Modeling Design Tasks and Tools
- The Link between Product and Flow Model -

Bernd Schürmann Joachim Altmeyer
University of Kaiserslautern
D-67653 Kaiserslautern, Germany

ABSTRACT - The important step towards a comprehensive CAD framework is the development of a suitable, complete design model on which the design system's components are based. To date, we generally find "island" solutions for different aspects as data and process management, but in future, we need more and more integrated solutions. Only the integration gives us the traceability we need for design planning, to generate parts of the design tool's code automatically, etc.

This paper describes how a suitable Design Task Model can be used to link the Product and Flow Models which are currently separated in most frameworks. Using the PLAYOUT Design Model as an example, we show how the Product, the Task, and the Flow Models may fit together, and we describe a modeling environment that guarantees the development of consistent, integrated models.

I. INTRODUCTION

While early design environments were built-up of a more or less large set of stand-alone tools, today's ECAD research is increasingly involved in solving the problems of design tool integration and managing design processes. Important topics of CAD framework research are data and process management (e.g. [4], [5], [10], [11]) as well as the support of implementing the software of new design tools more easily [2].

The basis for solving all these problems is an adequate design model describing all design aspects. Based on such a model, we can develop a CAD framework including data and process management components which are completely integrated.

An overview of a comprehensive design model can be found in [4] and [9]. The second paper describes a model that is partitioned into following partial models:

The *Product Model* describes all design objects which are generated during the design process or during design planning.

The *Process Model* describes all dynamic aspects of the design including everything from planning the overall design process down to controlling the individual tools. Process management is typically divided into flow management and task management, described by two submodels. The *Flow Model* describes all possible sequences of design steps/activities for a given design methodology. It is the basis of planning the design process. The *Task Model* describes all information necessary to automatically start design tools. It considers input and output data, the path names of the tools' object code, necessary environment variables, format conversions, etc.

The *Environment Model* describes the design infrastructure. This includes the description of designers and design teams as well as the available hardware and software resources.

We can find more or less promising approaches for all these partial models in the literature (e.g. [4], [5], [6], [11], 14]). Although many approaches consider either data management or flow management in detail and neglect the other aspects, the partial models are not independent. Two of the few CAD frameworks which consider data and process management equally are the NELSIS and the PLAYOUT frameworks [1], [18].

There are several reasons, why it is necessary to keep all partial models consistent and why they have to be integrated. One reason is design planning which is based on both, the Flow and the Product Model. Another reason are software generators which are able to automatically generate the software of the design tools from the design model to a large extend [2]. Section IV describes the importance of an integrated design model for both examples in more detail. However, there exist many more reasons for the model integration.

In this paper, we focus on the derivation of a Task Model from a given Product Model and of a Flow Model from that Task Model. The automatic derivation guarantees the consistency of all models. The Task Model, often neglected in the past, is the link between the two other models and will be described in more detail.

The remainder of this paper is partitioned as follows. Section II gives a short overview of the partial models of the PLAYOUT design model and some related approaches. Section III then describes how we integrate these partial models to guarantee the necessary traceability. Finally, Section IV states the advantages of the model integration, and Section V concludes the paper.

II. A COMPREHENSIVE DESIGN MODEL

This section gives a short overview of the modeling notations used by the PLAYOUT framework as well as of some related approaches from literature. Of course, since there exist so many approaches, the list cannot be complete.

For data management we developed an object-oriented Product Model [14], [15] that was extensively tested by performing many designs. For modeling the design flow, we favor an extended, hierarchical Petri net model. The design tools and tasks are described by an own, set-oriented notation which will be explained below. The Environment Model will not be considered in this paper because of two reasons: first, these aspects are not fully covered by the PLAYOUT framework until now, and second, the Environment Model would not add new aspects to this paper. With respect to the model integration, it can be regarded similarly as the Task Model. Nevertheless, the Environment Model is still an important and interesting research topic.

DAC 97, Anaheim, California
(c) 1997 ACM 0-89791-920-3/97/06 ..$3.50

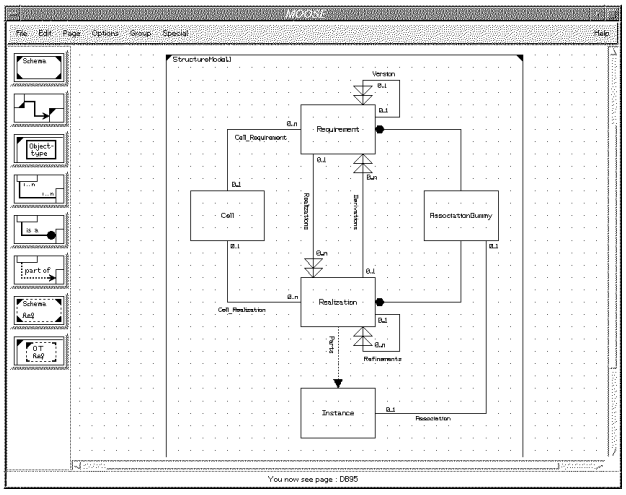


Fig. 1. EER schema editor

A. Product Model

Beginning with developing the first data exchange formats, much work has already been done for modeling the design objects (e.g. [5], [10], [18], [19]). These include micro data like pins and netlists as well as meta data which have to cover aspects like the decomposition hierarchy, type and version concept, configuration, requirement formulation, and design traces. Here, we do not want to describe any Product Model in detail because we are interested only in the model notation. Since the Product Model describes the structural relationships among the various data object, a graphical representation of these relationships is suitable. This approach is used in all cases we know.

We use an Extended Entity-Relationship (EER) model which - as we believe - describes the design objects and their relations best. Figure 1 shows an example of such an EER model. In addition, the figure also shows our graphical model editor. Besides the elements of the standard ER model, our editor supports inheritance (by an is-a relation) and aggregation (by a part-of relation). It was our goal to keep the number of basic modeling elements as small as possible without reducing the expressiveness of the model. Using the few elements shown in figure 1, we are able to describe all of our design data.

B. Process Model

Two different approaches exist for controlling the design process. The first approach, which was generally implemented with the early design systems, is *tool-based*. Here, the designer is completely responsible for his design without having computer support for supervising the design process. This approach is not suitable for complex designs. With increasing design complexity, computer-aided design process management became necessary which led to *task-based* systems (e.g. [1], [7], [11], [17]).

Figure 2 gives an overview of the task-based approach. The kernel of a design system is a computer-aided process management component. To date, it is common sense to divide the process management into a flow management component (planning and supervising the overall design process without considering the individual design steps in detail) and into a task management component (supervising the individual tasks or tools).

The flow management receives the data determining the current design state from the design database. For a new design step it signals the task management which supervises each design step individually.

Currently, the best solution for describing all necessary parameters for executing a design tool (path name, environment variables,

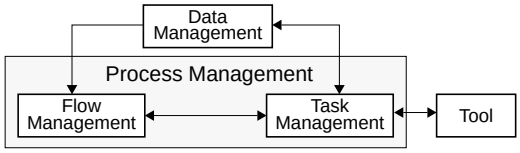


Fig. 2. Task-based design process management

etc.) is any kind of template. Providing the tools with design data needs to describe the necessary input and output data. Obviously, this aspect of the Task Model has to be consistent with the Product Model used by the framework. Although it seems to be reasonable to extend the EER model by further information, we favor a database approach similar to SQL as we will show below. Before that we present typical process management approaches.

- *DECOL* is the description language of the OASIS[1] process management [13]. Similar to a UNIX *make* file, a DECOL file consists of a set of rules which are used for controlling individual design steps. The file is interpreted by the process manager. The major disadvantage of the OASIS framework is the missing Product Model.
- The task management of *Ulysses* [7] is integrated into a real CAD framework. The Task Model is interpreted by a blackboard architecture instead of a fixed inference mechanism. Each design step is represented by a template that is similar to a DECOL template. However, the design management is more sophisticated. Unfortunately, there exists no explicit Product Model, too.
- The *Nelsis* framework belongs to the most advanced framework approaches. It is based on a central, object-oriented database [18] and on a flow management that uses a dataflow graph as Flow Model. The advantage of such a graph is its good overview of the overall design process. Although there is a tight relationship between the dataflow graph and the data model in Nelsis, both models are set up independently.
- The *PLAYOUT* framework is based on separate Task and Flow Models which are highly interrelated among themselves and with the Product Model. These interrelations are the main topic of this paper and will be described in detail below.

C. Task Model

Task management interprets a task frame for starting the tool. Depending on the type of the task, it also controls the input and output of the design data from and to the database. Further features of the task management are load balancing, name resolution, consistency enforcement, format translation, run logging, etc. To perform all these jobs automatically, the task management needs comprehensive knowledge of the design tools which is modeled by the Task Model.

For describing the tool execution, a template approach as described above seems to be best. Modeling the tool's input/output data, on the other hand, is different. The information has strong relation to the Product Model so that it seems obvious to extend the schema editor as shown in figure 1. These extensions have to describe how to retrieve the data objects for the given tools from a database. The retrieval engine needs to know which relation of the Product Model must be exploited to find the necessary tool input data. Set operations are necessary for each retrieval. The same is true for checking the tool's output data for consistency with the Product Model.

One possible way to extend the Product Model with information of the design tools might be inserting meta edges into the model. These meta edges describe the sequence of traversing the relations of the Product Model to reach objects whose data should

1. OASIS is a VLSI design system that has been developed at the Microelectronics Institute of North Carolina (MCNC).

be retrieved. Additional annotations describe required attributes of retrieved objects and operations on the elements of the relations (mostly set selection operations).

However, we think that such a graphical approach is not suitable for the Task Model. There are two main reasons:

- the graphical representation of all retrieval and validation statements together would be too difficult to survey
- in some cases the sequence of executing the retrieval statements may be of interest which is difficult to describe graphically.

In the PLAYOUT framework we use an approach that is similar to the SQL approach but our retrieval notation, addressed in the following section, is simpler than SQL.

PLAYOUT task management

The task model describes the data objects which are needed for starting a tool and which are expected to be the output of the tool. With that the task description has references to the object types of the Product Model. However, many Task Models are weak in referencing these object types. Using these references, the task and flow management need not have copies of the data stored in the database. E.g., the PLAYOUT Task Model is implemented on top of the design database so that some aspects of the task management could be delegated to the database retrieval engine and transaction management. The task and the data management are tightly coupled.

The PLAYOUT design database is an object-oriented, prototype database implementation written in Smalltalk. The underlying data schema has automatically been generated from the Product Model by a software generator, called MOOSE [2]. The database interface provides set-oriented access methods for each object type and each relation of the Product Model [16]. These access methods are named identically to the elements of the data model. With that, retrieval code mainly consists of concatenations of the database relation names only.

EXAMPLE

- `Cells cells realizations derivations`
  `Cells` represents a variable that holds a set with all instances of the object type `Cell` (see Figure 1). The method `cells` is applied to this variable and returns a set with all cell objects. The method `realizations` (implementing a relation of the Product Model) is applied to each element of this set which results in a new set with all realization (i.e. version) objects of all cells. Finally, the method `derivations` is applied to the new set.
- `((Cells cell: 5) realization: 3) parts`
  returns all subcells of the realization version `3` of cell `5`. Here, we have two further access methods, `cell:` and `realization:`, which are also generated automatically by MOOSE. These methods do not return a set with all members of the corresponding relation but only a set with at most the one element that has the specified key. For instance, the term `Cells cell: 5` is equivalent to the statement: `Cells cells select: [eachElement cellKey == 5].` ❑

Besides the select statement and the set of access methods corresponding to the elements of the Product Model, the retrieval language provides additional constructs for set manipulations (union, intersection, difference, iteration) and logical operators for writing more complex expressions. However, the most interesting aspect of our retrieval language are the methods provided by the software generators which implement the relations of the data model in the most easy way. Browsing the data model for the data retrieval is expressed by a simple concatenation of the relation names.

The retrieval code is implemented by Smalltalk methods. A method header contains a parameter representing the root object for the retrieval, and a parameter representing the retrieval stream. The method name and the root object description will be passed to the design tool as a task description. Figure 3 shows a simplified retrieval code for the design tool *Floorplanner.*

```
generateFloorplanFromNetlist: aFrame on: aRetrievalStream
| currentVersion |
currentVersion := aFrame derivedFrom.
" retrieve the structure data "
  " retrieve the structure interface data of the subcells "
  aRetrievalStream add: (currentVersion parts association attributes: #(cellname porttypes)).
  . . . . . .
" retrieve all subcell frame data "
  currentVersion parts association do:
     [eachElement attributes: #(frame pins) notEmpty
        ifTrue: [aRetrievalStream add: (eachElement attributes: #(frame pins))]
        ifFalse: [aRetrievalStream add: (eachElement allVersions attributes: #(frame pins))].
  . . . . . .
```

Fig. 3. Floorplanner retrieval code

After running a design step, the database has to check for the consistency of the generated data. In contrast to standard database systems, a CAD database cannot check the complete semantic correctness of all data. The correctness of the micro data is validated by special verification and simulation tools, e.g. design rule checkers.

In the PLAYOUT CAD framework the transaction management validates the output data with respect to the Product Model and the Task Model. The consistency of the output data is modeled by methods which are very similar as for the data retrieval. These methods contain statements which compare the newly generated data objects with a set of data objects expected to be the tool's output.

D. Flow Model

The flow management has to plan and to supervise the overall design process [4], [6], [11]. It communicates with the data management that provides informations about the current design state and with the task management that controls individual design steps (see figure 2). The basis for the flow management is a suitable Flow Model. It describes all possible sequences of design steps. In many cases the graph of all possible sequences is restricted to a certain design methodology. For instance, top-down design steps during the physical design phase may be forbidden. The corresponding elements of the Flow Model must then be removed. It is also possible to favor particular design tools if alternative tools are applicable in the same design state.

Existing Flow Models

The development of different Flow Models was influenced by research topics like system and net theory, databases, and artificial intelligence. Currently, existing Flow Models can be grouped into three classes:

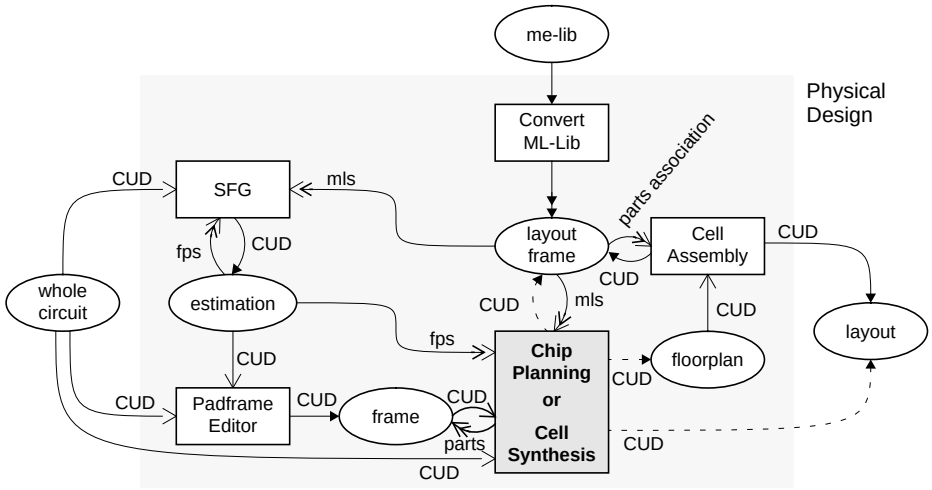
Procedural models. Design processes are controlled by shell scripts and procedural programs which are relatively rigid.

Knowledge-based models. Control flow is described locally by pre- and postconditions of single design steps which are interpreted by a central inference mechanism. While these process models are very flexible, their local view of the design process makes it difficult to understand the overall control flow. Examples of this class are UNIX *make*, Ulysses, and DECOL.

Net-based models. Systems are described by directed graphs with active elements as nodes. The edges of the graphs represent the relationships between the active elements (in our case data constraints among design tools). Net-based process models combine the advantages of the other two model classes: flexibility and global view. For describing design flows, we currently find three net models: *finite state machines* (e.g. ADAM Design Planning Engine [12]), *dataflow* model (e.g. Nelsis' Flow Model), and (extended) Petri nets.

The PLAYOUT Flow Model

The PLAYOUT design flow is modeled by predicate-transition nets (PrT nets) which are extended Petri nets. Petri nets are bipartite, directed graphs with transitions (representing the active system elements, e.g. design tools), places (representing the passive elements, e.g. design object types), and tokens on the places (e.g. representing design objects). Extensions are annotated transitions, places, and arcs (as well as global annotations). The annotations of places define the types of tokens stored on the places. Input arcs are annotated by terms identifying tokens which must be available on the input places for enabling the corresponding transition. Annotations of output arcs describe the newly generated tokens after firing the transition. Transitions are annotated by rules which extend the precondition (e.g. for implementing the design methodology). The formal definition of PrT nets can be found in [8].



mls ::= CUD parts association select: [eachElement domain == masklayout]
fps ::= CUD parts association select: [eachElement domain == floorplan]
parts ::= CUD parts

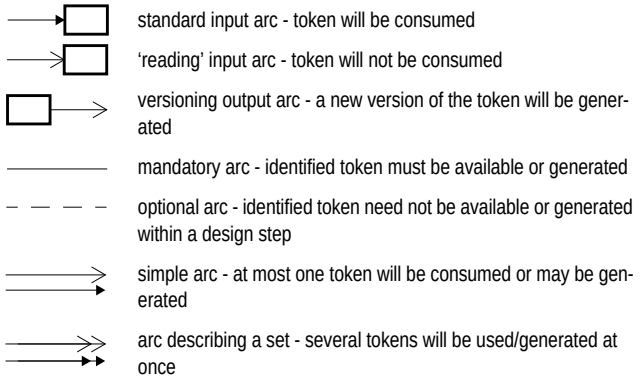


Fig. 5. PrT net of the physical design phase

Hierarchical net model

Hierarchical net models are very important for an adequate design process abstraction. The Flow Model, i.e. the PrT net need only be as precise as it is necessary for the current process state. At the beginning of a circuit design we are interested in global steps like behavioral and geometrical design phases (figure 4). Later, when we have more design knowledge, it becomes necessary to refine the Flow Model (figure 5). The firing of a transition at an upper hierarchy level causes the activation of a subnet [6].

Figure 4 shows a simple top-level Flow Model of the VLSI design. Shaded rectangles describe hierarchical transitions which represent more or less complex subgraphs. This example shows two hierarchical transitions and one design step (Repartitioning). The first phase in the design for a new cell is the generation of the structure description which needs library information (s-lib) only. The hierarchy of the whole circuit may or may not be repartitioned before the physical design phase takes place.

The PrT net model is strongly related to the Task Model presented above. Since each transition with its input and output places models a single design step, such a section of the Flow Model is equivalent to the corresponding section of the Task Model. The precondition of a transition corresponds to the retrieval method and the postcondition corresponds to the validation method of the related task.

The extended PrT nets use set-based terms in their arc annotations (see figure 5). The function symbols in the terms are conform with the function symbols of the Task Model. These are mainly the relation names of the Product Model, set operators, and operators of the boolean algebra. Places are annotated by attribute names which are consistent with the Product Model.

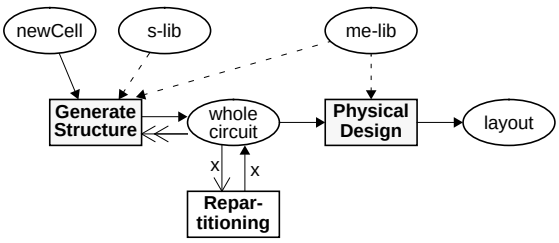


Fig. 4. Simplified top-level PrT net of the VLSI design process. The semantics of the arcs is depicted in figure 5.

Figure 5 shows a PrT net for the physical design phase which was abstracted by a single transition in figure 4. This net also contains a further hierarchical transition that abstracts from chip planning and cell synthesis. The different types of arcs and the labels of the arcs are described in the figure. The term CUD represents a variable describing the current design object and the other labels are shortcuts for the set-oriented expressions of the Task Model. ❑

The input arcs of a transition correspond to a retrieval method as shown in Figure 3. The annotations of the output arcs and the validation statements describe the same aspect, but in a different way. In both cases, the output data of a design step are addressed. However, the arc annotations describe these data in a constructive manner (they describe which new tokens will be generated) while the validation statements describe the data in a destructive manner (they declare which objects have to be removed from the list of input objects).

III. Combining Data and Process Management Provides Traceability

Section II.A describes the Product Model that represents the static design aspects and sections II.C and II.D describe the Task and the Flow Models which represent the dynamic aspects. The three models are not independent of each other. The basis for modeling design processes must be a suitable data model. Otherwise, if we do not regard the design data while talking about the design

process, we neglect the data dependencies which may cause conflicts in the Flow Model. A Flow Model that is not based on the Product Model results in a duplication of the data model or in incorrect process management in worse case.

The PLAYOUT framework approach avoids these problems by exploiting the interrelations of the models. Since the Product Model is the basis of all other modeling efforts, that model has to be set-up first. This step may be done independently of any knowledge of the design flow or the design tools. For instance, we may take the standardized VHDL data model that has been defined independently of any particular design system. However, for optimizing the data management in a CAD framework, requirements of the design tools should be taken into account.

A. Verifying the Task Model

In PLAYOUT, the input/output behavior of the design tools (tasks) are modeled textually. An example has been given above (figure 3). Since the relation names and their sequence depend very much on the Product Model, we propose to use a syntax-oriented editor for writing the retrieval and validation code.

The grammar which is the basis of the editor basically defines the object-oriented retrieval methods containing set-oriented expressions only. The part of the grammar that describes all allowed relation names will be generated automatically from the Product Model and included into the grammar. In addition, the allowed sequences of concatenating the relation names in the retrieval methods are extracted from the Product Model, too, and imported into the semantic analysis of the editor. Since it is possible to only set-up retrieval and validation methods which meet the specified grammar, the syntax-oriented editor may guarantee the consistency of the Task Model with the Product Model. The Task Model editor avoids writing database access code that is incompatible with the Product Model.

B. Generating the Flow Model

The Flow Model has to be generated on top of the Task Model, i.e. the Task Model must be available before setting up the Flow Model. Both models are closely related since each task modeled by the Task Model is an active element in the Flow Model - a transition in the case of extended PrT nets. The design object attributes which are the elements of the database access correspond to the tokens and places of the PrT-net-based Flow Model. With that, the PrT net that describes all possible sequences of design tasks can automatically be generated from the Task Model.

Each pair of retrieval and validation methods described in section II.C is translated into a subnet with one transition (representing the corresponding design task) and into the input and output places of the transition. An attribute addressed by the retrieval method is translated into an input place, and a retrieval statement is translated into an input arc annotation. Similarly, attributes and remove statements of the validation method are translated into output places and output arc annotations. After translating all task descriptions into basic PrT nets, these nets are combined to achieve the overall Flow Model. This is done by combining all places into a single place which represents the same attribute. This step makes all variable names unique for the whole net, too. The result is an automatically generated Flow Model which describes all possible sequences of design tasks and which is consistent with the Task Model (and hence with the Product Model, too). It is the lowest level of a hierarchical Flow Model. The PrT net can then be partitioned into abstract design steps of higher levels, and it can be restricted to a particular design methodology by adding additional rules to the transitions.

Since the semantics of the retrieval statements is identical to the semantics of the input arc annotations (both describe the required data which must be available to start a design step), the corresponding translation step is straightforward. A little more complicate is the translation of a validation method into output arc annotations because the semantics is different. While the statements of the validation method describe the removal of data (from the list of input objects), the arc annotations describe the generation of new data (tokens). On the other hand, the addressed data (attributes) are the same in both cases. The translation step can be simplified by using a slightly different intermediate format for the Task Model editor from which both, the validation method and the output arc annotations can be generated easily.

C. Overall model development process

Figure 6 depicts an overview of all generation steps described above. These steps enable us to generate the Product, the Task, and the Flow Models in a consistent manner.

The figure clearly shows that the Product Model is the basis for the Task Model and that the Task Model is used for generating the Flow Model. After analyzing the ECAD domain we use our graphical model editor to set-up a suitable Product Model from which software generators can automatically generate the data management components of our design tools [2]. From this Product Model, one of the software generators can generate the variant part of the Syntax on which the syntax-oriented editor, needed for writing the Task Model (retrieval and validation methods), is based.

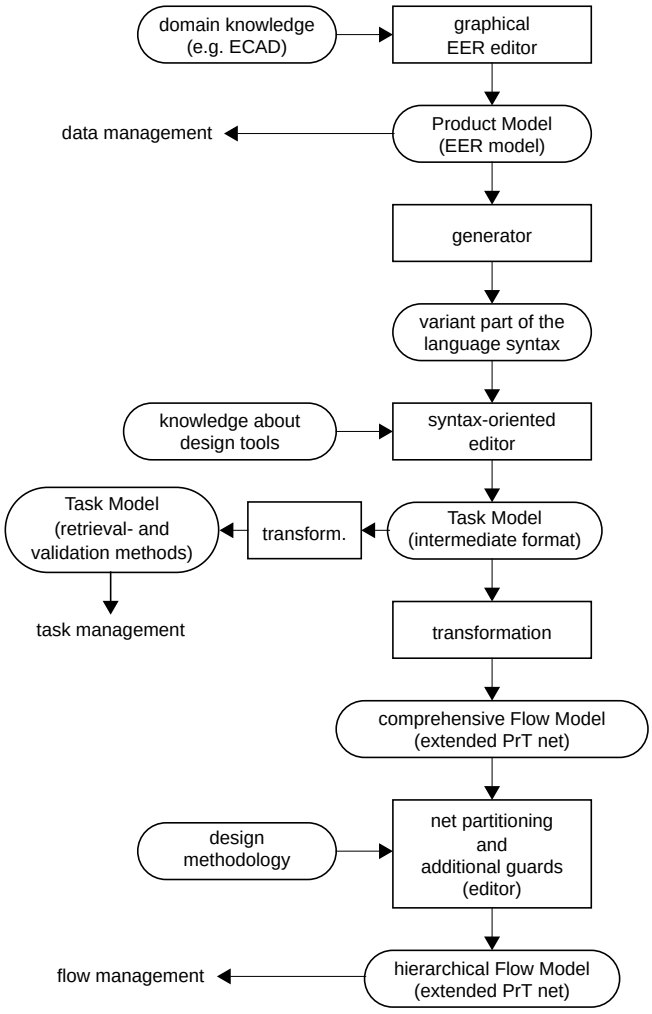


Fig. 6. Model generation process

An intermediate format of this Task Model is then used for generating the final Task Model that is used by the database retrieval engine as well as for generating a comprehensive Flow Model. Including some additional guards into the Flow Model and performing some partitioning steps, we get a hierarchical Flow Model (PrT net) that is adapted to a given design methodology.

IV. Advantages of a comprehensive, integrated design model

Besides the fact that an integrated, comprehensive design model is the basis of all good design environments, we would like to address two further advantages. The first one is the incorporation of the database into the design planning phase and the second one is the automatic generation of software from the design model. Due to space limitations, we will address these topics very briefly. Further information can be found in [2].

A. Process planning using the database

The Flow Model is the basis for controlling *and* planning the design process. While it is obvious that controlling an actual design process needs an intense interaction between process and data management, this is also the case for the process planning phase of our approach. In contrast to other PrT net approaches which support stand-alone flow management (e.g. [6]), our PrT nets need not have auxiliary places which store tupels representing important relations of the Product Model, e.g. the part-of relation. This information is provided by the database even during the process planning phase.

During the planning phase the task and data management *simulate* design steps and the generation of new design objects. For each simulated design step, the database stores the corresponding output data objects with special tags. These design objects need not have micro data. However, for a good planning strategy, it is sensible to replace the design tools by suitable estimation tools which also generate (estimated) design data which can be exploited by further planning steps.

B. Software generation

An important part of a CAD software development environment are software generators. To improve the implementation time and the correctness of software, we try to automatically generate parts of the implementation code of our design tools and of our central database.

Using our software generation environment called MOOSE (Model-based Object-Oriented Software generation Environment [2], [3]) we are already able to generate the data management code of the design tools and the data model implementation of the design database automatically from the Product Model. The database transaction and retrieval engine is generated from the Task Model. The integrated model development approach described above guarantees that the retrieval code of the database will fit to the underlying data model.

A big advantage of our generator approach compared to using fixed software libraries is that our generators are able to react on the various needs of the design tools and to generate highly tool-specific code [2]. This can be achieved by specifying optimization requirements for a specific tool which will be - in combination with the Design Model - the input of the code generator. The model integration also ensures that the Flow Model (PrT net) and with that the design planning component is consistent with the “automatically generated database”.

V. Conclusions

This paper proposes an integrated, comprehensive design model. We have addressed all modeling aspects which have to be considered by a CAD framework. The main part of the paper describes how we can use the Task Model (describing the I/O behavior of design tools) as a link between the Product Model and the Flow Model. The main focus was to guarantee the consistency between the partial models of the overall design model.

References

[1] Altmeyer, J., Schürmann, B., Schütze, M.. “The Framework of the PLAYOUT VLSI Design System”. In “Proc. GI/ITG/GMA-Fachtagung “Rechnergestützter Entwurf und Architektur mikroelektronischer Systeme“”, Oberwiesenthal, 1994.

[2] Altmeyer, J., Schürmann, B., Schütze, M.. “Generating ECAD Framework Code from Abstract Models”. In “Proc. 32nd Design Automation Conference (DAC)”, San Francisco, June 1995.

[3] Altmeyer, J., Schürmann, B., Schütze, M.. “A Generator-Based ECAD Framework Approach”. SFB 124 report No. 06/95. University of Kaiserslautern, 1995.

[4] Barnes, T.J., Harrison, D., Newton, A.R., Spickelmier, R.L.. “Electronic CAD Frameworks”. Kluwer Academic Publishers, Norwell, MA, 1992.

[5] Brielmann, M., Kupitz, E.. “Representing the Hardware Design Process by a Common Data Schema”. In “Proc. Int. European Design Automation Conference (EURO-DAC)”, Hamburg, 1992.

[6] Bretschneider, F.. “A Process Model for Design Flow Management and Planning”. In “VDI-Fortschrittsberichte, Volume 9, No. 157”. VDI-Verlag, 1993.

[7] Bushnell, M.L., Director, S.W.. “VLSI CAD Tool Integration Using the Ulysses Environment”. In “Proc. 23rd Design Automation Conference (DAC)”, 1986.

[8] Genrich, H.. “Predicate / Transition Nets”. In “Petri-Nets: Central Models and their Properties, LNCS 254”, W. Brauer, W. Reisig, G. Rozenberg, editor. Springer-Verlag, 1986.

[9] Hübel, C., Ruland, D., Siepmann, E.. “On Modeling Integrated Design Environments”. In “Proc. 1st European Design Automation Conference (EURO-DAC)”, Hamburg, 1992.

[10] Katz, R.H.. “Toward a Unified Framework for Version Modeling in Engineering Databases”. Journal ACM Computing Surveys, December 1990.

[11] Kleinfeldt, S., Guiney, M., Miller, J.K., Barnes, M.. “Design Methodology Management”. in Proceedings of the IEEE, 1994

[12] Knapp, D.W., Parker, A.C.. “A Design Utility Manager: The ADAM Planning Engine”. In “Proc. 23rd Design Automation Conference (DAC)”, 1986.

[13] N.N.. “OASIS Users Guide and Reference Manual”. MCNC, Research Triangle Park, NC, 1992.

[14] Schürmann, B., Altmeyer, J., Schütze, M.. “An Improved Data Model for Top-Down Design”. In “Proc. Int. Conference on Computer-Aided Design (ICCAD)”, San Jose, CA, 1994.

[15] Siepmann, E., Zimmermann, G.. “An Object-Oriented Datamodel for the VLSI Design System PLAYOUT”. In “Proc. 26th Design Automation Conference (DAC)”, pages 814-817, Las Vegas, 1989.

[16] Schürmann, B.. “Modeling Design Data and Design Processes in the PLAYOUT CAD Framework”. Journal Current Issues in Electronic Modeling (CIEM), Vol. 6, June 1996.

[17] van der Wolf, P., Bingley, P., Dewilde, P.. “On the Architecture of a CAD Framework: The NELSIS Approach”. In “Proc. 1st European Design Automation Conference (EDAC)”, 1990.

[18] van der Wolf, P., Sloof, G.W., Bingley, P., Dewilde, P.. “Meta Data Management in the NELSIS CAD Framework”. In “Proc. 27th Design Automation Conference (DAC)”, 1990.

[19] Wagner, F.R., Viegas de Lima, A.H.. “Design Version Management in the GARDEN Framework”. In “Proc. 28th Design Automation Conference (DAC)”, 1991.