

SUPPORTING PARTICIPATIVE SYSTEMS DEVELOPMENT BY TASK-ORIENTED REQUIREMENTS ANALYSIS

Reinhard Keil-Slawik Technische Universität Berlin
Institut für Angewandte Informatik
Franklinstr. 28/29, Sekr. FR 5-6
D-1000 Berlin 10, Berlin (West)

Task-Oriented Requirements Analysis (TORA) is a technique for analysing and documenting the requirements when developing software which is to be embedded into human work processes. The aim of TORA is to further understanding of the usage context of the DP system so as to enable the user to assess the quality of his work, and the developer to check if the software is adequate with respect to the working procedures to be supported by the software.

1. Introduction

TORA is part of a methodology called STEPS (Software Technology for Evolutionary Participative Systems development) which is being developed at the Technical University of Berlin. As software engineers, we are primarily interested in elaborating methods and techniques to support software development according to the process-oriented approach described in [1]. The conventional approach to software development is based predominantly on a view of software as an autonomous product on its own with no explicit relationship to the outside world. Since data structures and algorithms can be viewed as mathematical objects, and mathematics is a scientific discipline, a lot of research is being done to extend mathematical formalisms so as to enable them to be used from the beginning of the software development process. According to this view, documents such as the requirements definition should reflect the properties of mathematical objects: they should be complete, consistent and unambiguous.

In fact, the result of the software development process is a product which is reduced to formal aspects. However, the semantics of such a product is not defined by mathematical descriptions. Essential for understanding is participation in the reduction process. To understand, for instance, a mathematical abstraction, one has to know the design alternatives underlying the abstraction.

Participation makes no sense when the participants may only have to agree to, or criticize, a final document or product. They must have the opportunity to influence design decisions at every level relevant to the embedment of the DP system into the work tasks.

By choosing the appropriate notions for the task-oriented requirements analysis, we try to fix the areas where reductions are made in order to render the underlying decisions explicit and transparent. The aim of TORA is to enable the users to influence design decisions at the appropriate level and to further the developer's understanding about what is required to support the users' work tasks, and not to establish a political strategy on how to set up groups and committees for participative systems development.

Before describing TORA in more detail, the basic premises underlying the conventional approach to software development are examined and characterized by reference to the notion of the closed system. In contrast, the approach described in this paper is characterized by the notion of the open system. This distinction serves to illustrate our perspective, which views the development process as evolutionary rather than as a pre-planned sequence of activities, and to elucidate our view of the world, which influences our behaviour as well as our design decisions in the actual development process. The last chapter contrasts our approach with certain techniques for requirements analysis currently in use.

2. Open and closed systems in software development

In the following section the notion of open/closed system is used to characterize different views of software, rather than to denote different kinds of systems. The closed system view assumes that once the topmost functions are formally specified, the whole product can be constructed by specifying the semantics of each function or operation by subfunctions or lower-level operations. This is repeated until, eventually, each (sub-)function or operation is defined by elementary functions or operations, which may be regarded as semantic atoms. Such semantic atoms may be machine-executable operations, such as are provided by a programming language and a compiler; or they may be basic mathematical objects, when using a mathematical specification language. In this view of software as a closed system, it is not feasible to introduce new operations whose semantics are not derived from other existing operations or from the semantic atoms. On the other hand, if the topmost functions are specified, the construction process consists in dividing functions into subfunctions, whereby at each level "implementation details" are added. During the whole process, though, the identity of the system is maintained by the fact that the same functions remain to be realized.

The conventional approach embodies such a closed system view in two respects. Firstly, the phase model advocated by Boehm [2] generally assumes that requirements can be determined and fixed in advance, and that the later phases are only steps in which the initial document, the requirements specification, is successively transformed into a functioning system which, it is hoped, fulfills the specified requirements. Secondly, mathematical specification techniques, which may be employed in one phase, aim at formally defining the semantics of the system functions in a top-down manner, as described at the beginning of this section.

Such a closed system view exhibits several limitations:

1. No explicit relationship is established between the software and the "Lebenswelt" (Habermas) of the users.
2. No theory exists with respect to the application area, such a theory being equated with the system specification (see [3]).
3. Requirements are fixed in advance and cannot be changed; furthermore, it is assumed that they are complete, consistent and unambiguous.
4. Production and use are viewed as two opposite ends of a linear development chain.
5. Meaning is derived exclusively from the specification, without the usage context being taken into account.
6. Software quality is judged only on the basis of software-immanent characteristics, such as run time, number of operations, branches, etc.

To distinguish the notion of closed system as used here from other definitions, such as, for example, that of Maturana, Varela [4] and Bertalanffy [5], the term "productionally closed" is used. This notion emphasizes that closedness refers to the assumption that there is a complete specification of the system to be built and that the whole production process can be carried out only with reference to the specification.

A closer look at the development process shows that the closed system view is not adequate. According to Jones [6] – and this is corroborated by our own experience – there is no software system in practical use which has been successfully developed purely on the basis of a written specification.

Software development is part of an organizational process in the course of which production and decision processes are changed, the working routines and communication processes modified, and the organizational structure possibly redefined. These processes of organizational change are imposed by different interests and views put forward by the people involved in these processes. Furthermore, people's behaviour and their requirements may have changed or will change by the time the system is installed. New requirements emerge, and experience gained in using the system results in new insights and demands.

The same holds for the production of software in its narrow sense. Software and the related documents are modified during the development process; errors are found; some parts may be optimized; design decisions are revised; and the system, or some part of it, may be restructured.

The reasons for, and the motivations leading to, such changes are not usually documented, and cannot be documented in their entirety and with all their relations and mutual dependencies.

Each document or product reflects the history and intermediate development stages only to a very limited extent, or, as Naur [7] puts it: "reestablishing the theory of a program merely from the documentation is strictly impossible". This holds for all documents and products in the system development process: the requirements specification as well as the software system.

If we accept this view, it follows that any document or any piece of software can only be understood with reference to the knowledge of those who have participated in the development process and have experience in using the system. Software and the related documents are not autonomous entities, but are part of continuous processes of social interaction, such as communication, learning, understanding, and adapting to changing needs. In the development process, documents and programs serve as the explicitly formulated common memory of the people involved in that process. As is shown in [8], the relationship between such memory media and the processes they are part of can be characterized by ecological attributes comparable to those in use for describing natural ecological systems. These allow us to distinguish between data-processing machines and information-processing human beings.

Openness of a system means that some attributes or parts of the system are subject to later revisions due to interaction with the environment. The identity of a software system viewed as an open system is maintained by the people involved in the development process. Characterizing this as a process-oriented view, Floyd [9] states: "The software system is productionally open; the actual set of programs at any stage is considered a *version* subject to later revisions and embodying limited and possibly conflicting insights . . ."

Naur [7] gives some impressive examples of software systems whose structure and underlying theory were lost owing to the fact that they were not modified and extended by the same people who had produced them.

Regarding software as an open system means using documents and programs to promote the understanding of users and developers; establishing relations between software functions and the usage context; initiating a sequence of cycles of (re-)design, (re-)implementation and (re-)evaluation; and providing a flexible framework to cope with changing requirements.

With respect to requirements analysis, it follows that design decisions relevant to the embedment of the DP system into the work tasks should be explicitly modelled, and that the results of communication processes with various people or groups of people should be documented as well as their possibly differing views.

3. Task-oriented requirements analysis

3.1. Basic concepts and notions

Requirements analysis is the first step in the software development process. Software engineers tend to look at data-processing aspects with a view to finding out first which previously built software functions can be used in the new application area. Hence, it is a widely accepted practice to equate requirements with DP functions. In contrast, we distinguish between:

- *functional requirements*, describing the desired output to be attained for a given input;
- *performance requirements*, stating the resources available to achieve these functions, i.e. specifying quantities and time constraints;
- *handling requirements*, defining the manner in which the system is to be embedded into the working and communication processes of the users, in particular the user interface;
- *embedding requirements*, comprising the consideration of already existing equipment, organizational regulations for use of the DP system, required document standards, required qualifications and training courses etc.

As part of the requirements analysis, these have to be provided for the next development phase, namely functional analysis. Hence, a clear distinction is made between the requirements definition (a document describing what the users want) and the functional specification (a document describing what the developer will offer as a solution). The following refers to the requirements definition/requirements analysis phase only.

Users and user groups as well as managers, customers and clients are characterized by *functional roles*. This notion serves to relate the existing working tasks and those to be newly defined to the workplaces and interests of the users. According to Nygaard, Handlykken [10], a functional role is defined by a specific task or a set of related tasks which has to be performed by a person or a group of persons. A person may have more than one functional role.

Usually, a task can be performed using different working routines, which may also vary according to personal styles and individual preferences. Working routines co-ordinate activities so as to manipulate objects. For each task, there is a series of possible working routines for achieving the desired result. Hence, we have to take into account that:

- a modelled working routine is only a representative sample of such a class of routines;
- in order to perform the same task, different persons might use different working routines.

For these reasons, what is actually modelled is not what people are really doing or thinking, i.e. the real working routines. More precisely, we actually model the processing states

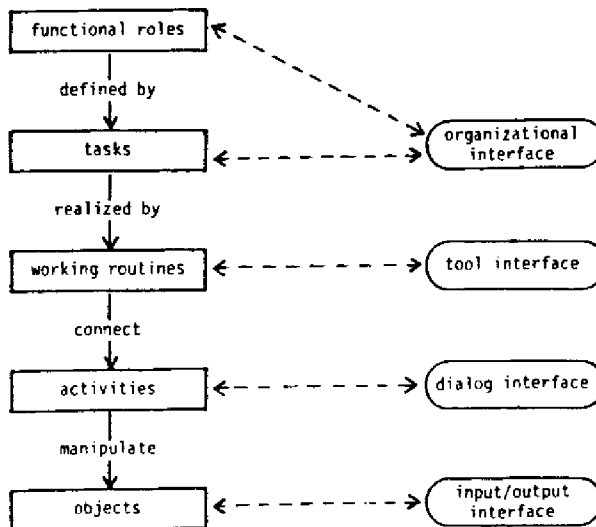


Figure 1. Conceptual levels in TORA

of objects which are created, used and modified. These objects and their processing states are meaningful with respect to the intentions and activities of the people involved. This can be expressed by attributes characterizing the processing state of objects, and by naming and describing the actions by which objects are transformed from one state to another.

The concepts underlying TORA can be related to those of human-computer interfaces. On the left-hand side of figure 1, the conceptual levels in TORA are shown. The right-hand side illustrates connections to the model proposed by Williamson [11] for the European User Environment Subgroup of IFIP WG 6.5. The German National Standard Organisation has adopted this model as the basis for a proposed standard for design criteria in human-computer interaction (see [12]).

The conceptual levels do not imply a top-down development strategy, nor are they supposed to indicate that there is a direct and exclusive relationship between, say, actions and the design of a dialog interface. Hence, the conceptual levels are only used as an analytical tool for assessing the handling requirements.

3.2. The modelling process

The approach embodied in TORA is crucially dependent on the situative context. In this paper, an ideal situation will be described, which serves to illustrate the potential benefits of TORA. This is easier to describe and understand than a real project situation with its numerous exceptions, special premises and constraints.

Requirements analysis can be split into two phases: task analysis, and the description of the computer embedment.

First, functional roles and tasks have to be modelled. Roles do not specify people's behaviour, but it should be emphasized that whenever there is a person to be consulted with respect to requirements, and this person possesses specific skills and qualifications or is

acting in a specific working environment, there emerges the need for establishing a functional role. Thus, for each task in a functional role, working routines are modelled.

Functional roles, working routines, activities and objects are documented by means of activity nets. Activity nets are closely related to a specific variety of petri nets, namely channel agency nets, but are interpreted differently (see figure 2).

Since activity nets grow very rapidly in size and complexity when modelling real working tasks, the petri net refinement mechanism is used to reduce complexity and achieve models which can be drawn on a single page. Thus, activities can be replaced by a subnet. This means that all objects being input to that activity, and all objects – or, more precisely, objects in a certain processing state – being produced by that activity have to be modelled in the subnet. Refinements are indicated by boxes with a black bar on the left side.

Figure 3 illustrates how activity nets are generally used.

When the activity nets with all their refinements have been drawn for each functional role, the interface between different functional roles must be examined. In particular, it is necessary to check:

- whether all objects in a certain processing state being delivered by a functional role are used by other functional roles;
- whether the description of an object being delivered by one role corresponds to the description of this object when being used by another role;
- whether there is appropriate feedback, i.e. whether objects handed over to another role for further processing will be given back in a new processing state or whether, for instance, payments are received when invoices have been sent to a client.

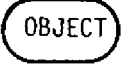

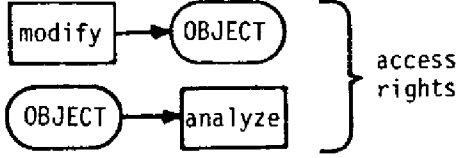
All objects (identifiers written in capital letters) of the activity nets are listed in a dictionary. For each object, the dictionary contains a short description, giving the attributes and, where appropriate, information about the format or structure of the object. An object is a form, handwritten notice, printed material or whatever, subject to modification in the working environment. An object must have a physical appearance, such as letters on a sheet of paper or lines on a video screen. People's thoughts, intentions or wishes are not objects in this sense.

In figure 4, an example of a student accommodation office is given. For each box in the tree structure, an activity net has been drawn. Two examples of these nets are given in figure 5. The tree structure serves merely as the table of contents and is not designed to indicate a top-down development.

In addition to the basic interpretation of activity nets, two further constructs are used. Communication coupling is used to indicate that the processing state of an object is achieved by communication between the functional role performing the task currently being modelled and some other role. The way in which the result is achieved, i.e. the intermediate or possible processing states, is not specified.

A storage facility contains a number of objects which have run along the processing chain at different times. It may be a card index, a file, or any place where several objects are stored.

It is assumed that a storage facility is always maintained by a person or group of persons. This person or group of persons is, on the one hand, responsible for ensuring that the internal structure does not get lost when objects are taken out from, or new objects are put into, the storage facility. On the other hand, a person who has maintained a storage facility throughout several processing cycles is always in possession of more information than can actually be retrieved by inspecting the storage facility. For example, a clerk who

symbols	interpretations
 channel	objects with associated attributes representing the state of modification
 agency	activities, processing and modifying objects
	<p>accomplish a certain processing state of an object (produce, deliver)</p> <p>use of an object in a certain processing state</p>

Basic concepts of channel agency nets.

Furthermore we use the following notations:

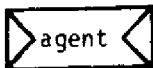
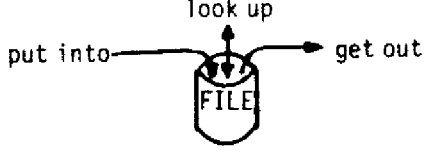
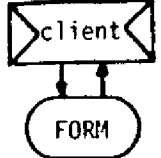
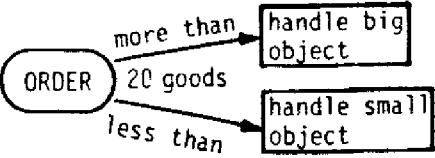
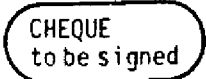
	functional role
	storage with access rights (always local to a functional role)
	communicational coupling
	predicates, inscribed on arrows
	attributes, inscribed on objects

Figure 2. Elements of activity nets

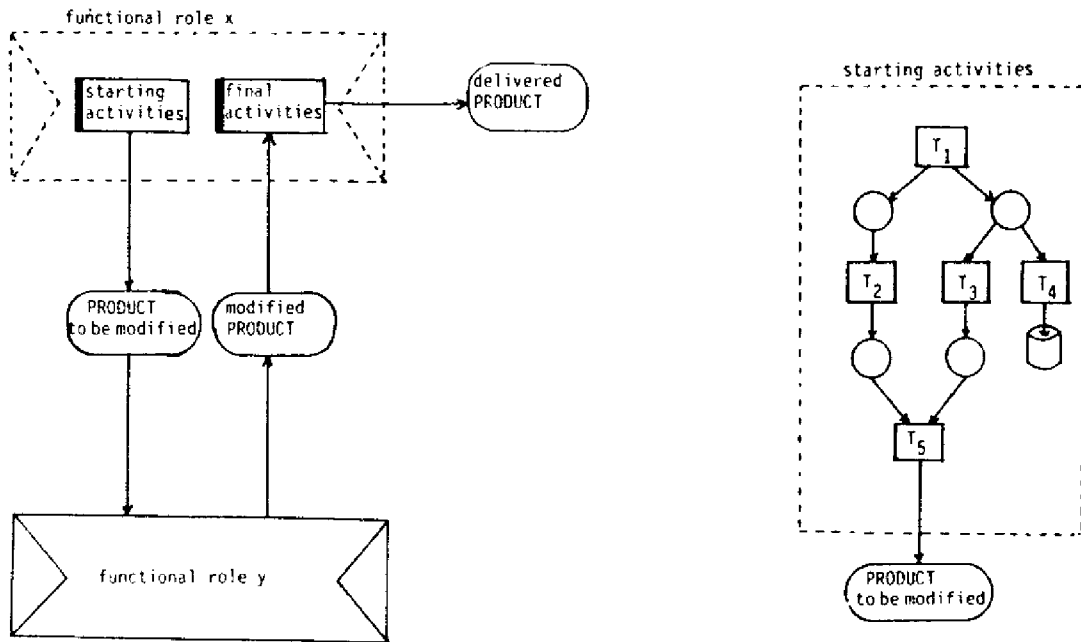


Figure 3. Activity net with refinements

knows whether clients have previously been punctual and correct in paying their bills may grant special credit terms to these clients.

Predicates, written along arrows, allow us to model decisions with respect to alternative working routines. They can also be used to indicate when an activity with no incoming objects has to be performed.

When the task analysis is complete, the embedment of the DP system into the work tasks must be modelled (Figure 6). New activities have to be specified, and details must be given of the activities which are to be automatized or supported by particular DP functions. The DP system is modelled in a similar fashion to the interaction between functional roles. However, functional roles are modelled in detail with the specification of local objects and a local storage facility, whereas the DP system is only characterized by specifying the desired functions and the objects which are exchanged via the dialogue interface. Again, objects and functions are kept in a dictionary. The entries in the functions dictionary contain the specified function name (in capital letters), a short note indicating what the user wants the system to do (including constraints on, or relationships to, other functions), and details with respect to the number of objects to be handled and the frequency with which the system function will be used.

Activity nets describing the embedment of the DP system into the work tasks include only those working routines which are affected or changed. In a new development cycle, when the next version of the system is due to be produced, these activity nets, together with those nets which have not been changed and the updated dictionaries, can be regarded as the result of the first phase of requirements analysis, the task analysis. Conceptually, there is no difference between a new development cycle and an initial task analysis in an organization where a DP system is already in use. In each development cycle, it is necessary to model the new requirements with respect to the embedment of DP functions into the working routines.

Student Accommodation Office

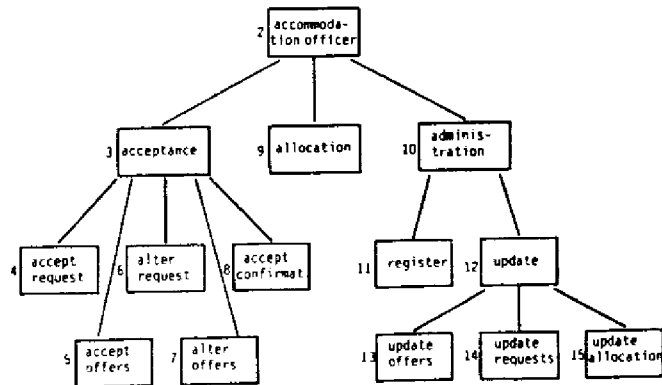
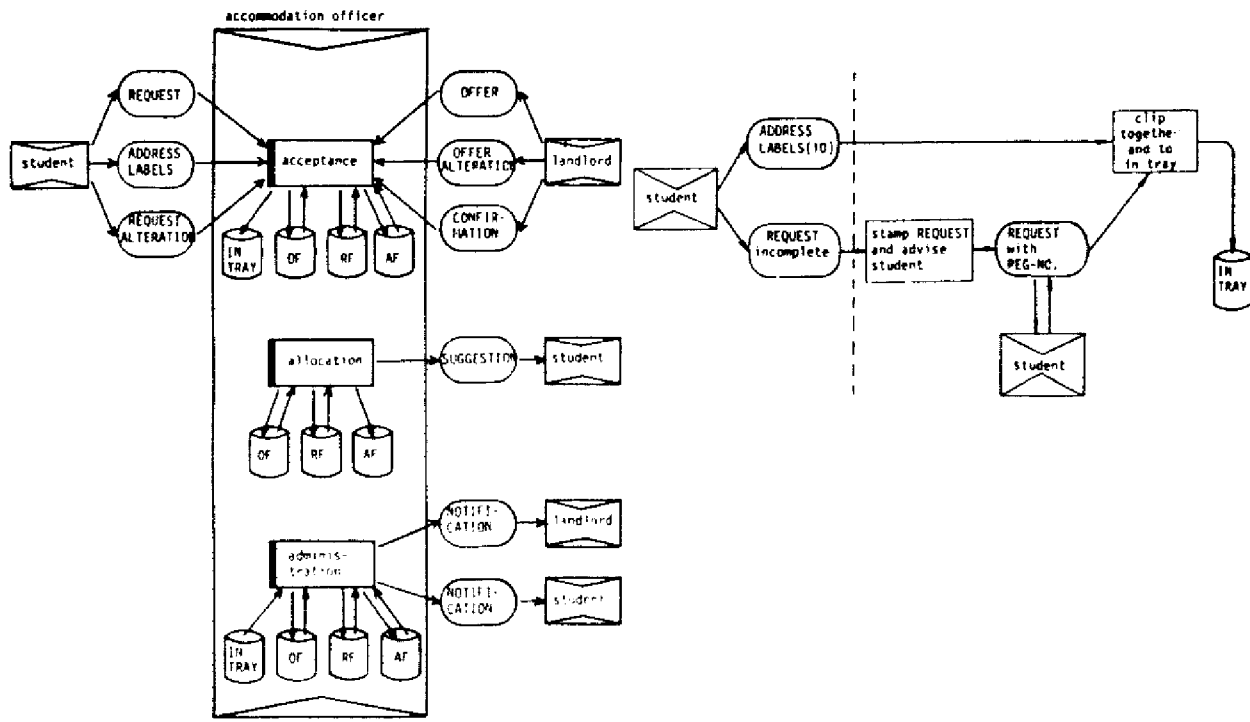


Figure 4. Tree structure of net refinements

functional role: accommodation officer
activity:

functional role: accommodation officer
activity: accept request

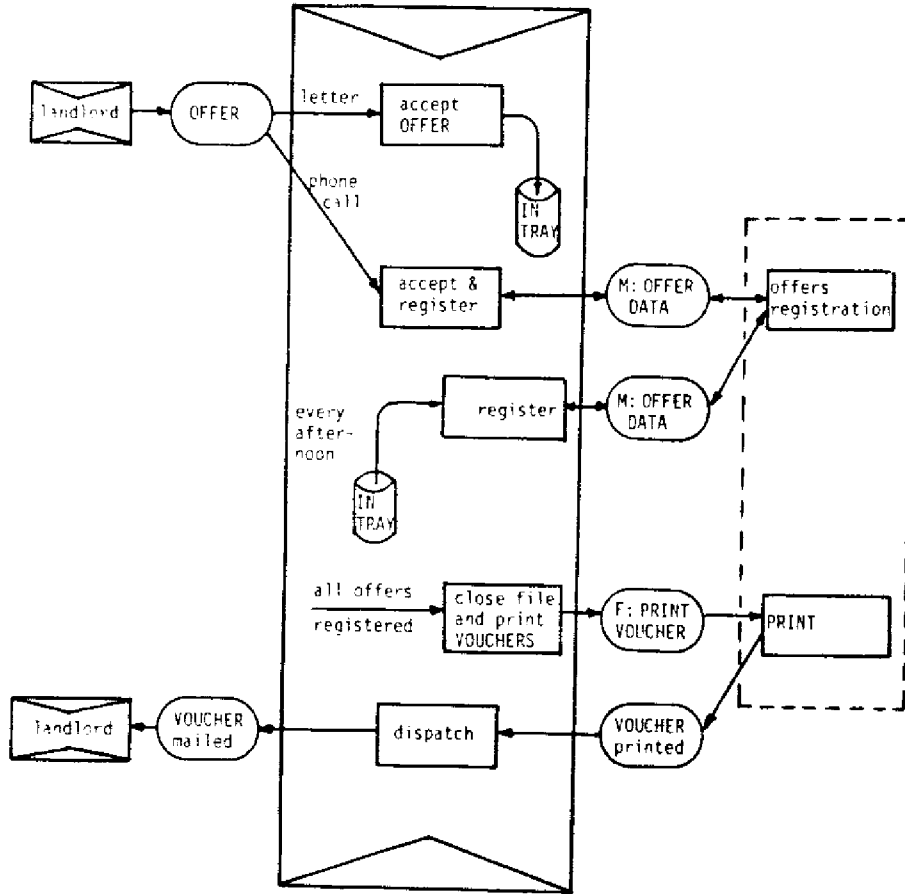


Note: Diagram shows objects interchanged between the accommodation officer and other functional roles.
Abbreviations: OF = OFFERS FILE
RF = REQUESTS FILE
AF = ALLOCATION FILE

Note: Requests are stamped with a registration number consisting of the date and a consecutive number. The registration number determines the order of processing.

Figure 5. Activity nets of the student accommodation office

functional role: accommodation officer
 activity: accept offer



Note: Offers can be registered on line during opening hours 10-12 a.m. Since there are also students coming in the morning who need advice most of the regular activities usually take place in the afternoon.

Figure 6. Embedment of DP functions in the student accommodation office

4. Summary

Task-Oriented Requirements Analysis is essentially based on a view in which tools and techniques are regarded as a means of enhancing people's understanding and furthering communication in a system development process. The development process should be open, in such a way that the people involved in the process have the possibility of influencing design decisions at every relevant level. Furthermore, the tools and techniques being used must be effective. Developers should not experience them as an additional burden hindering the software development process.

TORA is characterized by the following features:

- its strict separation of man and machine;
- a both user-oriented and version-oriented development process;
- its explicit reference to the usage context, i.e. the working routines of the users;
- its use of a flexible modelling tool based on a semi-formal documentation technique;
- the provision of information for technical solutions, in particular the design of the user interface and the development of a logical data model.

An extended view of this sort, necessary for handling embedded systems, may also be achieved by using other techniques for requirements analysis such as SADT, ISAC or various kinds of petri-nets. However, there are certain limitations inherent in these approaches which are briefly outlined below.

First of all, none of these approaches is effectively embedded in a software development methodology (see [13]). SADT can be used to model working tasks by means of actigrams, but there is no mechanism for incorporating data processing aspects; these have to be modelled separately using datagrams. Moreover, a strict top-down approach is advocated. It is not possible to group together activities which are not derived from the same refinement. Different views can be modelled, but there is no additional support such as that provided by the concept of functional roles.

Advocaters of petri-nets tend to focus primarily on the formal aspects of nets. Neither the application area, nor the support of communication and learning processes has substantially influenced the development of these techniques. In [14] and [15], an introduction to, and extensive discussion of, these techniques can be found. Another petri-net interpretation developed by Richter [16] focusses on modelling clerical work. Various points are deserving of criticism here: documents as well as people are modelled as objects being input to an activity; the user as an object is interpreted as an available resource; the net models describe abstract office functions which can be realized by a DP system. It is neither possible to model user-oriented views nor embed the DP system into the working tasks.

An approach closer to that of TORA is ISAC. However, ISAC offers no techniques for supporting a user-oriented description. As has been shown in [17], activity graphs in ISAC -- although very useful for describing information/data structures -- are not a sufficient tool for analysing requirements. Goldkahl/Lyytinen [18] have developed an extension of the ISAC formalism based on the theory of communicative action (Habermas). However, here, too, there is no reference to concepts such as working routines, tasks or functional roles, and the embedment of the DP system is not described.

The difference between TORA and all the other techniques outlined here is that TORA attempts to combine a user-oriented view with a technical approach to the development of software systems. This brings the software developer into contact with methods and techniques which are not primarily designed for developing DP systems, but whose aim is rather to provide qualified jobs and interesting workplaces where people can improve their knowledge and maintain their individual working styles and preferences. Adequacy of the software system with respect to the work tasks of the user has thus become the basic quality criterion.

References

- [1] Floyd, C., Keil, R.: Adapting Software Development for Systems Design With Users; in: *Systems Design for, with and by the User/* ed. Briefs, U., Ciborra, C., Schneider, L.: Amsterdam, North-Holland, 1983

- [2] Boehm, B.W.: Software Engineering, IEEE Transactions on Computers; Vol. C-25, No. 12, 1976
- [3] Turski, W.M.: Informatics. A Propaedeutic View, Warszawa, Amsterdam, New York, Oxford: North-Holland, 1985
- [4] Maturana, H., Varela, F. (eds.): Autopoiesis and Cognition, Reidel, 1980
- [5] Bertalanffy, L. von: General System Theory, New York, George Braziller, 1984
- [6] Jones, C.: A Survey of Programming Design and Specification Techniques, Proceedings, Specification of reliable Software, IEEE Catalog No. 79CH1401-9c, 1979
- [7] Naur, P.: Programming as Theory Building, Microprocessing and Microprogramming, No. 15, 1985
- [8] Keil-Slawik, R.: KOSMOS – Ein Konstruktionsschema zur Modellierung offener Systeme als Hilfsmittel für eine ökologische Orientierung der Softwaretechnik, Dissertation, Technische Universität Berlin, 1985
- [9] Floyd, C.: Outline of a Paradigm Change in Software Engineering, in: Computers and Democracy. A Skandinavian Challenge/ ed. Bjerkness, G., Ehn, P., Kyng, M.: Hampshire, Gower Publishing, 1987
- [10] Nygaard, K., Håndlykken, P.: The System Development Process – Its Setting, Some Problems and Needs for Methods, in: Software Engineering Environments/ ed. Hünke, H., Amsterdam, New York, Oxford, North-Holland, 1981
- [11] Williamson, H.: User Environment Model, in: Report of the 1st Meeting of the European User Environment Subgroup of IFIP WG 6.5., 1981
- [12] Dzida, W.: Ergonomische Normen für die Dialoggestaltung. Wem nützen die Gestaltungsgrundsätze im Entwurf DIN 66234, Teil 8?, in: Software-Ergonomie '85/ ed. Bullinger, H.-J., Stuttgart, Teubner, 1985
- [13] Floyd, C.: A Comparative Evaluation of System Development Methods, in: Information Systems Design Methodologies: Improving the Practice/ ed. Olle, T.W., Sol, H.G., Verijn-Stuart, A.A., Amsterdam, New York, Oxford, North-Holland, 1986
- [14] Wedde, H. (ed.): Adequate Modeling of Systems, Berlin, Heidelberg, New York, Springer, 1983
- [15] Floyd, C.: Design Viewed as a Process. Comments on "Giving back some Freedom to the System Designer" by de Cindio, F., de Michelis, G., Simone, C., Systems Research, Vol. 2, No. 4, 1985
- [16] Richter, G.: Realitätsgetreues Modellieren und modellgetreues Realisieren von Bürogeschehen, in: Informationstechnik und Bürosysteme/ ed. Wißkirchen et al., Stuttgart, Teubner, 1983
- [17] Lyytinen, K.: The Philosophical Nature of Information Requirements and some Research Implications, Syslab Report No. 21, University of Stockholm, 1983
- [18] Goldkuhl, G., Lyytinen, K.: Information Systems Specification as Rule Reconstruction, in: Beyond Productivity: Information Systems Development for Organizational Effectiveness/ ed. Bemelmans, Th.M.A., Amsterdam, New York, Oxford, North-Holland, 1984