Methodologies for Computer System Design
W.K. Giloi and B.D. Shriver (Editors)
Elsevier Science Publishers B.V. (North-Holland)
IFIP, 1985

A MULTILEVEL CYBERNETIC MODEL OF THE DESIGN PROCESS

Franz J. Rammig

Universität-Gesamthochschule-Paderborn

Federal Republic of Germany

This paper presents a conceptual framework for the modelling of design processes. The system analyst gets a kit of models of elementary design activities and a selection of composition rules. By this he can construct an adequate model of the specific design process. Influenzed by I.H.A.E. Amkreutz we start with a feedback loop view of the design process. In contrary to him we introduce various levels of abstraction. Now it is possible to differ between levelinvariant and levelvariant design activities. As compositon rules we investigate the main techniques: "Chaining" (output of one subprocess is used as input of the next one), "Recursion" (subprocess is substituted by entire design process) and "Multilevel Feedbacking" (special design activities are used as interlevel links). All alternatives can easily be expanded to describe composition/decomposition. We now have available the framework to describe classical design strategies.

INTRODUCTION

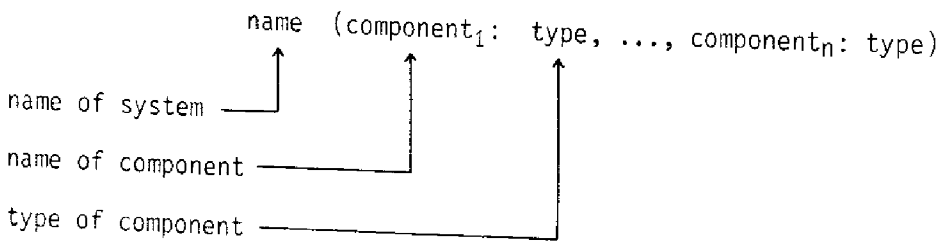
In order to obtain a good CAD solution for a specific application the present design process as well as the desired one have to be analyzed carefully. In a very interesting paper I.H.A.E. Amkreutz proposed a cybernetic model of the design process [AM1]. Unfortunately he did not introduce different levels of abstraction. But without such a multilevel view it is rather difficult to speak about specific design strategies. In addition with the aid of a couple of different levels of abstraction a more detailled classification of specific design activities is possible. In our approach we differ between levelinvariant construction activities like modification, optimization and levelvariant ones like implementation or aggregation. In a similar manner we differ between abstractioninvariant and abstrationvariant feedback activities. Another advantage of the multilevel approach is, that the composition between different design activities within one level or on different levels have to be analyzed. Within this paper we investigate three basic composition rules: "Chaining", "Recursion" and "Multilevel Feedbacking". All schemes may be used within one model in an intermixed manner and they all can easily be generalized to describe composition/decomposition. Now classical design strategies like "Top Down", "Bottom Up", "Yoyo", "Australian Yoyo" together with the common "Divide and conquer" way of doing can be modelled. Heterogenous strategies may be modelled as well.

The investigation of a design process using such modelling tools itself is a "Divide and Conquer" strategy. Usually not the complete design process is covered by a single CAD tool. Having available a good modelling tool enabled us to identify such subactivities which are well suited and worth to be automated. Furthermore it is indicated how the individual tools have to be integrated.

SINGLE LEVEL MODELS

NOTATION

In order to obtain a concise and easy to understand description of our models of design activities, throughout this paper we use a simplified VDM [BJ1] notation.



As types we use:

- objects that are elementary in our context, e.g. integer, language
- lists/pairs, e.g. integer-list, integer-pair
  The first element of a list or a pair K is denoted by first (K) or head (K) , the second element by second (K)
- sets, e.g. language-set
- sequences, e.g. $<\text{language}_i \mid i \in \{1 : n\}>$
- mappings, e.g. integer x integer → boolean
  the domain of mapping f is denoted by dom (f) , its range by map (f) .

Up to now only the structure of a system to be specified is described. The desired features are given in the following notation:

is-wf name (component$_1$, ..., component$_n$) = feature$_1$ ∧...∧ feature$_K$

Features may be defined by reducing to already known features using a self explanatory notation, e.g. (is-odd ∧ is-positive) or may be given using the usual mathematival definition language in a free manner.

EXAMPLE

We want to describe a system, that checks whether the second one of two entered natural numbers is the square root of the first one.

```
square-root-checker (number     : integer,
                     hypothesis : integer,
                     answer     : boolean,
                     test       : number  x  hypothesis → answer
                    )
```

<u>is-wf</u> square-root-checker (n,h,a,t) =

$$\forall n \in \mathbb{N}_0 : \forall h \in \mathbb{Z} : a(n,h) = (n = h * h)$$

STRAIGHT FORWARD GENERATION

A first idea for modelling a design process may be the simple black box approach: A design process is looked at as a black box where tasks are entered. Whenever a task is put into such a design process the process answers with a solution. We will not go into further details but switch immediately to a more dynamic point of view. Usually design processes are not independent from history (at least if persons are involved). Thus, if we interpret a design process as a transformation mapping that maps an object description given in an input language L_i into a description given in an output language L_0, $G : L_i \to L_0$, $G(L_i) = L_0$, we have to take into consideration the fact that this mapping G is parametrized in some kind by the design process's history.

In order to obtain a consistent model we divide the input language L_i into two sublanguages $L_{i,d}$ and $L_{i,c}$. The first one serves as language for an operational description of the object to be modelled, the latter one as language to describe general constraints, either specific to the model or general. Thus via $L_{i,c}$ not only object specific "design rules" may be put in but also the knowledge base of the design process may be altered.

Thus we obtain:

<u>Def. 1</u> (Straight forward design process)

```
    SFDP (inp : language-pair, outp : language-pair
          gen : inp-list → outp-list,
          hist: ℤ → ℕ₀
    )
```

<u>is-wf</u> SFDP (i,o,g,h) =

$$\forall L,L' \in i\text{-}\underline{list} : (\langle L_K \mid K \in \{i - h(i) : i\}\rangle = \langle L'_K \mid K \in \{i - h(i) : i\}\rangle \Rightarrow (h(L))_i = (g(L'))_i$$

Thus, what a straight forward design process puts out at a certain point of time depends on the complete input during a certain time period which may vary from time to time.

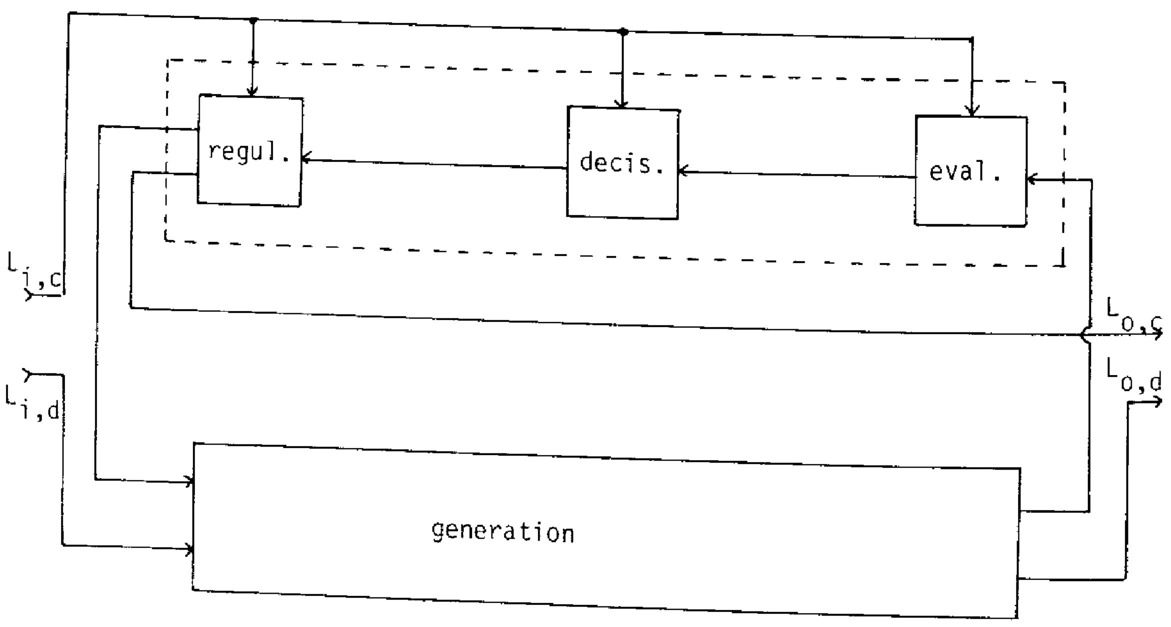
FEEDBACK LOOP GENERATION

We now want to follow an idea of Amkreutz [AM1] in order to obtain a closer look to the inside of a design process. The basic idea is to differ between two basic classes of design activities:

- Generation itself
- Checking activities (verification, validation, evaluation)

These checking activities use the output of the generation activity as input and pass the result back to the generation activity in order to obtain an effect. Therefore the two activities form together a feedback loop. Following this approach now we also can route the different sublanguages within the design process, i.e. identify information channels.

The object we have in mind may be shown as follows:



This model seems to be a good first approach to model a design process. The main classes of activities can be identified, information channels may be routed and the dynamic behaviour can be investigated.

For the latter task we need a more formal model with a variety of tunable parameters.

In our notation such a formal model looks like as follows:

Def. 2 (Feedback loop generation)

```
    FLG (inp      : language-pair,
         outp     : language-pair,
         correct  : language,
         check    : language,
         gen      : (first(inp))-list x correct-list →
                    check-list x (first(outp))-list,
         fdbk     : (second(inp))-list x check-list →
                    correct-list x (second(outp))-list
         hist-g   : ℤ → ℕ₀ ,
         hist-f   : ℤ → ℕ₀
```

```
is-wf  FLG (inlng, outlng, corlng, chklng, gen, fdbk, hg, hf) =
    is-dependent-on-last-hg(i)-inlng-inputs-and-on-actual-corlng-
       input (gen_i)  ∧
    is-dependent-on-last-hf(i)-inlng-inputs-and-on-actual-chklng-
       inp (fdbl_i)
```

The two features above may be formulated in the way like in Def. 1. By gen_i we mean the output of gen at point of time i .

Note that for the two main components different learning capabilities may be possible.

ENHANCED FEEDBACK LOOP GENERATION

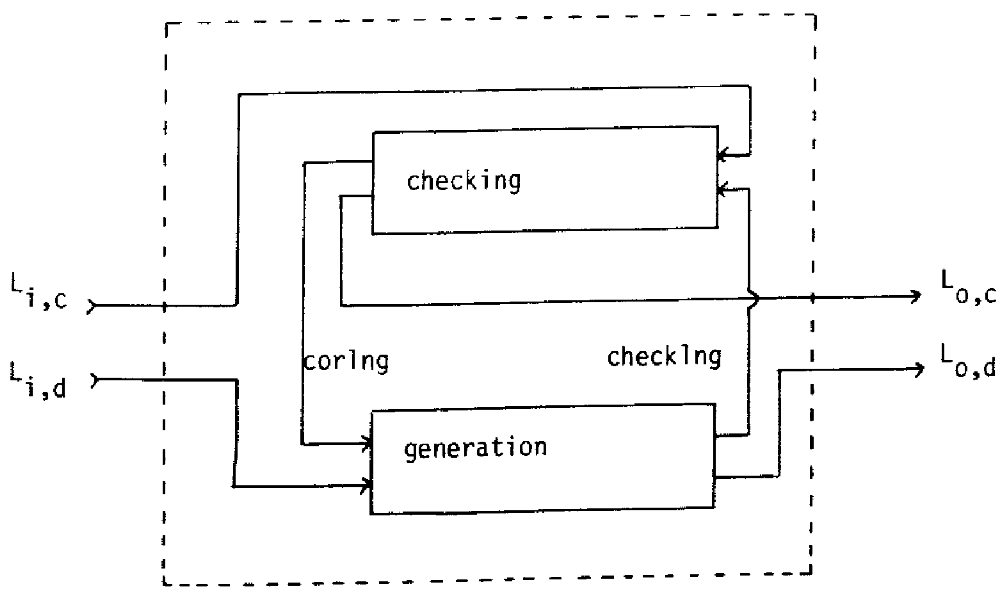
Again following Amkreutz[AM1] we will identify three subobjects within the feedback-object of a design process:

- an evaluation function
- a decision making function
- a regulation function

The evaluation function maps the exogenous input and the output of the generation object into an evaluation result. As exogenous information the evaluation function gets the evaluation criteria and general restrictions. From the generation object it gets the internal design result to be evaluated.

In order to make sure that an evaluation results in the proper consequences further subobjects are necessary. It is the job of the decision making function to calculate the best suited strategy out of the exogenous input and the result of the evaluation function.

Finally the selected strategy have to be transformed into an input into the generation object of the design process. This is done by the regulation function. This discussion leads to a system that may be shown in the following manner:



In order to obtain a formal description we only have to redefine the feedback object of our design process:

Def. 3 (Enhanced feedback object)

```
     EFO (inp  : language-pair,
          outp : language-pair,
          ed   : language,
          dr   : language,
          eval : inp-list → ed-list,
          dec  : ed-list  x  (head(inp))-list → dr-list,
          reg  : dr-list  x  (head(inp))-list → outp-list
         )

   is-wf EFO  (inlng, olng, ed, dr, eval, dec, reg) =
              is-wf FLG-with-fdbk-substituted-by-compatible-EFO
```

In order to avoid unwieldiness within this paper we restrict ourselbes on this relatively vague definition for the wellformedness of enhanced feedback objects.

Up to now we have obtained a model that reflects a top level view of the design process, either as description of the existing situation (when the model's parameters are properly calibrated) or as descriptions of the desired process. In both cases the structure does not necessarily reflect the structure of labour division within the real process (e.g. the different departments envolved) but the different actions to be carried out. In a first step of system analysis using this model the design process should be analized globally and locally. By this the following information per unit to be analized has to be sampled:

- what kind of transormation is carried out by the generation object,
- what are the input and output languages of the generation object,
- what is the knowledge base of this object and how is it updated,
- how far is this object dependent on its history,
- what kind of evaluation is carried out,
- how far is evaluation seperated from generation,
- how is the evaluation object provided with inputs (design to be evaluated and evaluation criteria),
- what is the further processing of evaluation results,
- how are modification strategies calculated,
- how is the decision making object provided with inputs (evaluation result and exogenous parameters),
- in which form the calculated strategies are made available,
- how are modification strategies transformed into input of the generation object,
- how is the regulation object provided with inputs (strategy and exogenous paramenters),
- what is the general mode of operation of the design process, especially what are the expected or observed information rates compared with the processing rate
- how are input and processing synchronised,

- what are the possible main states of the processes, especially what are the conditions for equilibrium.

Note that these questions habe to be answered as well in the case of analyzing an existing design process as well as in order to design the desired design process. It can be observed immediately that up to now we have a model that seems to be valuable for a macroscopic view or as description tool for single actions in a complex design process. The gap to be attacked now can be characterized as microscopic view of the entire system. I.e. we need composition techniques that allows us to integrate single models to a global one.

MULTIPLE LEVELS OF ABSTRACTION

LEVELS OF ABSTRACTION

Usually with respect to a design process one has in mind not a momolithic point of view. One has the imagination of a sequence of design subprocesses at various levels of abstraction. Of course in any field there is no natural or generally adapted partition into levels of abstraction. Taking into consideration the variety of reasons for a partitioning into levels of abstraction one even will not expect to find such a natural partition.

As reason for such a partitioning are worth to be mentioned the following ones:

- Technological facts
  Just the existence of a certain set of elementary building blocks establishes a level of abstraction.

- Engineering facts
  Any engineering methods (algorithm) assume a more or less exactly defined set of objects. Of course there exists a close interaction to technology: On one hand science tends to investigate areas it is confronted with. On the other hand objects that have a safe scientific foundation are preferred candidates for implementation.

- Management facts
  Complex systems can be desinged only in a labour dividing environment. Of course one will choose such a kind of labour division that minimzes frictional losses. Obviously this is not the case if different groups use mutually the other group's outputs as elementary building blocks. Therefore a hierarchy of abstraction levels is preferable. Of course again we have a strong technological impact on organisation structures.

- Traditional facts
  Science, technology and management are (have to be) conservative to a certain amount. Levels of abstraction may be argued for a priori with the aid of tradition.

Just to give a glimpse how a partition into levels of abstraction looks like, we will present a possible partition for the design process for conventional digital computers:

a) Level of marketing

   Action: Decision for a certain product line

b) Conceptual level

   Action: Rough conception of system family

c) Level of programming system

Action: Decision what is the set of programs to be offered: Application programs, compilers, utilities

d) Level of operating system

Action: Definition of: Data management, process management, memory management, resource management, pripheral control

e) Firmware level

Action: Realization of the virtual operating system machine with the aid of a set of microprograms

f) Architectural level

Action: Definition of microinstrustion format, adressing, memory organization, bus organization, peripheral control

g) Register transfer level

Action: Decomposition of the architecture into objects like: Memory, register, ALU, simple controllers

h) Gate level

Action: Logical realization of above building blocks

i) Physical design level

Action: Transofrmation of a logical network either to populated PC-boards or to a VLSI layout

In any case an adequate model of the design process has to reflect the concept of partition into levels of abstraction. In the following, three basic approaches to this goal will be discussed. We will characterize a level of abstraction simply by the set of those languages within a design process which are attached to a common level of abstraction. I.e. within a given design process we assume a totally semi ordered partition on the set of design languages.

Def. 4 (Design language set)

```
DLS (languages: language-set,
     level    : languages → ℕ
    )

is-equivalent ($lng_1$, $lng_2$, dls) =
    is-DLS(dls) ∧ {$lng_1$, $lng_2$} ⊂ languages(dls) ∧ level($lng_1$) = level($lng_2$)

is-upper-neighbour($lng_1$, $lng_2$, dls) =
    not-is-equivalent($lng_1$, $lng_2$, dls) ∧
    not-exist-lng -between-$lng_1$-and-$lng_2$
```

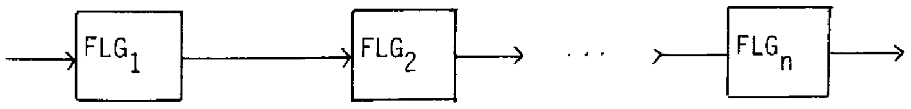
CHAINING

As a first approach it simply may be assumed that the input from the environment is routed to a first design process FLG_1. Further we have to assume that the input language and the output language of FLG_1 are either equivalent or neighboured with respect to the semi ordering of the design languages.

Now the output of FLG_1 may be used as input of FLG_2 for which the same conditions hold and so on.

Classical design strategies like "Top Down" or "Bottom Up" are obtained if we request that the neighbourhood of the input and outputlanguages of all involved FLG_i are in the same direction. Of course levelinvariant design subprocesses may be inserted.

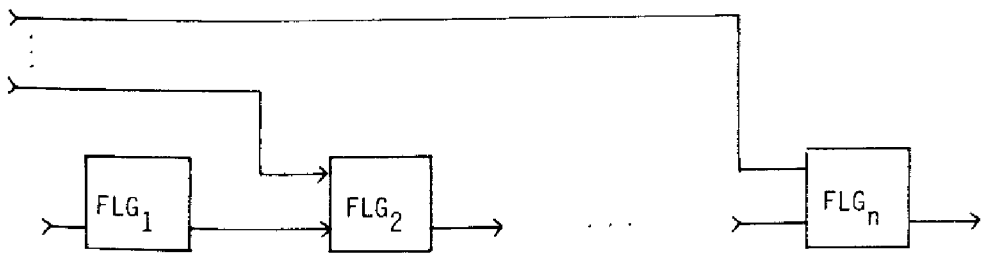
The model obtained may be sketched in the following way:



$$\forall i \in \{2 : n\} : \mathrm{inp}(FLG\ ) = \mathrm{outp}(FLG_{i-1})$$

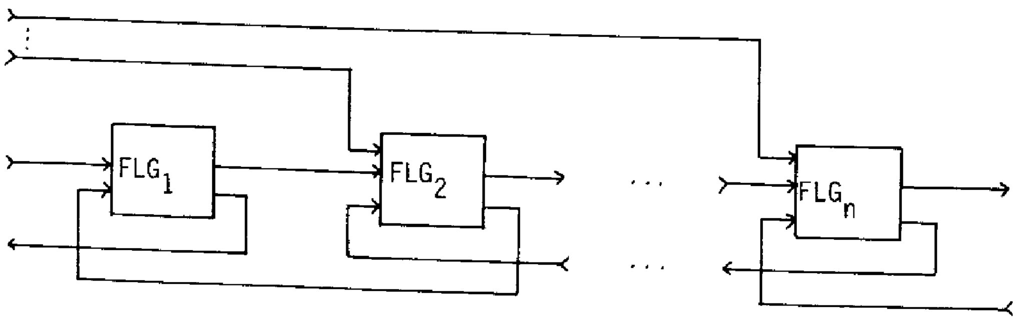
A little closer look at our "Bottom Up" interpretation shows that the model is a little bit to simple. A design process where at a very low level of abstraction some information is put in and then in successive steps a more abstract description is generated makes no sense. Usually both, the foundation and the goal are to put in.

Therefore we obtain a system that may be sketched as follows:



This model seems to be more realistic in the "Bottom Up" as well as in the "Top Down" case. Unfortunately in most cases unidirectional design strategies are not practicable. In the case of "Bottom Up" we must be able to process the information that the design goal cannot be reached on the basis of the modules already designed on lower levels. In the case of "Top Down" we mustbe able to react on the information that, what have veen designed at a higher level can not be implemented at lower levels. Only if such information channels are established the realistic design strategies "Yoyo" (iterated "Top Down" and "Bottom Up" with main direction "Top Down") and "Australian Yoyo" (like "Yoyo" but with main direction "Bottom Up") may be modelled.

What we obtained now is a double chain looking like the following sketch:



That is our first really useful composition scheme for multilevel design processes.

Def. 5 (Double chained design process)

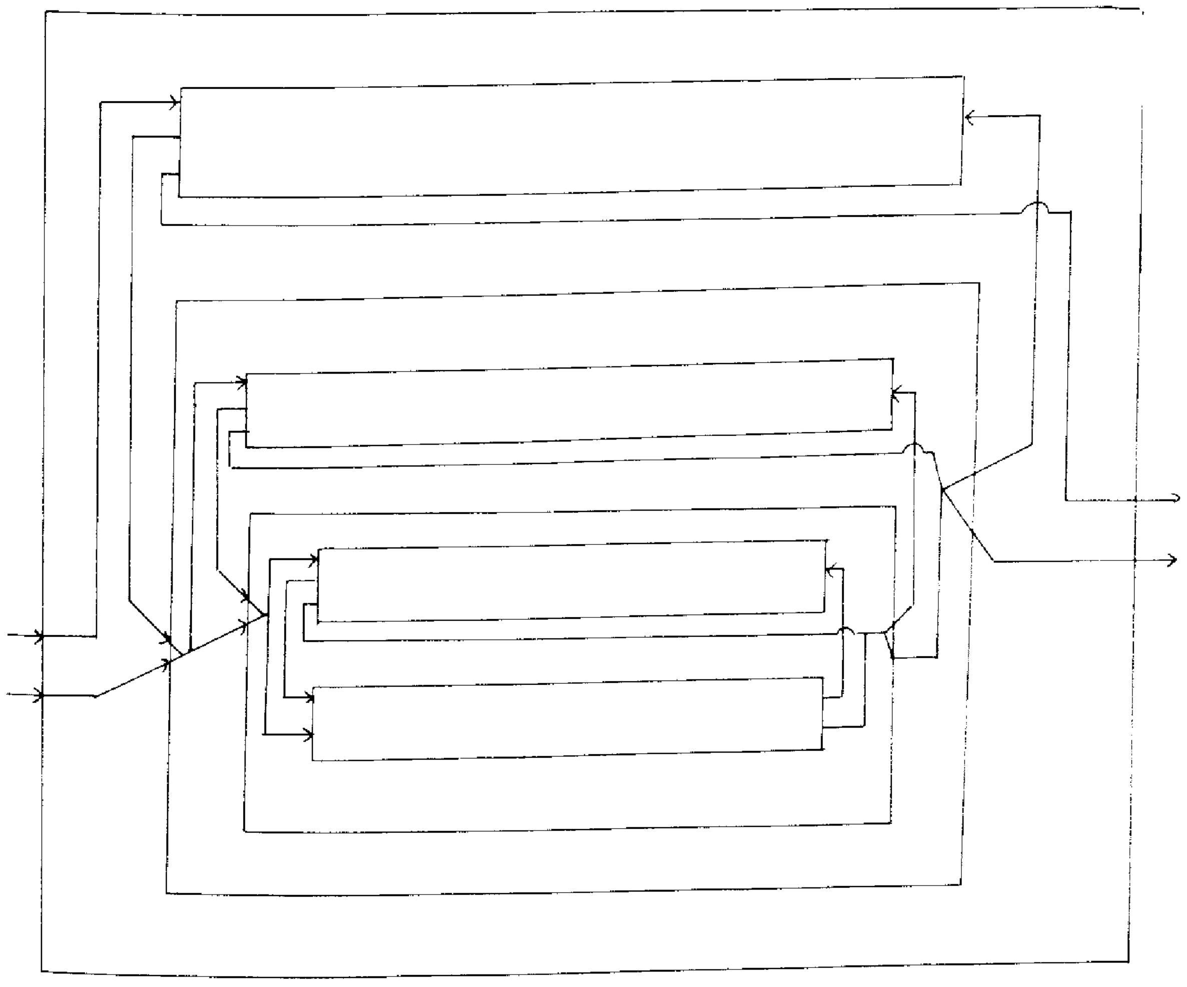
```
    DCDP (flgs      : <FLG_i | i ∈ {1 : levelcnt} >,
          mainlngs  : <language_i | i ∈ {2 : levelcnt + 1} >,
          inlngs    : <language_i | i ∈ {1 : levelcnt} >,
          fdbklngs  : <language_i | i ∈ {1 : levelcnt + 1} >,
          levelcnt  : ℕ
    is-wf DCDP (flg, mlng, ilng, flng, lcnt) =
          is-DLS (all-languages)
          ∀i ∈ 1 : lcnt  : is-equivalent (mlng_i, flng_i, ilng_i, DLS) ∧
          is-equivalent (mlng_lcnt+1 , flng_lcnt+1 , DLS) ∧
          ∀i ∈ {2 : lcnt}  : is-from-flg_i-1-to-flg_i-language(mlg_i)
                             is-from-flg_i-to-flg_i-1-language(flg_i)
                             is-from-evironment-to-flg-language(ilng_i)
          ∀i ∈ {1 : lcnt}  : level(flng_i)-level(flng_i+1) ∈ {0,1}∨
                             level(flng_i)-level(flng_i+1) ∈ {-1,0}
```

Definition 5 has been simplified a little bit in order to make it more readable. It can be observed that there are a couple of parameters that can be filled in when a given or desired design process has to be modelled. The model is very general but unfortunately conceptuable heterogenous. A global model of the FLG type can not be refined into a DCDP type model in a straight forward manner. Our next approach therefore will expecially emphasize model consistence.

RECURSION

From a methodological point of view it is very attractive to model a hierarchy of abstraction just by substituting subobjects of a FLG by a FLG object itself. This is possible with all subobjects: Generation, evaluation, decision making, regulation. This approach is recursive by its nature. Furthermore it is consistent and elegant throughout the process of stepwise refinement of a model of the design process to be investigated.

Unfortunately this model describes only one single design methodology: "Bottom up with predefined design goal". The design task is passed down to the lowest level, is processed there by its generation object (the only one involved!) is checked there and is then pushed up for checking at various levels of abstraction until it reaches the top level again. As an example may serve the following daft of a recursive substituted design process: (The subobject "generation" is substituted twice, the feadback object is not further substituted).



Of course, the formal definition of the recursive substituted design process is very simple:

<u>Def. 6</u> (Recursive substituted design process)

RSDP (flg(,,,, gen, eval, dec, ref,,)) =

<u>is</u>-FLG(flg( )) $\wedge$

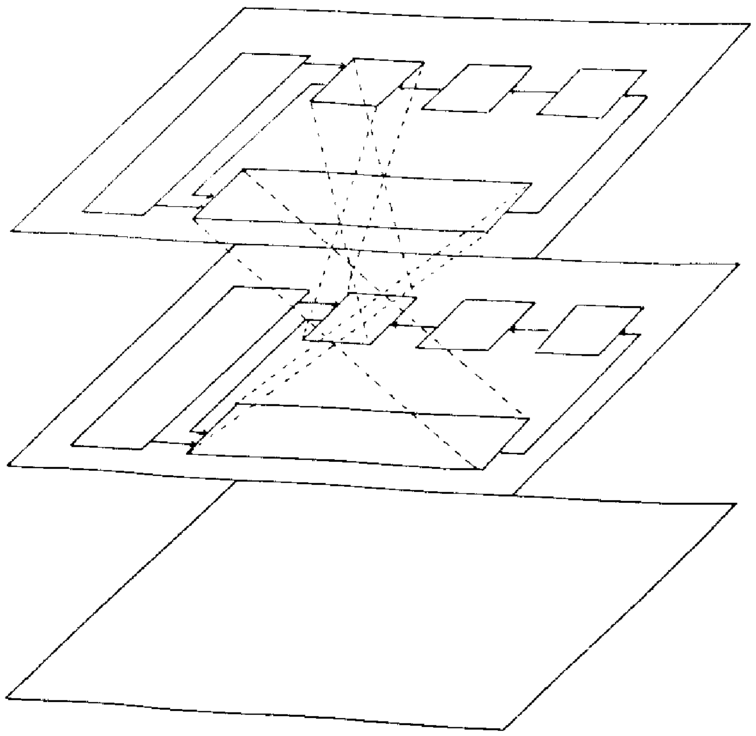
$\exists$obj ϵ {gen, eval, dec, reg}: <u>is</u>-FLG(obj)

MULTILEVEL FEEDBACKING

Up to now we have presented two extreme methods of composition: Recursion as purely endogenous approach and chaining as purely exogenous one. Now a compromise shall be developped that combines the generality of chaining with the consistency and elegance of recursion. The fundamental idea is to identify such subjects within FLG from which there exist abstracting or deabstracting variations. Such objects may be used to couple various levels of abstraction. It should be noted that in this approach active subobject are used as coupling elements while in the other approaches language objects serve for this purpose. First of all the subobjects of FLG have to be investigated in order to identify suitable candidates.

- Generation object
  The most popular variation of generation is a deabstracting one, namely implementation. But also the contrary is very important, we will call it aggregation. As an example may serve the extraction of a program scheme out of a program.
- Evaluation object
  Of course, evaluation is an act of abstraction. But that is no abstraction in the sense discussed within this context. E.g. if the description of an adder is mapped by an evaluation object on "faster than 20 ms" this predicate is no longer a description of an adder. We will assume that every level of abstraction has its own evaluation criteria with the same level of abstraction attached to. Of course then there is no deabstracting nature of evaluation.
- Decision making object
  This object may decide that an abstraction or deabstraction has to be carried out; the object itself remains on one single level however.
- Regulation object
  Here there exist both variations, the abstracting one and the deabstracting one. An example of an abstracting regulation is the message "desired object not implementable" within a "Yoyo" environment while as deabstraction regulation one may mention a message "offered module don't fits into my requirements".

The model obtained may be drawn in the following way:



In order to keep this paper readable we have to restrict ourselves on the structural part within the formal definition of the multilevel feedbacking:

Def. 7 (Multilevel feedbacking)

```
MLG (levels : $\mathbb{N}$ ,
     flgs   : <$FLG_i$ | i $\in$ {1 : levels}> ,
     impls  : <dom (gen($FLG_{i+1}$)) $\rightarrow$
                rng (gen($FLG_i$)) | i $\in$ {1 : levels-1}> ,
     aggrs  : <dom (gen($FLG_i$)) $\rightarrow$
                rng (gen($FLG_{i+1}$)) | i $\in$ {1 : levels-1}> ,
     regdns : <dom (reg($FLG_{i+1}$)) $\rightarrow$
                rng (reg($FLG_i$)) | i $\in$ {1 : levels-1}> ,
     regups : <dom (reg($FLG_i$)) $\rightarrow$
                rng (reg($FLG_{i+1}$)) | i $\in$ {1 : levels-1}>
    )
is-wf MLF ( ) =
     a-couple-of-straight-forward-restrictions-especially-stating-
     dependencies-of-mappings
```

COMPOSITION/DECOMPOSITION

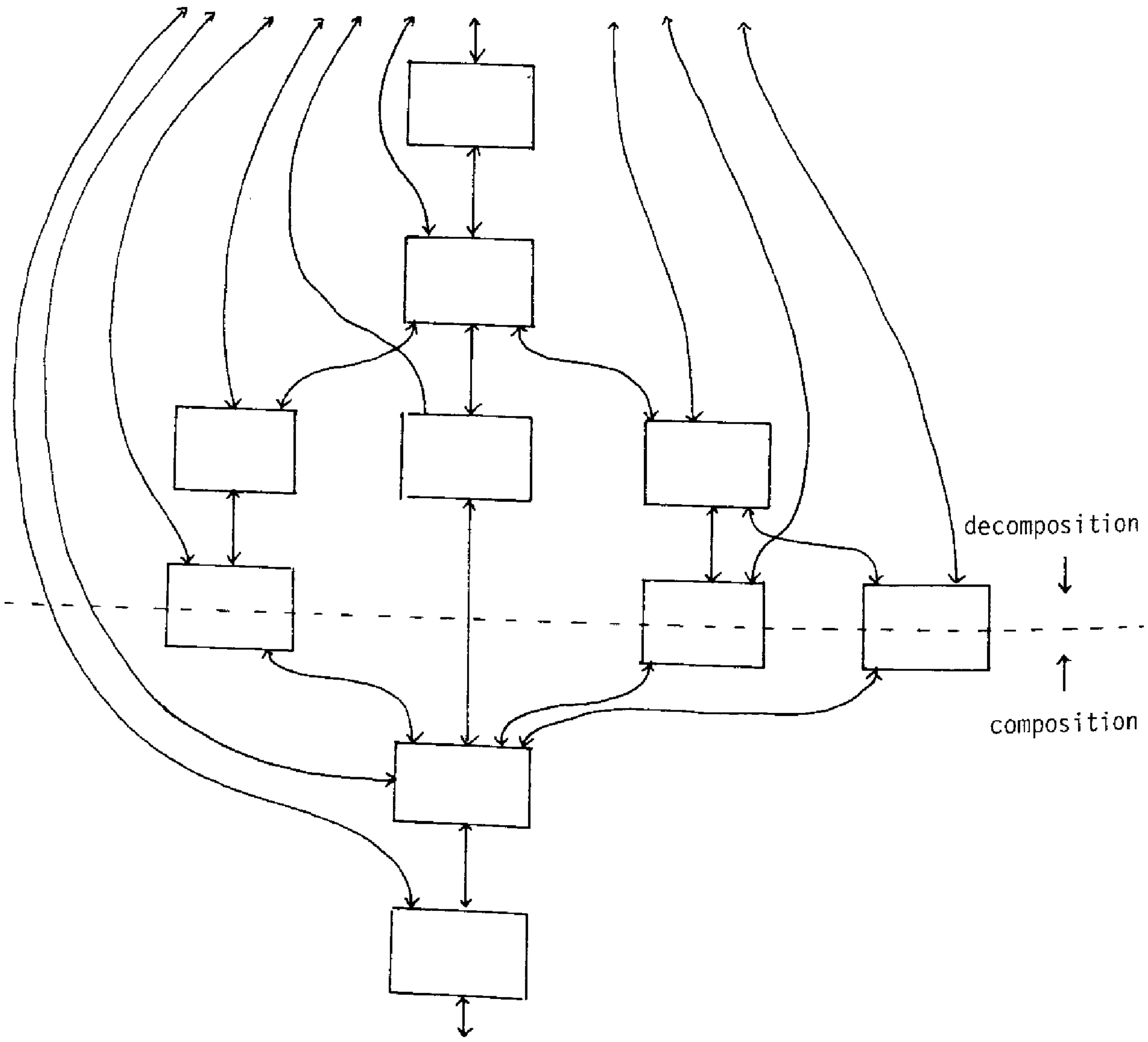
Up to now we did not mention the extremly important design activities "composition" and "decomposition". The reason for this is that decomposition is a typical deabstracting activity while composition (aggregation) is an abstracting one. I.e. we must have available a multilevel model to be able to speak about these activities.

It's not surprising that all of our three approaches to model a multilevel design process can easily expanded to compository/decompository schemes.

Because of this simplicity we will introduce these expansions only by means of drawings.

EXTENDED CHAINING

In order to expand the chaining scheme we simply have to transform our quasi linear list to a quasi tree. If we want to describe both decomposition and composition within one model we have to combine two quasi trees. In order to obtain a simple mapping from levels of abstraction onto levels of the quasi trees we request that all design process objects on one level of a tree lay on one level of abstraction. Now a decomposition/composition scheme may look like follows:



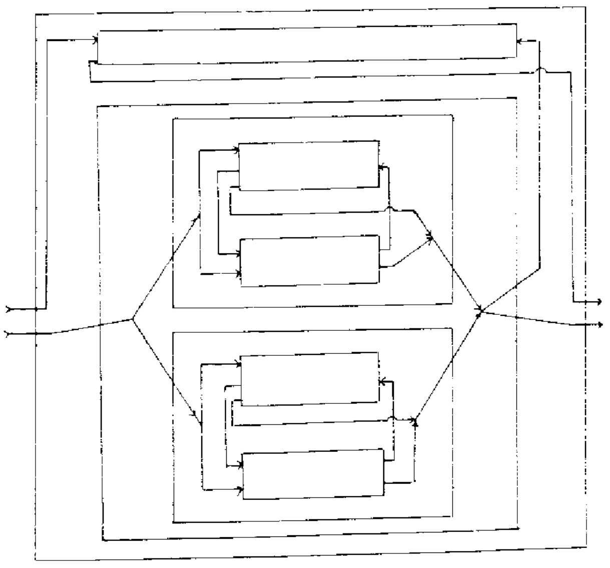
Again we have to pay for the generality of this approach by the deficiency that we obtain very heterogenous structures which tend to become unwieldy.

EXTENDED RECURSION

In the case of recursion we simply have to use lists of FLGs instead of a single FLG as substitution for subprocesses.

By the nature of the recursive substituted design process both decomposition and composition are covered by this single scheme.

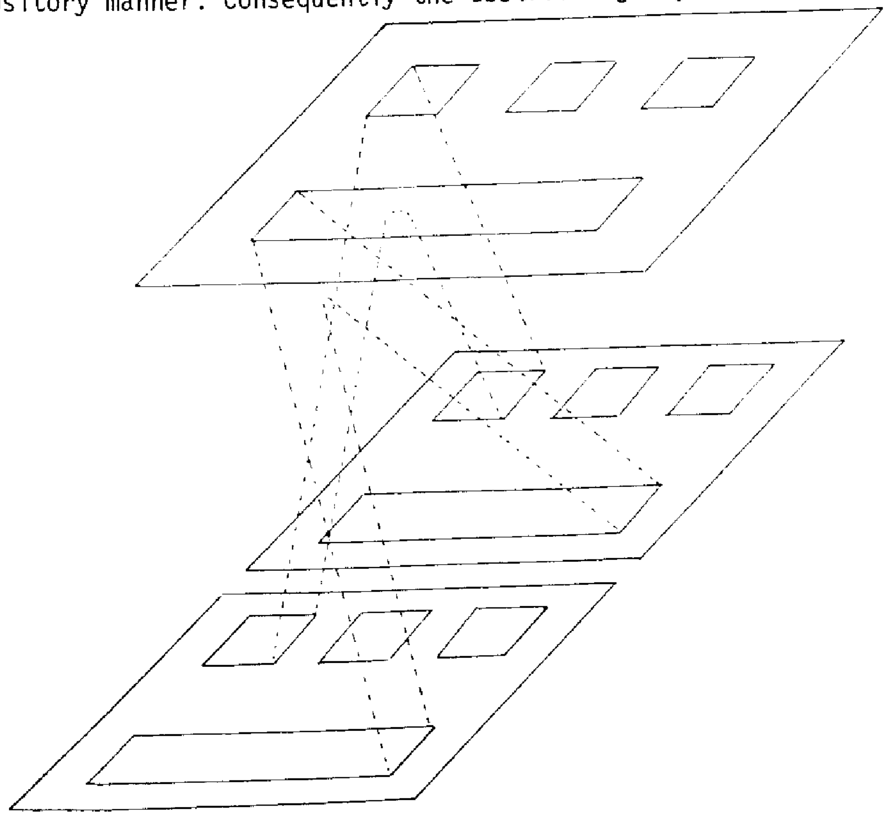
The following draft may illustrate the idea. In this example the generation subprocess is decompised/composed into two subprocesses at a lower level of abstraction.



EXTENDED MULTILEVELFEEDBACKING

The multilevel feedbacking may be extended in order to describe decomposition/composition as easily as the other approaches. The only thing to do is to allow cartesian products as range or domain of the deabstracting or abstracting subprocesses.

The following example shows a design process where implementation is carried out in a decompository manner. Consequently the abstracting regulation is a compository one.



DESIGN STRATEGIES

The most cited design strategies are "Top Down" and "Bottom Up". The first one may also be characterized as "stepwise refinement" the latter one as "stepwise aggregation". Both strategies are academic in nature. Therefore we will restrict ourselves within this paper to describe the realistic strategy "Yoyo" within the model of extended multilevel feedbacking. It may be assumed that a design task is entered into the system at a certain level of abstraction. A first reaction may be some modifications on the same level (levelinvariant generation) and at least a validation process on the input data.

Assuming that there are no more problems a first implementation may occur (levelvariant generation). This may happen either in a decomposing way or not. All design processes on the lower level of abstraction now have to check the implementation result. Problems may be solved on this lower level by some modifications or by regulating the design process one level higher (levelvariant regulation). Here we see the Yoyo game in action: The regulation function acting one level up is nothing else than a local "Bottom Up" activity. Note that this "Bottom Up" activity may have a reaction up to the highest level of abstraction. The usual reaction on an upward regulation is a modification on the higher level of abstraction, its verification and, if the redesign is satisfactory, a new implementation attempt.

HETEROGENOUS DESIGN SYSTEMS

Up to now we discussed only pure schemes for the modelling of multilevel design processes. One easily observes howerver that heterogenous schemes are possible as well and make sense. E.g. one component in a chaining scheme may be a recursive substituted design process containing a multilevel feedback process as one of its componenets. This feature is important especially when a new design system has to be created. Of course one will tend to start with a well structured approach at the top level, i.e. recursion or multilevel feedbacking. In practice one will be forced to leave the path of beauty when a more detailled planning of the design process takes place. Then different chaining configurations will become the basic structuring technique.

CONCLUSIONS

This paper presents an approach to systematic system analysis of design processes. By introducing levels of abstraction, dividing the design activities into generating ones and checking ones and finally both classes in levelinvariant and levelvariant ones one obtains a valuable classification scheme to identify separate design activities. This separation together with the set of composition rules offered makes the necessary information channels transparent.

REFERENCES

[AM1] J. H. A. E. Amkreutz:
Cybernetic model of the design process
Computer Aided Design, Vol. 8, No. 3, 1976

[BJ1] D. Bjorner, C. B. Jones (eds):
The Vienna Development Method: The Metalanguage
Springer Lecture Notes in Comp. Sc., Vol. 61, 1978

[KO1] C. J. Koomen:
Information Laws For System Design
Proc. Intern. Conf. on Cybernatics and Society, Tokyo, Nor. 1978, Vol. II

[MC1] C. Mead, L. Conway:
Introduction to VLSI Systems
Addison-Wesley, 1980

[RV1] C. R. Rupp:
Components of a Silicon Compiler System
Proc. First International Conference on VLSI, Edinburgh, August 1981