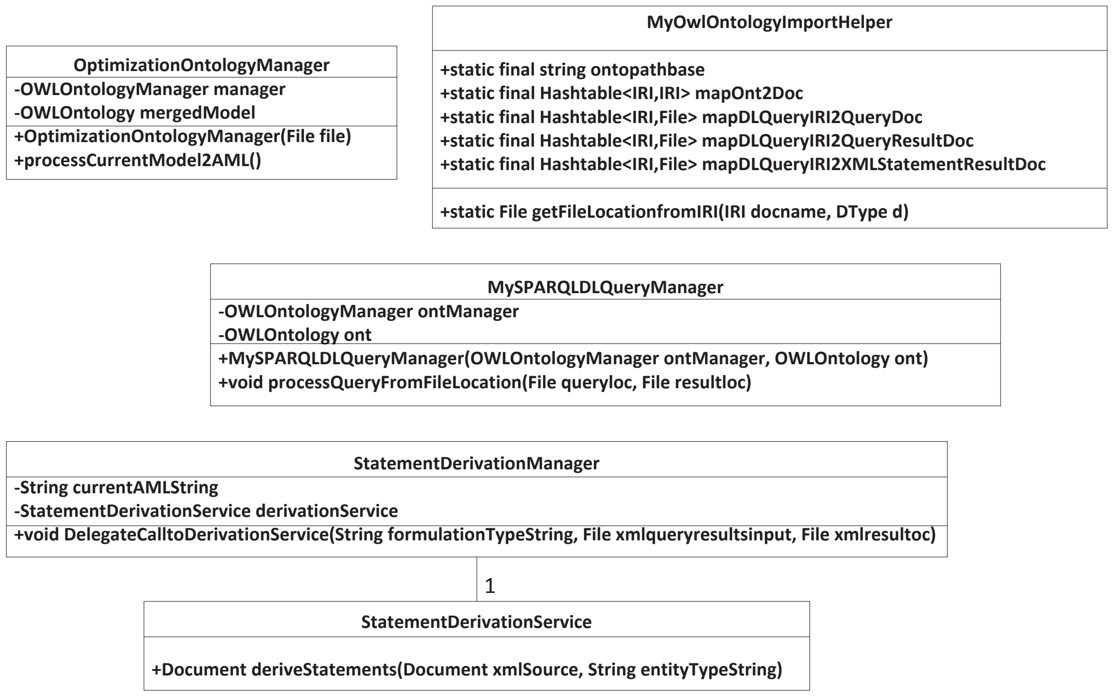

MyOwlOntologyImportHelper
+static final string ontopathbase
+static final Hashtable<IRI,IRI> mapOnt2Doc
+static final Hashtable<IRI,File> mapDLQueryIRI2QueryDoc
+static final Hashtable<IRI,File> mapDLQueryIRI2QueryResultDoc
+static final Hashtable<IRI,File> mapDLQueryIRI2XMLStatementResultDoc
+static File getFileLocationfromIRI(IRI docname, DType d)
OptimizationOntologyManager
-OWLOntologyManager manager
-OWLOntology mergedModel
+OptimizationOntologyManager(File file)
+processCurrentModel2AML()
MySPARQLDLQueryManager
-OWLOntologyManager ontManager
-OWLOntology ont
+MySPARQLDLQueryManager(OWLOntologyManager ontManager, OWLOntology ont)
+void processQueryFromFileLocation(File queryloc, File resultloc)
StatementDerivationManager
-String currentAMLString
-StatementDerivationService derivationService
+void DelegateCalltoDerivationService(String formulationTypeString, File xmlqueryresultsinput, File xmlresultoc)
1
StatementDerivationService
+Document deriveStatements(Document xmlSource, String entityTypeString)

Figure 8.3.: Relevant excerpt of classes and methods for the general derivation functionality

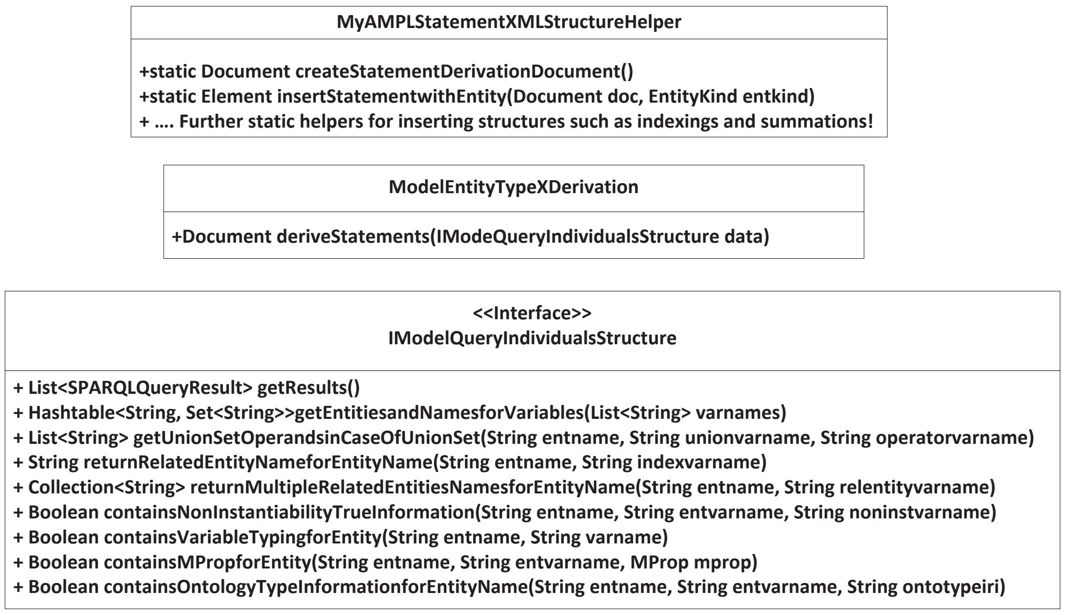
respective call leading to the creation of an `OptimizationOntologyManager` object is performed by the action methods behind the respective buttons on the GUI. The constructor loads the file with the aid of an `OWLOntologyManager` from OWL API version 3.5.0. With the aid of OWL API functionality, directly and indirectly imported ontologies are loaded recursively and merged into the `OWLOntology` object `mergedModel` storing the ontology represented optimization model. A call to `processCurrentModel2AML()` on the `OptimizationOntologyManager` object initiates the AML Derivation process. For the execution of the process, both an object of the classes `MySPARQLDLQueryManager` and `StatementDerivationManager` are created. Inside `processCurrentModel2AML()` all model entity types used in the current optimization model are identified first. For the identified types, the respective query file locations are retrieved with the aid of a helper class method described below. The queries are executed by a call to the `processQueryFromFileLocation(...)` method of the `MySPARQLDLQueryManager` object. This call has the query file location as well as a query result file location as input. In order to process the query, the SPARQL-DL API in version 1.0.0 is used. The results are serialized in SPARQL Query Results XML Format. When the query results are available, the `DelegateCalltoDerivationService` method of the

`StatementDerivationService` object can be called. This method gets the current model entity type IRI in form of a string as an input, together with the locations of the query results and a target file location for the statement results. The derivation functionality is performed by the AML Derivation Service outlined in Section 5.4. To that end, the query results are loaded into a JDOM Document. For compatibility reasons, JDOM in a version starting with number 1 is used here. The query result document and the model entity type string are passed to the method `deriveStatements(...)` of the class `StatementDerivationService` representing the general AML Derivation Service. The result is a JDOM `Document` representing the statements for all individuals of the respective model entity type. These results are further processed and stored in the surrounding method calls.

The functionality above requires locations of ontologies and files. Inside the demonstrator application such resources are treated as IRIs and files. In order to resolve required locations, e.g., for referenced ontologies or queries, the method `static File getFileLocationFromIRI(IRI docname, DType d)` from the class `MyOwlOntologyImportHelper` can be called. It uses some static hashtables of the latter class that store required locations in an ontology-path relative way. The ontology path should have been configured for the target machine during the installation procedure.

The next, the inner workings of the class `StatementDerivationService` representing the composed AML Derivation Service shall be explained. Due to the simple interface, the service object could easily be deployed as a real service and the call of its `deriveStatements(...)` method would not include a reference to a JDOM object and a string but rather a real XML document and a string, both wrapped into a suited protocol such as SOAP. Inside the service's single method, the model entity type string is parsed and used to instantiate the respective type derivation implementation class by the aid of reflection. The derivation implementation's `deriveStatements(..)` method that itself could be deployed as a service integrated in the composed AML Derivation service is then called with the query results. In order to get rid of redundancies, before calling `deriveStatements(..)`, the general AML Derivation Service class loads the query results into a specific class allowing for suited query answering on the query results. The class functionality is prescribed by the interface `IModelQueryIndividualsStructure`. The latter classes that are more specific to the derivation functionality are visualized in Figure 8.4.
Finally, the derivation implementation itself needs to consecutively build up a XML document (as a JDOM document), recap. Subsection 5.4.2. In order to support certain often occuring tasks such as the creation of an initial document that can capture multiple statements, inserting a subtree representing a single statement, indexings, dummy indices and also certain summations, static helper methods from the class `MyAMPLStatementXMLStructureHelper` can be used.

To conclude, for the technical scope of the demonstrator application of this thesis, a



```
MyAMPLStatementXMLStructureHelper

+static Document createStatementDerivationDocument()
+static Element insertStatementwithEntity(Document doc, EntityKind entkind)
+ …. Further static helpers for inserting structures such as indexings and summations!

ModelEntityTypeXDerivation

+Document deriveStatements(IModeQueryIndividualsStructure data)

<<Interface>>
IModelQueryIndividualsStructure

+ List<SPARQLQueryResult> getResults()
+ Hashtable<String, Set<String>>getEntitiesandNamesforVariables(List<String> varnames)
+ List<String> getUnionSetOperandsinCaseOfUnionSet(String entname, String unionvarname, String operatorvarname)
+ String returnRelatedEntityNameforEntityName(String entname, String indexvarname)
+ Collection<String> returnMultipleRelatedEntitiesNamesforEntityName(String entname, String relentityvarname)
+ Boolean containsNonInstantiabilityTrueInformation(String entname, String entvarname, String noninstvarname)
+ Boolean containsVariableTypingforEntity(String entname, String varname)
+ Boolean containsMPropforEntity(String entname, String entvarname, MProp mprop)
+ Boolean containsOntologyTypeInformationforEntityName(String entname, String entvarname, String ontotypeiri)
```

Figure 8.4.: Relevant excerpt of classes and methods for derivation implementations

user wanting to implement a new model entity type has to provide the following:

1. A consistent definition of the type in an ontology that imports the basic vocabularies of the framework presented in this thesis. The type should be specified as a subclass of other types whereever possible. Annotation IRIs for identifying the query and derivation implementation should be provided.

2. A suited SPARQL-DL query for retrieving the relevant individuals and properites from a model.

3. A derivation implementation class for the respective type. If this implementation should be used together with the project, then the respective class should be implemented in the `de.dsor.ontooptmod.derivation` package and implement the interface `IEntityTypeStatementDerivation`. Furthermore, for an implementation within the demonstrator, updating the Hashtables of `MyOwlOntologyImportHelper` in order to be able to resolve the file locations for queries and results on the target machine is required.

The demonstrator application and ontologies provided with this thesis contain type definitions, queries and derivation implementations for over thirty types. The min-cost-flow models from the examples of this thesis have all been implemented as ontology represented models and can be tested with the demonstrator.

9. Conclusion and Outlook

This chapter concludes the results of this thesis and gives an outlook towards future research activities. The outlook part is of special importance as the framework presented within this thesis includes many ideas that leave space for extensions. In summary, a greater vision of decision support system composition and model management will be described and different extensions will be characterized in terms of a gap between the findings of this thesis and the greater vision. The next section starts with the conclusion, whilst the outlook will be given in Section 9.2.

9.1. Conclusion

This thesis presented a novel approach to the formulation of abstract optimization models and the generation of respective optimization systems. The central component is a novel, ontology-based representation of abstract optimization models. This representation, besides others, allows to represent reusable fragments of optimization models' goal and constraint formalizations as simple ontology classes. The formulation types in combination with an abstraction of data structure furthermore allow to represent abstract optimization models as simple ontology graphs. This structure enforces simple model formulation operations. Furthermore, the automated analysis of models and the service composition and data mediation steps for system composition are supported by the representation. Different examples from the domains of network flow problems and water distribution system planning were given to demonstrate the structure of formalizations and the different possible scenarios in model formulation. Important type conceptualizations cover, besides others, hierarchies for network flow balance constraints, water network pipe headloss linearization and smoothing types, as well as flexible types for network flow resource availability constraints which are generic in their inner summation and expression structure. In the future, further such type conceptualizations can be defined by using provided top-level vocabularies such as `OM, SPI` and `OM.MProps`, as well as that in case of extensions, existing type- and mdo-ontologies such as `NetForms` and `Net` can be reused.

Different transformations inbetween the three representation layers were discussed. The generation of AML models by AML Derivation is based on a simple, iterated process of ontology query processing and the transformation of XML results. The logics of the statement derivation are put into derivation implementations which are associated to the reusable types. Different examples of queries and derivation implementations were given together with design considerations. A demonstrator tool and the source code for a variety of queries and derivation implementations accompany this thesis. Concerning the other direction, AML Reengineering has been outlined in short, leaving the investigation open for future research. Finally, it has been discussed and illustrated how ontology represented optimization models can be analyzed in terms of their mathematical properties, and how semantic model instantiation service descriptions can be generated out of the obtained properties. Here, pre- and postconditions can, e.g., be applied to describe the range of possible solvers.

The introductory chapters of this thesis gave a literature review on model management where also the state of the art in optimization technology was described. An extra chapter provided the basic notions and methods for using ontologies and semantic software services in the model representation and system generation approach. Chapter 3 ended with the identification of a research gap concerning the integrated treatment of model formulation by reusable types and the highly automated generation of optimization systems out of an optimization model.

Concerning the exemplary type formalizations, different scenarios for support and automatization in the model formulation were discussed. The hierarchy of pipe headloss smoothing and linearization types in Subsection 6.3.3 gives rise to an iterated or parallel solution of model variants in the respectively composed system, leaving place for an automated improvement of the overall result. E.g., in this example, the explicit formalization of choices should allow to automatically modify the formulation of a model. Furthermore, the retrieval of the consistency of an ontology model specification according to type class restrictions was discussed from a general perspective as well as that examples, e.g., in the context of changing a pipe headloss formulation, were elaborated. For the water network case study model and especially the example of pipe headloss constraints, the adaptions when integrating pipe headloss constraints with network balance constraints were also discussed. Concerning network balance constraints, the considerations for the different variants of network flow models gave insight into the process of extending a model to more general variants, e.g., by introducing multiple commodities. The automated recommendation of formulation types was outlined at the example of network flow balance constraints from the balance type hierarchies and gave a first insight into a method for matching formulation types and model specifications based upon the class restrictions. Multi-instantiation of different types gives rise to the reusage of formulation types. The waternetwork formalizations demonstrated how formulation types can encapsulate greater blocks of AML declarations in a work-saving and error-reducing way.

Finally, some numerical results for a nonconvex MINLP model, which dealt with the renewal planning of pipes in water distribution systems, were given. In the scope of an optimization system ansatz for the water network domain, the solvability of the model for relevant instances was demonstrated in principle, as well as that type formalizations and concepts from the approach of this thesis were brought into the respective context.

The coverage of the requirements from Section 4.2 will now be discussed in a concluding form:

- **Model Formulation by abstract Types**: The ontology representation is centered around the data model abstracted types as it is stated in the requirement. Model formulation can be seen as a process of ontology graph manipulations. By the abstractions of formulation elements and data, the required operations for adding new elements reduce to a small number of such specifications. By the subsumption of ontology classes that represent model entity types, the necessary number of specifications can be further reduced. The idea of reusing formulations in different model contexts was demonstrated by the examples of this thesis. Different scenarios for model formulation were illustrated and the process of recovering the consistency of ontology represented models according to class restrictions was discussed. Though the technique of formulation recommendation has only been outlined by an exemplary scenario, the requirement as a whole is covered by the means provided within this thesis.

- **Model Semantics**: All required types of semantics can be expressed by the representation approach as it should have become clear from the definitions of the basic vocabularies `OM, OM.MProps` and `SPI` as well as the specifications for the different examples of this thesis.

- **Automated Modifications and Recomposition**: The formalization of choices in modeling has been demonstrated by the type hierarchy of the pipe headloss types as well as by the two type formalizations for stopping the short-term activation and deactivation of pumps in Subsection 6.3.5. The scenario of recomposing a system for different model variants was discussed for the pipe headloss smoothing and linearization types. The general methods to generate semantic model service descriptions were presented in Section 5.6. Together, the presented framework covers the requirement, though further research on the evaluation of solutions after the execution of a system as well as the proposition of modifications should take place.

- **Models as Services and Composition**: Model instantiation services that cover the respective requirement were discussed in Section 5.6. Semantic service desriptions and AML models for data instantiation can be generated straight forward out of the ontology representation and the generated service descriptions in principle allow for the composition with solvers and further preprocessing steps.

- **System Integration**: Integrating models into the system is a central step and almost covered by the means to generate model instantiation services. An issue to discuss is the integration of data for the data model that comes along with an abstract optimization model as well as for the inputs and outputs of the further services in the approach. By providing ontology classes for data in the model representation approach, as well as for the inputs and outputs in semantic service descriptions, data mediation might be performed based upon ontology mappings. Placeholders for respective transformations were introduced to the optimization system's service architecture as well as that the general development and execution process(es) allow to generate such transformations after the composition step. By this architecture, the requirement is basically covered though an evaluation of the practicability would be desireable in the future.

- **Service Enabling**: Methods for deriving AML models out of the ontology representation were a topic of Section 5.4 and have been demonstrated in detail. Further means were also provided by the demonstrator presented in Chapter 8. The automated generation of semantic service descriptions together with the AML Derivation functionality and the associated capability to use AML tools for instantiating models with data complete the basic coverage of this requirement. Nonetheless, AML Reengineering has only been presented in terms of some basic ideas and should be investigated further in the future research.

- **Compatible with (semantic) Web Technologies**: This requirement is covered as the design is based on current standards such as OWL, XML, XSD as well as emerging technologies such as SPARQL-DL or SWRL.

9.2. Outlook

The framework presented within this thesis can be seen as part of a greater vision in which multiple decision models, solvers and different software services can be combined to executable decision support systems in a highly automated fashion. Decision proposals are generated by the systems and methods for evaluation and modification lead to automated modifications of models and a respective automated recomposition of the systems. The term model in the context of the vision can cover almost everything that has a mathematical representation and for which the solution of instances yields a respective computational result. Multiple models can be brought into solution processes where preceding models yield the inputs for the ones following subsequently by means of model composition. Service composition in general yields a solution and processing workflow for models and other services where different functionality, e.g., for simulation, visualization and analysis is incorporated into the systems. Provided instance data that has been obtained from databases and users can furthermore be analyzed in order to configure the solution process and the models. Considering the work of users and developers, suited tool support allows to formulate models and specify decision support as a service in a simple and user-friendly fashion. All in all, different efforts from diverse areas such as model management and semantic service composition are combined within the ansatz.

The following depiction will characterize important techniques to develop and aspects to consider for the realization of the latter vision. The following is stated for the respective topics:

- Formulation Recommendation: In the context of model formulation for ontology represented abstract optimization models, means to provide recommendations of model entity types to a user in a certain modeling situation are important for the practicability of the overall approach. As the design of the ontology types introduces a huge number of types that differ in details such as indexing, sign or the application of operations such as a linearization, an automated approach to find suited types for a given model context is of importance. Finding suited types corresponds to a matching of ontology class restrictions. In case of a formulation entity, this would typically mean that the types of required individuals for a recommendation should match the types that occur in the model. In an example from Section 6.1, this led to the recommendation of a commodity-set indexed network balance type to be inserted for a single commodity balance type that had been used in the model before. In the example, the recommendation should have originated from the user-induced introduction of a commodity set to the model. Nonetheless, this thesis did not define or evaluate any measures and methods that allow to compute the "best fitting recommendations". A definition would have to consider the ontology types that occur in the model, but not in a too naive way. I.e., in the discussed example it was important to modify not only the balance constraint formulation entity itself, but also the types and specifications of flow and supply conceptualizations. The "old" data conceptualizations were most similar to the requirements of the "old" formulation type for balance. But the new ones reflected the orientation of the model towards multiple commodities. To that end, the mechanism needs to recommend multiple changes in an integrated consideration. In the example, the change of the formulation type should induce the changes in data. Considering the data entity types at first could lead to a conflict between the "old" balance requirements and the new "commodity orientation".

- Instance Data: The size of model instances and the contained values can have an influence on the solvability of models and hence on the quality of solutions. E.g., when instances for nonlinear models get too huge, approximate solutions might be considered in order to obtain a solution at all. When the data for a model is available, different decisions can be made. To give an example, if the instance data for a water network describes a bigger network, it might be indicated to use a combined approach in which the network's size itself is reduced before a model is instantiated and solved. Afterwards, the solution based upon the simplified network should be evaluated by a simulation. The consideration of instance data hence can have an influence on the target system's architecture and workflow as well as on the system composition. Furthermore, concerning the linearization of nonlinear optimization models, the introduction of a linearization changes the model formulation itself. A decision for a linearization might again be triggered by instance data considerations and knowledge about the applicability of different solution strategies such as the solution of an approximative MIP by a high performance MIP solver. After this decision, which is based on the provided instance data, a different model and system might have to be considered than it would be the case with the originating model. To that end, the process of system generation and modification that was described in this thesis should be extended for the suited consideration of instance data.

- Error Measures and Automated Modifications: The latter point can be extended by the treatment of result data after the computation of solution(s). The quality of solutions should be evaluated in an automated fashion and lead to respective modifications in case of non satisfactory results. E.g., when the linearization of an optimization model yields a solution that is too far away from the solution of a nonlinear and hence more realistic one, other solution approaches might have to be exploited. An example for this scenario can be found in the linearization of pipe headloss constraints in optimization models for water networks. If a hydraulic scenario for a water network that is described within a linearized optimization model's solution turns out as problematic in a simulation, e.g., because a slightly differing flow scenario that was obtained from the more realistic nonlinear simulation model leads to the violation of nodal head bounds, a respective decision on modifications has to be made. Model manipulations were presented as a part of the system generation process in this thesis and a fundament has been laid by the formalization of choices in the model formulation. But the decision on a modification itself was left open as a point for future research. Furthermore, besides the modification of the model, the modification of the system's solution ansatz as it would, e.g., be given by the consideration of a network reduction, provides another opportunity for modifications.

- (Meta-) Heuristics and other Solution Approaches: Concerning the solution of optimization models, other solution approaches than mathematical programming solvers should also be integrated. E.g., problem specific heuristics might be applied to compute solutions. The point with this is that differing solution approaches might require different representations of the optimization problems. To that end, means for providing transformations could be integrated into the approach. This does not have to be a crucial point as there might also be integrated solver solutions that do the transformations on their own. The LocalSolver presented in Subsection 3.3.1 already provides an example for a solver that has an AML language input and treats the application of meta heuristics internally.

- Model and Service Composition: The current approach relies on one decision model that is furthermore restricted to be of optimization nature. In the future, different models for decision support might be solved in a suited solution process. In that process, models provide the input for other models according to the execution order. The solution process of the models might also be combined with the consideration of the whole system's service composition that includes further services, e.g., for preprocessing (network reduction), simulation and visualization. Such services were already considered in the architecture of this thesis, but a general composition approach for different models, solvers and further services leaves place for extensions.

- Other Mathematical Models: The latter aspect introduced the idea that also other decision models than the mathematical programming / optimization models considered within this thesis should be integrated into the framework. As the description of the vision from above says, everything that "*has a mathematical representation and for which the solution of instances yields a respective computational result*" should be considered. This might cover models and solution methods from different areas such as (partial) differential equations, image processing or satisfiability problems in logics. Suited semantic representations of such models from other domains would also be required for the composition approach.

- Model Formulation Tools: Model formulation should be supported by respective tools that make use of the means provided by the semantic representations of models. Considering again ontology-represented optimization models, suited tool support might allow modelers to specify complex mathematical programming models in a very simple way with a small number of required specifications. Furthermore, the tool support should also cover development environments that lead to the respective executable systems by the means provided within this thesis.

- AML Reengineering: As AML environments provide common tools for developing optimization models and systems, means to integrate AML represented models and the AML tool functionalities into the development process should be provided. AML Reengineering was outlined in this thesis as a method to allow for the latter, but its realization is still open for the future research.

Bibliography

[ABKW08] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint Integer Programming: a New Approach to Integrate CP and MIP. In *Integration of AI and OR techniques in constraint programming for combinatorial optimization problems*, pages 6–20. Springer, 2008.

[AM05] M. H. Afshar and M. A. Marino. A Convergent Genetic Algorithm for Pipe Network Optimization. *Scientia Iranica*, 12(4):392–401, 2005.

[AMO93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: Theory, algorithms, and applications.* Prentice Hall, Upper Saddle River and NJ [u.a.], 1993.

[Apa15] Apache Software Foundation. ARQ - A SPARQL Processor for Jena. https://jena.apache.org/documentation/query/index.html. Retrieved 2015-07-07, 2015.

[APG^{+}14] Svetlana Arifulina, Marie Christin Platenius, Christian Gerth, Steffen Becker, Gregor Engels, and Wilhelm Schäfer. Configuration of Specification Language and Matching for Services in On-The-Fly Computing - Version 0.1. Technical Report, University of Paderborn, 2014.

[Baa03] Franz Baader. *The description logic handbook: Theory, implementation, and applications.* Cambridge University Press, Cambridge and UK and New York, 2003.

[BBI^{+}12] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Achille Fokoue, and Zhe Wu. OWL 2 Web Ontology Language Profiles (Second Edition). W3C Recommendation, 2012.

[BDL^{+}06] Cristiana Bragalli, Claudia D'Ambrosio, Jon Lee, Andrea Lodi, and Paolo Toth. An MINLP solution method for a water network problem. In *Algorithms–ESA 2006*, pages 696–707. Springer, 2006.

[Ben62] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

[BG14] Dan Brickley and R. V. Guha. RDF Schema 1.1. W3C Recommendation, 2014-02-25/2014.

[BGS09] Jens Burgschweiger, Bernd Gnädig, and Marc C. Steinbach. Nonlinear programming techniques for operative planning in large drinking water networks. *Open Applied Mathematics Journal*, 3:14–28, 2009.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web Services Architecture. W3C Working Group Note, 2004-02-11/2004.

[Bin96] Meral Binbasioglu. Building Linear Programming Models from Structural Model Components. *Int. Trans. Opl. Res.*, 3(2):151–168, 1996.

[BK93] Hemant K. Bhargava and Steven O. Kimbrough. Model management: An embedded languages approach. *Decision Support Systems*, 10(3):277–299, 1993.

[BKK91] Hemant K. Bhargava, Steven O. Kimbrough, and Ramayya Krishnan. Unique names violations, a problem for model integration or you say tomato, I say tomahto. *ORSA Journal on Computing*, 3(2):107–120, 1991.

[BKM08] Christoph Bussler, Vipul Kashyap, and Matthew Moran. *The Semantic Web: Semantics for Data and Services on the Web ; with 17 tables.* Data-Centric Systems and Applications. Springer, Berlin [u.a.], 2008.

[BL04] Shawn Bowers and Bertram Ludäscher. An Ontology-Driven Framework for Data Transformation in Scientific Workflows. In Erhard Rahm, editor, *Data Integration in the Life Sciences*, volume 2994 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2004.

[BL15] T. Berners-Lee. Linked Data. Personal Note, http://www.w3.org/DesignIssues/LinkedData.html, Retrieved 2015-06-17, 2015.

[BLHL01] T. Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific America*, 2001.

[BLK06] Paul F. Boulos, Kevin E. Lansey, and Bryan W. Karney. *Comprehensive water distribution systems analysis handbook for engineers and planners.* American Water Works Association, 2006.

[BLPF06] Jos de Bruijn, Holger Lausen, Axel Polleres, and Dieter Fensel. The web service modeling language WSML: An overview. In Y.Sure and J. Domingue (Eds.): ESWC 2006, LNCS 4011. pages 590–604, 2006.

[BMC03] Sean Bechhofer, Ralf Möller, and Peter Crowther. The DIG Description Logic Interface. *Description Logics*, 81, 2003.

[BPB12] Bernardo Cuenca Grau, Peter Patel-Schneider, and Boris Motik. OWL 2 Web Ontology Language Direct Semantics (Second Edition). W3C Recommendation, 2012.

[Bra12] Tim Bray. WS-Pagecount. Personal Note, http://www.tbray.org/ongoing/When/200x/2004/09/21/WS-Research, retrieved 2015-06-12, 2012.

[Bri15] Lukas Brinkmeyer. *Modellierung und Lösung des Ersatzerneuerungsproblemes in Wasserversorgungsnetzen unter AIMMS/BARON. Bachelorarbeit.* Universität Paderborn, Paderborn, 2015.

[BW08] Thadthong Bhrammanee and Vilas Wuwongse. ODDM: A framework for modelbases. *Decision Support Systems*, 44(3):689–709, 2008.

[CHJK02] Ivica Crnkovic, Brahim Hnich, Torsten Jonsson, and Zeynep Kiziltan. Specification, implementation, and deployment of components. *Communications of the ACM*, 45(10):35–40, 2002.

[CHRR04] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Version 3.0.2: UDDI Spec Technical Committee Draft, Dated 20041019, 2004.

[CK90] Srikanth Chari and Ramayya Krishnan. Towards a logical reconstruction of structured modeling. In *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, volume 3, pages 524–533, 1990.

[CMM98] Joseph Czyzyk, Michael P. Mesnier, and Jorge J. Moré. The NEOS server. *Computing in Science & Engineering*, (3):68–75, 1998.

[CRC11] CRC 901. CRC 901 On-The-Fly Computing website. https://sfb901.uni-paderborn.de/, retrieved 2015-06-16, 2011.

[Cro36] Hardy Cross. Analysis of flow in networks of conduits or conductors. *University of Illinois*, 1936.

[DBP15] DBPedia. DBPedia project page. http://wiki.dbpedia.org/, retrieved 2015-06-17, 2015.

[DCT+11] Kathrin Dentler, Ronald Cornet, Anette ten Teije, Nicolette de Keizer, et al. Comparison of reasoners for large ontologies in the OWL 2 EL profile. 2011.

[DFG+08] Elizabeth D. Dolan, Robert Fourer, Jean-Pierre Goux, Todd S. Munson, and Jason Sarich. Kestrel: An Interface from Optimization Modeling Systems to the NEOS Server. *INFORMS Journal on Computing*, 20(4):525–538, 2008.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, pages 662–673, 2002.

[DMR14] David Wood, Markus Lanthaler, and Richard Cyganiak. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 2014.

[dR04] da Conceicao Cunha, Maria and Luisa Ribeiro. Tabu search algorithms for water network optimization. *European Journal of Operational Research*, 157(3):746–758, 2004.

[DSM96] Graeme C. Dandy, Angus R. Simpson, and Laurence J. Murphy. An Improved Genetic Algorithm for Pipe Network Optimization. *Water Resources Research*, 32(2):449–458, 1996.

[EGD13] Omar El-Gayar and Amit Deokar. A semantic service-oriented architecture for distributed model management systems. *Decision Support Systems*, 55(1):374–384, 2013.

[EGT07] Omar El-Gayar and Kanchana Tandekar. An XML-based schema definition for model sharing and reuse in a distributed environment. *Decision Support Systems*, (43):791–808, 2007.

[EM03] Obi C. Ezechukwu and Istvan Maros. OOF: Open Optimization Framework. *Imperial College London, Department of Computing, Departmental Technical Report*, (7):1–49, 2003.

[Erl05] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[EV+04] Jérôme Euzenat, Petko Valtchev, et al. Similarity-based ontology alignment in OWL-lite. In *ECAI*, volume 16, page 333, 2004.

[FG14] Fabien Gandon and Guus Schreiber. RDF 1.1 XML Syntax. W3C Recommendation, 2014.

[FGK03] Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A modeling language for mathematical programming.* Thomson Brooks Cole, Pacific Grove and Calif, 2. edition, 2003.

[FIC16] FICO. FICO Xpress Optimization Suite website. http://www.fico.com/en/products/fico-xpress-optimization-suite#overview, retrieved 2016-01-19, 2016.

[FMM10a] Robert Fourer, Jun Ma, and Kipp Martin. Optimization Services: A Framework for Distributed Optimization. *Operations Research*, 58(6):1624–1636, 2010.

[FMM10b] Robert Fourer, Jun Ma, and Kipp Martin. OSiL: An instance language for optimization. *Computational Optimization and Applications*, 45(1):181–203, 2010.

[GE14] Gavin Carothers and Eric Prud'hommeaux. RDF 1.1 Turtle. W3C Recommendation, 2014.

[Geo87] Arthur M. Geoffrion. An Introduction to Structured Modeling. *Management Science*, 33(5):547–588, 1987.

[Geo89] Arthur M. Geoffrion. Reusing structured models via model integration. In *System Sciences, 1989. Vol. III: Decision Support and Knowledge Based Systems Track, Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, volume 3, pages 601–611, 1989.

[Geo92a] Arthur M. Geoffrion. Indexing in Modeling Languages for Mathematical Programming. *Management Science*, 38(3):325–344, 1992.

[Geo92b] Arthur M. Geoffrion. The SML language for structured modeling: Levels 1 and 2. *Operations Research*, 40(1):38–57, 1992.

[Geo92c] Arthur M. Geoffrion. The SML language for structured modeling: Levels 3 and 4. *Operations Research*, 40(1):58, 1992.

[GGMO03] Aldo Gangemi, Nicola Guarino, Claudio Masolo, and Alessandro Oltramari. Sweetening wordnet with dolce. *AI magazine*, 24(3):13, 2003.

[GK05] Michael Gruninger and Joseph B. Kopena. Semantic integration through invariants. *AI magazine*, 26(1):11, 2005.

[Gru93] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gru03] M. Gruninger. A guide to the ontology of the Process Specification Language. In “Hand-book on Ontologies in Information Systems”, R. Studer and S. Staab Eds, 2003.

[GS95] Marco Gagliardi and Cosimo Spera. Toward a formal theory of model integration. *Annals of Operations Research*, 58(6):403–440, 1995.

[GS97] Marco Gagliardi and Cosimo Spera. BLOOMS: A prototype modeling language with object oriented features. *Decision Support Systems*, 19(1):1–21, 1997.

[Gur15] Gurobi Optimization. Gurobi Optimizer Reference Manual: Version 6.5, 2015.

[Hal15] Corinna Hallmann. *Optimierung von Wasserbehältern in einem Wasserversorgungssystem mittels einer Kombination aus Netzreduktion, mathematischer Optimierung und hydraulischer Simulation.* PhD thesis, Universität Paderborn, Paderborn, 2015.

[HB04] Hugo Haas and Allen Brown. Web Services Glossary. W3C Working Group Note, http://www.w3.org/TR/ws-gloss/, 2004-02-11/2004.

[Hen08] P. Hensel. *Optimierung großer Gas- und Wasserversorgungsnetze: Vergleich der Gemischt-Ganzzahligen Linearen Optimierung mit Genetischen Algorithmen: Diplomarbeit.* Freie Universität Berlin, Berlin, 2008.

[HHMW12] Volker Haarslev, Kay Hidde, Ralf Möller, and Michael Wessel. The RacerPro knowledge representation and reasoning system. *Semantic Web*, 3(3):267–277, 2012.

[HKRS08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, and York Sure. *Semantic Web: Grundlagen.* eXamen.press. Springer-Verlag Berlin Heidelberg, Berlin and Heidelberg, 1. edition, 2008.

[HLWW12] William E. Hart, Carl Laird, Jean-Paul Watson, and David L. Woodruf. *Pyomo – Optimization modeling in Python*, volume 67 of *Springer optimization and its applications.* Springer, New York, 2012.

[Hor11a] Matthew Horridge. A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools Edition 1.3, 2011.

[Hor11b] Matthew Horridge. Protege OWL Tutorial P4_v1_3, 2011-03-24/2011.

[HPSB$^+$04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission, 2004-05-21/2004.

[HWW11] William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011.

[IBM15] IBM. IBM ILOG CPLEX Optimization Studio webpage, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, retrieved 2015-11-07, 2015.

[Inn07] Innovation 24. LocalSolver 5.5 Documentation. http://www.localsolver.com/documentation/index.html, retrieved 2015-11-07.

[JD13] Jeen Broekstra and Dave Beckett. SPARQL Query Results XML Format (Second Edition). W3C Recommendation, 2013.

[JNM$^+$07] Jean-Jacques Moreau, Noah Mendelsohn, Martin Gudgin, Marc Hadley, Henrik Frystyk Nielsen, Yves Lafon, and Anish Karmarkar. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C Recommendation, 2007.

[KC00] R. Krishnan and K. Chari. Model Management: Survey, Future Research Directions and a Bibliography. *The Interactive Transactions of OR/MS*, 3(1), 2000.

[KCLH08] Rosemarie Karger, Klaus Cord-Landwehr, and Frank Hoffmann. *Wasserversorgung.* Springer, 2008.

[KS89] Avner Kessler and Uri Shamir. Analysis of the linear programming gradient method for optimal design of water supply networks. *Water Resources Research*, 25(7):1469–1480, 1989.

[KTSW08] Dominik Kuropka, Peter Tröger, Steffen Staab, and Mathias Weske, editors. *Semantic Service Provisioning.* Springer, Berlin, ©2008.

[KW08] Dominik Kuropka and Mathias Weske. Implementing a semantic service provision platform. *Wirtschaftsinformatik*, 50(1):16–24, 2008.

[LF07] Holger Lausen and Joel Farrell. Semantic annotations for WSDL and XML schema. *W3C recommendation, W3C*, 2007.

[LIN07] LINDO Systems Inc. Lindo website. http://www.lindo.com/, retrieved 2015-11-07.

[LPW14] Christopher Lovelock, Paul G. Patterson, and Jochen Wirtz. *Services Marketing*. Pearson Australia, 2014.

[LW07] Kung-Kiu Lau and Zheng Wang. Software component models. *Software Engineering, IEEE Transactions on*, 33(10):709–724, 2007.

[Mad07] Therani Madhusudan. A web services framework for distributed model management. *Information Systems Frontiers*, 9(1):9–27, 2007.

[MB13] Felix Mohr and Hans Kleine Büning. Semi-Automated Software Composition Through Generated Components. Proceedings of the 15th. International Conference on Information Integration and Web-based Applications and Services, iiWAS2013, 2013.

[McC14] Bruce A. McCarl. McCarl GAMS User Guide: Version 24.0, 2014.

[MH09] Ralf Möller and Volker Haarslev. Tableau-based reasoning. In *Handbook on Ontologies*, pages 509–528. Springer, 2009.

[Mic12] Michael Schneider. OWL 2 Web Ontology Language RDF-Based Semantics (Second Edition). W3C Recommendation, 2012.

[MMS96] Pai-Chun Ma, Frederic H. Murphy, and Edward A. Stohr. An Implementation of LPFORM. *INFORMS Journal on Computing*, 8(4):383–401, 1996.

[MN05] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005.

[Mot06] Boris Motik. *Reasoning in description logics using resolution and deductive databases.* PhD thesis, Universität Friedericiana zu Karlsruhe, Karlsruhe, 2006.

[Mot09] Boris Motik. Resolution-Based Reasoning for Ontologies. In *Handbook on Ontologies*, pages 529–550. Springer, 2009.

[MP12] Matthew Horridge and Peter Patel-Schneider. OWL 2 Web Ontology Language Manchester Syntax (Second Edition). W3C Working Group Note, 2012.

[MPPS09] Boris Motik, Bijan Parsia, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language XML serialization. W3C Recommendation. *W3C Recommendation, W3C–World Wide Web Consortium*, 2009.

[MSM92] Frederic H. Murphy, Edward A. Stohr, and Pai-Chun Ma. Composition Rules For Building Linear Programming Models From Component Models. *Management Science*, 38(7):948–963, 1992.

[MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *Web Semantics: science, services and agents on the World Wide Web*, 3(1):41–60, 2005.

[MSZ01] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE intelligent systems*, (2):46–53, 2001.

[MU06] Therani Madhusudan and Naveen Uttamsingh. A declarative approach to composing web services in dynamic environments. *Decision Support Systems*, 41(2):325–357, 2006.

[N. 15] N. V. Sahinidis. BARON User's Manual v. 15.2, 2015.

[Noy09] Natalya F. Noy. Ontology Mapping. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, International Handbooks on Information Systems, pages 573–590. Springer Berlin Heidelberg, 2009.

[NP01] Ian Niles and Adam Pease. Towards a standard upper ontology. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 2–9, 2001.

[OAS12] OASIS. Reference Architecture Foundation for Service Oriented Architecture Version 1.0: OASIS Committee Specification 01, 2012-12-04/2012.

[Pad00] Manfred Padberg. Approximating separable nonlinear functions via mixed zero-one programs. *Operations Research Letters*, 27(1):1–5, 2000.

[PBB12] Peter Patel-Schneider, Boris Motik, and Bijan Parsia. OWL 2 Web Ontology Language XML Serialization (Second Edition), 2012.

[PL04] Terry R. Payne and Ora Lassila. Semantic web services. *IEEE intelligent systems*, 19(1):14–15, 2004.

[PLA$^+$08] Pierre Bonami, Lorenz T. Biegler, Andrew R. Conn, Gérard Cornuéjols, Ignacio E. Grossmann, Carl D. Laird, Jon Lee, Andrea Lodi, François Margot, Nicolas Sawaya, and Andreas Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5(2):186–204, 2008.

[PMW92] P. Piela, R. McKelvey, and A. Westerberg. An introduction to ASCEND: Its language and interactive environment. *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, 3:449, 1992.

[PP14] Patrick Hayes and Peter Patel-Schneider. RDF 1.1 Semantics. W3C Recommendation, 2014.

[PTRC07] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[RB13] G.H.M. Roelofs and J. J. Bisschop. AIMMS - The Language Reference: AIMMS 3.13, 2013.

[RK03] Roland T. Rust and P. K. Kannan. E-service: a new paradigm for business in the electronic environment. *Communications of the ACM*, 46(6):36–42, 2003.

[Ros00] Lewis A. Rossman. EPANET 2: users manual. 2000.

[RR08] Leonard Richardson and Sam Ruby. *RESTful web services.* O'Reilly Media, Inc, 2008.

[SAJR07] Sanjiva Weerawarana, Arthur Ryman, Jean-Jacques Moreau, and Roberto Chinnici. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C Recommendation, 2007.

[SE13] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):158–176, 2013.

[SMH08] Rob Shearer, Boris Motik, and Ian Horrocks. HermiT: A Highly-Efficient OWL Reasoner. In *OWLED*, volume 432, 2008.

[Som12] Ian Sommerville. *Software Engineering.* Pearson Studium - IT. Pearson, München, 9. edition, 2012.

[SP07] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *OWLED*, volume 258, 2007.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, 5(2):51–53, 2007.

[SSL01] Hanif D. Sherali, Shivaram Subramanian, and G. V. Loganathan. Effective relaxations and partitioning schemes for solving water distribution network design problems to global optimality. *Journal of Global Optimization*, 19(1):1–26, 2001.

[Szy02] Clemens Szyperski. *Component software: beyond object-oriented programming*. Pearson Education, 2002.

[TBB03] M. Turner, D. Budgen, and P. Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.

[TD99] T. Maschler and D. A. Savic. Simplification of Water Supply Network Models Through Linearization. Technical Report, 1999.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Automated reasoning*, pages 292–297. Springer, 2006.

[The15] What Is SOA? The Open Group Website, retrieved 2015-06-10, 2015.

[TP88] E. Todini and S. Pilati. A gradient algorithm for the analysis of pipe networks. In *Computer applications in water supply: vol. 1—systems analysis and simulation*, pages 1–20, 1988.

[Tsa98] Yao-Chuan Tsai. Model integration using SML. *Decision Support Systems*, 22(4):355–377, 1998.

[VM07] Patrick Valente and Gautam Mitra. The evolution of web-based optimisation: From ASP to e-Services. *Decision Support Systems*, 43(4):1096–1116, 2007.

[W^+12] World Wide Web Consortium et al. OWL 2 web ontology language document overview. W3C Recommendation, http://www.w3.org/TR/owl2-overview/. Retrieved 2016-01-14, 2012-12-11/2012.

[W^+13] W3C SPARQL Working Group et al. SPARQL 1.1 Overview. W3C Recommendation, 2013-03-21/2013.

[W3C15a] W3C. W3C Data Activity: Building the Web of Data. Retrieved 2015-06-16, 2015-06-16/2015.

[W3C15b] W3C. RDF Current Status. http://www.w3.org/standards/techs/rdf#w3c_all, retrieved 2015-06-17, 2015-06-17/2015.

[WB06] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006.

[Wik16] Wikimedia Commons. Public Ontology Visualization Example. https://de.wikipedia.org/wiki/Ontologie_%28Informatik%29, retrieved 2016-01-14, 2016.

[WK03] Jos B. Warmer and Anneke G. Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.

[WVM+11] Mark D. Wilkinson, Benjamin P. Vandervalk, E. Luke McCarthy, et al. The Semantic Automated Discovery and Integration (SADI) Web service Design-Pattern, API and Reference Implementation. *J. Biomedical Semantics*, 2(8), 2011.

[Zus07] Zuse Institut Berlin. SCIP Optimization Suite Documentation. http://scip.zib.de/doc/html/, retrieved 2015-11-07.

Appendix A.

Common Shortcuts

A definition of common shortcuts will be given in this part of the appendix. Shortcuts will be listed with their full names and references where appropriate.

AIMMS *Advanced Integrated Multidimensional Modeling Software*, see, 3.3.2

AML Shortcut for *Algebraic Modeling Language*, see, 3.3.2

AMPL *A Mathematical Programming Language*, see, 3.3.2

ASCEND *Advanced System for Computations in Engineering Design*, see, 3.2.2

BARON *Branch-And-Reduce-Optimization Navigator*, see, 3.3.1

BLOOMS *Basic Language Object Oriented for Modeling Systems*, see, 3.2.2

Bonmin A MINLP solver, see, 3.3.1

CBSE *Component-Based Software Engineering*, see, 2.1

CDO/cdo Concrete Domain Ontology, see, 5.3

CPLEX Abbreviation for the solver software *IBM ILOG CPLEX Optimization Studio*, see, 3.3.1

DSS/dss *Decision Support System*, see, 3.1

LPFORM A system for formulating linear programs, see, 3.2.1

GAMS *General Algebraic Modeling System*, see, 3.3.2

Gurobi Abbreviation for the solver software *Gurobi Optimizer*, see, 3.3.1

LINGO An AML and a modeling system, see, 3.3.2

LP *Linear Programming*

MDO/mdo *Meta Domain Ontology*, see, 5.3

MINLP *Mixed Integer Nonlinear Programming*

MIP *Mixed Integer Programming*, means only linear problems within this thesis

ODDM *OWL Declarative Description for Modelbases*, see, 3.2.4

OO *Object-orientation*

OOF *Open Optimization Framework*, see, 3.3.3

OS *Optimization Services*, see, 3.3.3

OWL *Web Ontology Language*, see, 2.5.1

RDF *Resource Description Framework*, see, 2.5.1

SaaS *Software as a Service*, see, 2.2

SAWSDL *Semantic Annotations for WSDL and XML Schema*, see, 2.6.1

SM *Structured Modeling*, see, 3.2.2

SML *Structured Modeling Language*, see, 3.2.2

SMML *Structured Modeling Markup Language*, see, 3.2.2

SOA *Service-oriented Architecture*, see, 2.2

SOAP Primarily *Simple Object Access Protocol* but now a proper name, see, 2.3

SPARQL *SPARQL Protocol And RDF Query Language*, see, 2.5.3

SPARQL-DL Query language that is related to SPARQL, see, 2.5.3

SSL Here: *Service Specification Language*, see, 2.6.1

UDDI *Universal Description, Discovery and Integration*, see, 2.3

WSDL *Web Service Description Language*, see, 2.3

WSML *Web Service Modeling Language*, see, 2.6.1

WSMO *Web Service Modeling Ontology*, see, 2.6.1

XML *Extensible Markup Language*

XSD *XML Schema Definition*

XSLT *Extensible Stylesheet Language Transformation*

Appendix B.

Ontology Definitions and Digital Material

This part of the appendix shall provide the definitions of some fundamental ontologies defined within this thesis. Furthermore, the data that is contained in the digital material accompanying this thesis should be explained in short. To that end, the digital material comes in form of a DVD containing the following files and programs:

- Files for all ontology vocabularies defined in this thesis.
- The demonstrator for AML Derivation from Chapter 8 as an archive file without external libraries due to license restrictions. For installation support or a "live demo" you may contact the author of this thesis.
- Ontology representations of the three network flow models, respective derivation implementations and queries. With the aid of these files, the demonstrator can be tested on the network flow case study models.
- An ontology model definition for the water network case study model from Chapter 6. There are no derivation implementations and queries such that an AML model can not be derived with the demonstrator, but the instance model can be tested in an ontology form for freeness of contradictions by reasoning.
- An AMPL XSD for validating the results of AML Derivation.
- The AMPL models of all case study models from the network and water network domains. Furthermore, some modeling excerpts of further examples.
- The renewal planning model from Chapter 7 in the AMPL version together with the data files for the numerical experiments with Bonmin in `.dat`-format.

In order to make a statement on the validity of the delivered tools and data, the following is stated: "All ontologies were checked for freeness of contraditions by using the reasoner HermiT version 1.3.8 in Protégé version 4.3. Queries and derivation implementations have been applied to derive the three network flow models with the demonstrator tool. All AMPL models and excepts were checked for syntactical correctness and static semantics by AMPL via AMPL IDE version 1.0.0. The pipe renewal planning model in AMPL and the data files where not only compiled correctly but also solved with the solver Bonmin in version 1.4.2, as it has been desribed in Chapter 7."

Vocabularies:
In what follows, vocabulary definitions of fundamental ontologies will be given in Manchester Syntax. As the ontology definitions of all vocabuaries and models presented in this thesis would be too big to print, this part of the appendix is reduced to the central top-level vocabularies that make up the ontology optimization framework from a language perspective. To that end, no ontologies for the network and water network domains will be printed here. The latter meta domain and formulation ontologies can be found on the accompanying DVD.

Optimization Modeling Vocabulary
The prefix of this ontology is `OM`. The specifications can be found from Figure B.1 onwards.

Set-Parameter-Indexing Ontology
The prefix of this ontology is `SPI`. The specifications can be found from Figure B.4 onwards.

Mathematical Properties Vocabulary
The prefix of this ontology is `OM.MProps`. The specifications can be found from Figure B.9 onwards.

Technical Constants
The prefix of this ontology is `TechConst`. The specifications can be found in Figure B.13.

Time Horizons
The prefix of this ontology is `TimeHorizons`. The specifications can be found in from Figure B.14 onwards.

Mathematical Operations Parameters
The prefix of this ontology is `MathOpParams`. The specifications can be found from Figure B.16 onwards.

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: sp: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter.owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf−schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl>
Datatype: xsd:string
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    expRequires>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            ExpressionEntity>
    Range:
        Thing
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    requires>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            FormulationEntity>
    Range:
        Thing
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    dataMProp>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            DataEntity>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            DataEntityProperty>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    formulationMProp>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            FormulationEntity>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            FormulationEntityProperty>
```

Figure B.1.: OM Ontology - Part 1

```
DataProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    EName>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            ModelEntity>
    Range:
        xsd:string
Class: Thing
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    Constraint>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            Constraints>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#Goal>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            FormulationEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#Set>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            DataEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    ModelEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    FormulationEntity>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            ModelEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    DataEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    ExpressionEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    DataEntity>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            ModelEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    FormulationEntityProperty>
```

Figure B.2.: OM Ontology - Part 2

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    Parameter>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            DataEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#Variable
    >
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            DataEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    Constraints>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            FormulationEntity>
```

Figure B.3.: OM Ontology - Part 3

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf−schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl>
Datatype: xsd:boolean
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#SetIndexedBy>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SetOfSets>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#IndexingIndice>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#IndiceForSet>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#IndexingIndice>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleSet>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#hasOperand>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#UnionSet>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#hasIndexingElement>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Indexing>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#IndexingElement>
```

Figure B.4.: Set-Parameter-Indexing - Part 1

```
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#hasIndexSet>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#IndexingElement>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#definedOn>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#ParameterCollection>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#TimeSetgivenBy>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#TimeSet>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleParameter>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#WithinCartProductTwoSame>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set
    −Parameter−Indexing.owl#Within>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
    Range:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
```

Figure B.5.: Set-Parameter-Indexing - Part 2

```
DataProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#NonInstantiable>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPIEntity>
    Range:
        xsd:boolean
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#OrderedParameterCollection>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#definedOn> only <http://www.semanticweb.org/
            florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#
            OrderedParameterCollection>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#ParameterCollection>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SimpleSingleElementIndexing>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Indexing>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#hasIndexingElement> max 0 <http://www.
            semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−Parameter−
            Indexing.owl#IndexingElement>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#UnionSet>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#hasOperand> min 1 <http://www.semanticweb.org
            /florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#Set>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#IndexingIndice>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#IndiceForSet> exactly 1 <http://www.semanticweb
            .org/florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#
            SimpleSet>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>
```

Figure B.6.: Set-Parameter-Indexing - Part 3

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SPITopLevelConcept>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#Set>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleSingleElementIndexing>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPIEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SetOfSets>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SetIndexedBy> exactly 1 <http://www.
            semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−Parameter−
            Indexing.owl#IndexingIndice>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SPIEntity>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SimpleParameter>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#definedOn> max 0 <http://www.semanticweb.org/
            florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#Set>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPIEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#IndexingElement>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#hasIndexSet> min 1 <http://www.semanticweb.org
            /florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#Set>
```

Figure B.7.: Set-Parameter-Indexing - Part 4

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SPIIndexing>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#ParameterCollection>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#definedOn> only <http://www.semanticweb.org/
            florianstapel/ontologies/2014/8/Top−Set−Parameter−Indexing.owl#
            SPITopLevelConcept>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPIEntity>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#Indexing>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SPITopLevelConcept>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#OrderedSet>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleSet>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SimpleSet>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#Set>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#TimeSet>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#TimeSetgivenBy> max 1 <http://www.
            semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−Parameter−
            Indexing.owl#SimpleParameter>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#OrderedSet>
```

Figure B.8.: Set-Parameter-Indexing - Part 5

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf−schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl>
Datatype: xsd:double
DataProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.
    MProps.owl#hasUpperRange>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            RangeProperty>
    Range:
        xsd:double
DataProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.
    MProps.owl#hasLowerRange>
    Domain:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            RangeProperty>
    Range:
        xsd:double
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    DataEntityProperty>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            TopLevelMathProp>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    C2Formulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            C1Formulation>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    FormulationEntityProperty>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            TopLevelMathProp>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    TopLevelMathProp>
```

Figure B.9.: OM.MProps - Part 1

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    GeneralNonLinearFormulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            FormulationEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    IntegerData>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            DataEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    BinaryData>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            IntegerData>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    ContinuousFormulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            FormulationEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    C1Formulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            ContinuousFormulation>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    NonNegativeLowerBound>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            DataEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    LinearFormulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            FormulationEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    RangeProperty>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            DataEntityProperty>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            hasUpperRange> max 1 xsd:double,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            hasLowerRange> max 1 xsd:double
```

Figure B.10.: OM.MProps - Part 2

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    StrictlyPositiveLowerBound>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            DataEntityProperty>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    ConvexFormulation>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            FormulationEntityProperty>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#ContinuousFormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            ContinuousFormulation>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#BinaryPropertyInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            BinaryData>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#IntegralityPropertyInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            IntegerData>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#GeneralNonLinearFormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            GeneralNonLinearFormulation>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#ZeroLowerBoundInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            NonNegativeLowerBound>
```

Figure B.11.: OM.MProps - Part 3

```
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#ConvexFormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            ConvexFormulation>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#StrictlyPositiveLowerBoundInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            StrictlyPositiveLowerBound>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#C2FormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            C2Formulation>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#LinearFormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            LinearFormulation>
Individual: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.
    owl#C1FormulationInd>
    Types:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
            C1Formulation>
```

Figure B.12.: OM.MProps - Part 4

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-
    TechConst.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
    Parameter-Indexing.owl>
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO-TechConst.owl#hasDLQuery>
    SubPropertyOf:
        rdfs:comment
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO-TechConst.owl#hasAMPLXMLDerivation>
    SubPropertyOf:
        rdfs:comment
AnnotationProperty: rdfs:comment
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
    owl#EarthGravitation>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
            owl#hasAMPLXMLDerivation> <http://www.semanticweb.org/florianstapel
            /ontologies/2015/9/TechConst.owl#EarthGravitationXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
            owl#hasDLQuery> <http://www.semanticweb.org/florianstapel/ontologies
            /2015/9/TechConst.owl#EarthGravitationDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
            Parameter-Indexing.owl#SimpleParameter>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
    owl#Pi>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
            owl#hasDLQuery> <http://www.semanticweb.org/florianstapel/ontologies
            /2015/9/TechConst.owl#PiDLQuery>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-TechConst.
            owl#hasAMPLXMLDerivation> <http://www.semanticweb.org/florianstapel
            /ontologies/2015/9/TechConst.owl#PiXSL>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
            Parameter-Indexing.owl#SimpleParameter>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
    Parameter-Indexing.owl#SimpleParameter>
```

Figure B.13.: Technical Constants

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf−schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    TimeHorizons.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl>
AnnotationProperty: rdfs:comment
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO−TimeHorizons.owl#hasAMPLXMLDerivation>
    SubPropertyOf:
        rdfs:comment
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO−TimeHorizons.owl#hasDLQuery>
    SubPropertyOf:
        rdfs:comment
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    dataMProp>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    IntegerData>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    TimeHorizons.owl#PlanningHorizonLength>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            TimeHorizons.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−TimeHorizons.owl#
            PlanningHorizonLengthXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            TimeHorizons.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−TimeHorizons.owl#
            PlanningHorizonLengthDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            dataMProp> exactly 1 <http://www.semanticweb.org/florianstapel/
            ontologies/2015/6/OM.MProps.owl#IntegerData>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleParameter>
```

Figure B.14.: Time Horizons - Part 1

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    TimeHorizons.owl#PeriodNumber>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            TimeHorizons.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−TimeHorizons.owl#
            PeriodNumberXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            TimeHorizons.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−TimeHorizons.owl#
            PeriodNumberDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            dataMProp> exactly 1 <http://www.semanticweb.org/florianstapel/
            ontologies/2015/6/OM.MProps.owl#IntegerData>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleParameter>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SimpleParameter>
```

Figure B.15.: Time Horizons - Part 2

```
Prefix: : <http://www.w3.org/2002/07/owl#>
Prefix: owl: <http://www.w3.org/2002/07/owl#>
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
Prefix: xml: <http://www.w3.org/XML/1998/namespace>
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Ontology: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-
    MathOpParams.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
    Parameter-Indexing.owl>
Import: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl>
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO-MathOpParams.owl#hasAMPLXMLDerivation>
    SubPropertyOf:
        rdfs:comment
AnnotationProperty: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/
    MDO-MathOpParams.owl#hasDLQuery>
    SubPropertyOf:
        rdfs:comment
AnnotationProperty: rdfs:comment
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
    dataMProp>
ObjectProperty: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set
    -Parameter-Indexing.owl#definedOn>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    IntegerData>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-
    MathOpParams.owl#LinearizationGridValues1D>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-
            MathOpParams.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO-MathOpParams.owl#
            LinearizationGridValues1DXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO-
            MathOpParams.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO-MathOpParams.owl#
            LinearizationGridValues1DDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
            Parameter-Indexing.owl#ParameterCollection>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top-Set-
            Parameter-Indexing.owl#definedOn> exactly 1 <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO-MathOpParams.owl#
            LinearizationStepNumber1D>
```

Figure B.16.: Mathematical Operation Parameters - Part 1

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    MathOpParams.owl#LinearizationStepNumber1D>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationStepNumber1DDLQuery>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationStepNumber1DXSL>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            dataMProp> exactly 1 <http://www.semanticweb.org/florianstapel/
            ontologies/2015/6/OM.MProps.owl#StrictlyPositiveLowerBound>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            dataMProp> exactly 1 <http://www.semanticweb.org/florianstapel/
            ontologies/2015/6/OM.MProps.owl#IntegerData>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleParameter>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    MathOpParams.owl#LinearizationStartEndValues>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationStartEndValuesXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationStartEndValuesDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#ParameterCollection>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#ParameterCollection>
```

Figure B.17.: Mathematical Operation Parameters - Part 2

```
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    MathOpParams.owl#AbsoluteValueSquareRootSmoothingAlpha>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            AbsoluteValueSquareRootSmoothingAlphaDLQuery>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            AbsoluteValueSquareRootSmoothingAlphaXSL>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/OM.owl#
            dataMProp> exactly 1 <http://www.semanticweb.org/florianstapel/
            ontologies/2015/6/OM.MProps.owl#StrictlyPositiveLowerBound>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#SimpleParameter>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
    MathOpParams.owl#LinearizationGrid1D>
    Annotations:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasAMPLXMLDerivation> <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationGrid1DXSL>,
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#hasDLQuery> <http://www.semanticweb.org/
            florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationGrid1DDLQuery>
    SubClassOf:
        <http://www.semanticweb.org/florianstapel/ontologies/2015/9/MDO−
            MathOpParams.owl#LinearizationStartEndValues>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#ParameterCollection>,
        <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
            Parameter−Indexing.owl#definedOn> exactly 1 <http://www.semanticweb.
            org/florianstapel/ontologies/2015/9/MDO−MathOpParams.owl#
            LinearizationStepNumber1D>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2015/6/OM.MProps.owl#
    StrictlyPositiveLowerBound>
Class: <http://www.semanticweb.org/florianstapel/ontologies/2014/8/Top−Set−
    Parameter−Indexing.owl#SimpleParameter>
```

Figure B.18.: Mathematical Operation Parameters - Part 3