# Information Model of a Compound Graph Representation for System and Architecture Level Design

Peter Conradi
Center for Microelectronics
University of Kaiserslautern
Federal Republic of Germany
Email: conradi@rhrk.uni-kl.de

## Abstract

In order to extract a suitable common core information model, design representations on both system and architecture levels are analyzed. Following the specification trajectory, three design phases with different description methodologies are found to be widely used. The representations can be affiliated in an easily upgradable way using a compound graph representation.

An information model of the control-dataflow domain is further explained as an example of the information modeling style.

## 1. Introduction

Computer Aided Design (CAD) research and development in the past two decades have resulted in fairly mature design tools and automation of several steps in the logic and layout design of electronic circuits.

Arriving at the point of more complex programs and frameworks for Electronic Design Automation (EDA), tool interoperability becomes increasingly important. A partial aspect of tool interoperability is the definition of procedural access interfaces transferring design status and actual data between tool sites. To guarantee distinct access and to save translation efforts, the design data types have to be partitioned as far as possible, while connected parts have to be transmitted as linked to each other.

Working on such a model, common for system and architectural levels of EDA, we rely on an EXPRESS-G[1] similar entity-relationship presentation [1]. In this work we use the technique of information modeling to analyze the semantics of common grounds and differences concerning system and architecture level design. Information modeling is a kind of conceptual meta-modeling of data structures. In opposite to the data modeling, which concerns computer language primitives, information modeling is used for more applied modeling aspects. Using EXPRESS-G, we use structural object-oriented description methods by adding the specific concept of type inheritance to other relationships. Using such an information model, procedural interfaces, exchange formats, or database schemes can be derived from the same meta-modeling source.

In the past, information modeling was invented for defining data base schemas for EDA design repositories [2,3]. In a second step it was used to support cooperative work and standardization efforts [4,5]. Application topics for information model applications in CAD environments affect

- all tasks, helping the designer to organize his information space, version handling mechanisms, preparation for reuse, documentation etc.,
- all domains, in which designers have to manipulate different design descriptions simultaneously, and
- all design phases and abstraction levels, in which different abstraction levels are to be managed.

User access to such a system could be supplied by a number of access modes that supporting specific needs on different views (black box information, floorplan infor-

[1] the graphical product data specification language of ISO 10303-11

mation, technological conditions, register transfer-behavior, -structure, etc.).

At present, the technique of information modeling also supports generic construction of databases and tool interfaces for EDA domains [6].

The paper is structured as follows: After presenting an overview of related work and our objective (section 2), we focus on the analysis of system level design graphs (section 3). We then present a methodology for the representation of system level objects combined with architectural level objects. Section 4. describes the treatment of additional finer grained information, which will be needed for related implementation purposes, while section 5 includes reports on experimental studies.

## 2. Basic Motivation

To derive a core information model on system and architecture levels, design data could be important for two reasons:

- In order to create long term strategies for top down synthesis over the whole design and analysis trajectory [7] it is essential to understand all the links between specification and design levels.
- Also, for design back annotation purposes as well as documentation reasons it is always worth striving for the conservation of links between system level, architectural level, and any kind of implementation. This should be realized as far as possible.

### 2.1 Related Work

On the system level, the communication of agents, their configuration, performance, and dependability is the major area of interest. A model of communicating agents, memories, and variables, widely used for teaching purposes in [8] has been taken into account, as it is sufficiently general for very high level specification and well suited for data refinement strategies. It was designed for specification of both voluminous concurrent industrial software and related documentation purposes [9].

Another analysis source is the ADEPT system, a specific methodology of system level performance and dependability analysis.

Additionally, we analyzed the interface formats of SPECSYN [10,11]. Since SLIF [12] is the basic input-output data format for SPECSYN, its information model offers one set of the data types needed on the system level.

In contrast to the comparatively simple graph constructs on the system level, the architectural design level comprises control and data flow, finite state machines and netlists [13]. In architectural level synthesis, there is a need for agreement on a common procedural format for cooperative work, and so both a C++ functional interface of the Synthesis-Internal Representation (SIR) was defined in the project SYDIS[2] and an EXPRESS information model was derived [14]. This information model represents another source for this work.

Some particular concepts are also used in our approach, but for brevity they were ignored in the paper explanations:

- **Versions and Alternatives:** everything indispensable for storing design data in different stages of the design and with different selected realization properties. For instance, in the VLSI design system PLAYOUT concepts for versioning and alternatives enable top down design by both constructing versions of realizations from requirements and deriving versions of requirements from realizations [3,7].
- **Connectivity:** basic net and library construction. Standardization committees like the Electronic Design Interchange Format (EDIF) [4] and the CAD Framework Initiative, Inc. (CFI) [5] propose related information models for folded and unfolded design.
- **Constraint mechanisms:** different constraint types like timing, power, and wafer area. Constrains are added as annotations or derived attributes to different parts of the information model. For brevity, this is not explained in detail.

### 2.2 Our objective

The aim of this paper is to present a new specification refinement and type splitting method from the information modeling point of view combined with dependencies between related graph types, and to define a compound graph semantics for discussion, prototype implementation, and future standardization input.

[2.] founded by German Ministry for Research and Technology

## 3. Compound System and Architecture level Representation

In the early system specification phase, no underlying realization model exists and no implementation details are required. It may not be known whether the design is realized by hardware or software implementation.

During design activities, the successive splitting of the required design graph types provides decisions like hardware-software implementation, technology selection, and more and more semantic distribution of properties over the design trajectory.

### 3.1 Single-Graph

At the beginning of the design a typical user starts on the upper system level specification with a single graph representation, which we call the **module net**. In a single-graph representation there is no syntactic difference between control, data flow, access, or structural representation.

An example for single-graph representation on the system level is given, e.g., by the ADEPT (Advanced Design Environment Prototyping Tool) [15] in its uninterpreted form. Single-graphs are specification methods, comprising control and transformational, and structural properties in a single representation. This is achieved by using Petri Nets in different variations. Tables 1-3 give some examples of each of the used types of representations.

**Table 1:** Different Single-Graph types

| **type I** Modules | Reference |
|---|---|
| ADEPT modules | Kumar et al. [15] |
| Predicate-Transition Net | Rammig [16] |

### 3.2 Dual-Graph

Using a single graph as a starting point, the design refinement can be accomplished by splitting and refining the semantics of the module net elements. Typically, the difference is worked into these elements, defining **functionality** and **resources**; the latter executes defined functional parts. Depending on the objective of a system level design, very different instances for a dual graph representation (shown in Table 2) can be detected. A suitable example for a resource net (type-Ia), called Agent-Memory Net [8], is a bipartite graph with directed edges, consisting of information processing (agent) and storing (memory, variable) nodes, as shown in Figure 1. Memory symbols (cycles) are separated from agents (rectangles), and the special semantics of the latter is to enable and synchronize communication services between agents.

In hardware systems, agents are resources representing abstract models of physical units, realized either by processors or application specific components, and memories are exchanged variables. In software implementations, agents may be parallel tasks. In combination with a Petri Net for control expressions (type-Ib), this method is a dual-graph-construct.

**Table 2:** Different Dual-Graph types

| **type-Ia** Resources | **type-Ib** Functionality | Reference |
|---|---|---|
| Agent-Memory Net | Petri Net | Wendt [8] |
| Structural Objects | Functional Objects | Vahid, Gajski [12] |
| VHDL-Component | VHDL-Process | Giumale, Kahn [17] |

The type-Ia and type-Ib graphs are principally separated from each other, however showing interrelations. Type-Ia units execute resources in a calculative or transformational manner, while Type-Ib units control them.

Since resources are not always understood as hardware constructs, the application of the dual-graphs strongly depends on the designer's view.

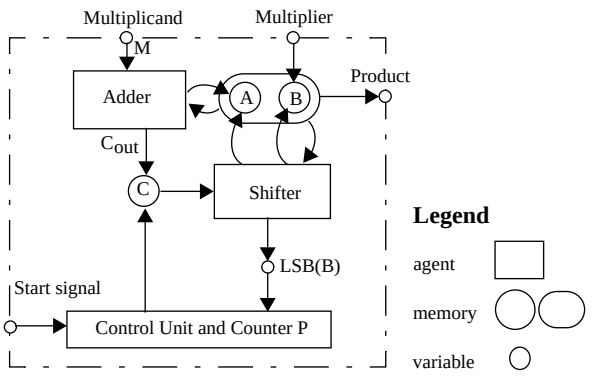


**Figure 1**: A multiplier resource example

### 3.3 Triple-Graph

During the next design phase, the triple graph construct can be derived by a further splitting of the type-Ia or type-Ib objects into permanent and part-time required resources (Table 3). As we see, the different approaches do not

**Table 3:** Different Triple-Graph types

| **type-Ia1** physical component | **type-Ia2 or type-Ib1** activity | **type-Ib2** control and state | Ref. |
|---|---|---|---|
| Annotated Component Graph | CDFG | State-Transition Graph | Rundensteiner, Gajski [13] |
| (StateMate) Module-Chart | (StateMate) Activity-Chart | (StateMate) State-Chart | i-Logix |

clearly express a common way to split the phase-two-descriptions into phase-3 graphs. Figure 2 shows graphically, how the splitting process is performed over the trajectory. The unnamed attributes in Figure 2 are derived

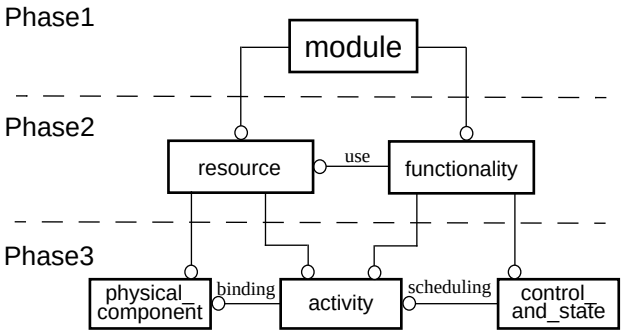


**Figure 2**: System level entity splitting

relationships. Since the graph splitting depends on the designer's working style, we allow derivations of the activity graph from either the resources or the functionality graphs. Our aim is to construct the compound graph representation by supporting support both resource-activity and functionality-activity derivation.

In the following text we present aspects of a core information model, supporting the design trajectory on one hand and the successive graph extension on the other.

### 3.4 Resources

Resources are defined as an abstraction of available hardware representing physical devices, which can be reused for a design. In the early beginning of a design (phase 2), it may be unclear, how often the allocated resources will be referred to. And it depends on their usage demands, their frequency and timing constraints, whether they are realized as data flow expressions or as physical systems in the third phase of the design trajectory. To realize resource nets, either folded or unfolded archivation can be used. Both variants can be represented by structural repository concepts as shown in [4,3,5].

### 3.5 Functionality

The functionality of a design can be expressed as a flow of control, along with associated data transformations, as already presented in [18]. In a general context, control modeling can be performed synchronously or asynchronously as shown in Figure 3 . Furthermore, control flow

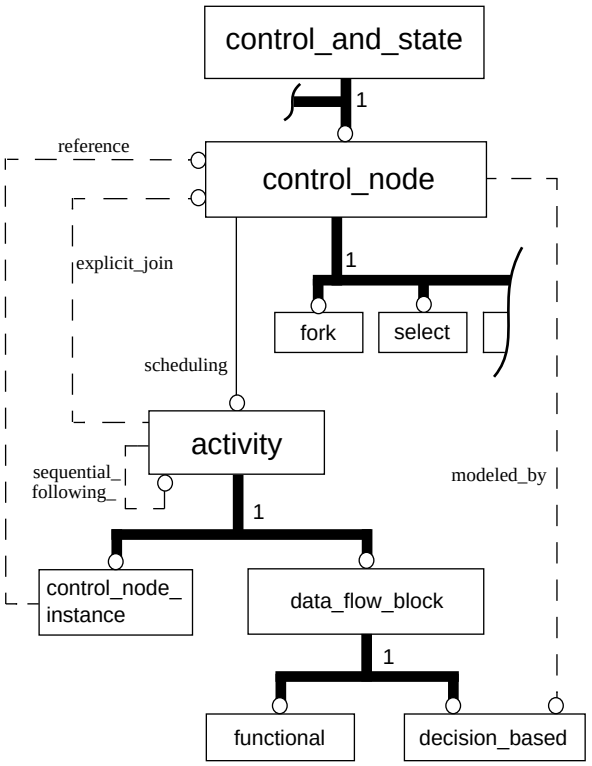


**Figure 3**: Control-Activity interaction (EXPRESS-G)

itself can be serial (sequence), parallel (fork), or conditional (select). The main entity for control is called *control_node*, a meta-control unit, which can be inter-

preted as a superset of different modes like, e.g., a selection (*select*), or a parallel control flow (*fork*).

A *control_node* has the capability of scheduling either the beginning of an activity or supervising the complete execution path, eventually including the related join construct. This can be justified individually by the attribute *control_kind* (not shown here), which refers to an enumerated type. If *control_node* supervises the complete execution, the entity *control_node* can interrupt the execution of the controlled *activities* and any contained active *control_nodes* at any time if necessary. Using *control_kind*, it can also be defined whether a *control_node* is continually active. STATEMATE, for instance, provides design modeling features like „history“, exceptions, or compound transitions, which can easily be meta-modeled by fine grained attributes of the *control_node* of the information model in Figure 3.

The *control_node* is based on a *decision*, which could be modeled by a *decision_based data_flow_block*.

Hence it is possible to model structured control-flow graph (CFG) and fork-join graph (FJG) constructs. The feature of a behavioral hierarchy represents one of the fundamental properties to express the semantics of system level specification programs, like SpecSyn [11]. A behavioral hierarchy is realized by an activity, representing an instance of another *control_node*. In this concept, subprogram access and conditional statements are realized in a common graph type fashion.

The atoms in the functionality view are *activities* which are interpreted, for instance, as data flow constructs containing connected operation nodes, where the edges are understood as informations on data dependencies. The events of processing, like activities, are scheduled to one control command or transition.

## 4. Refinement on Implementation Level

As stated above, the implementation is defined as a refinement of the system level definitions, depending on the partitioning decisions.

In architectural level synthesis tasks it is important to specify the agents' structural properties in further steps of granularity refinement: a component netlist, a floorplan, and a physical layout. This results in a more precise timing estimation (fine grained attribute of *activity*), area or power consumption (fine grained attributes of *physical_component*), thus refining specification. Depending on the way of specification refinement through the phases I and II, the original resource specification version has to be conserved for later design verification, substituting the early resource binding links by equivalent links on the *physical_component* level of detail.

On the other hand, we assume that the given functionality will not change during the design process, rather than eventually be refined. This leads to the modeling of activities as constant design fixing objects during the design trajectory; however, the module net is changed into a net of interconnected physical components. It may be that the synthesis process of the architectural level completely reconfigures the resource net allocating and binding physical components from a library.

To achieve this, the binding has to be derived from activity-resource binding (dotted line) to an activity-component binding as expressed in Figure 4. This methodology shows some similarities to the binding approach of VHDL'93 [19], where a configuration can be specified by *primary binding indication* and later, in the description, be overwritten by *incremental binding indication.*

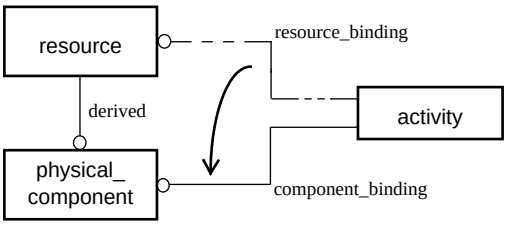


**Figure 4**: Activity-Component interaction

## 5. Experimental Schema Generation

Not all of the reference systems have been tested for information model extraction purposes.

System-level experiments were performed using of ADEPT [15] at the University of Virginia. Architecture-level experiments included STATEMATE[3] and different architectural level synthesis environments, by designing small exemplary objects like a data transmission rate detector and several kinds of multipliers. Details on the experimental work can be found in [20].

The related information model analysis shows the need for a compromise between

- the possibility of interactive control of design progress by navigating through the different graph representations,
- and the required conservation of the reference to

[3] trademark of i-Logix Inc., USA

the initial design specifications.

Furthermore, the object-oriented database system VERSANT[4] was used for implementing different interacting schemas. The runtime measurements on the scheme implementation indicate that the present available technology of commercial object-oriented databases is applicable for object-oriented design data representations, as required for applications in system and architecture level EDA.

## 6. Summary

Our aim was to integrate existing design representation work into a common core model. It is an initial effort towards the representation of a compound, integrated graph based model of design representation. Doing so, we found a harmonizing method to cope with the versioning problem during the design trajectory. As a further benefit, our compound graph representation methodology provides restriction of the applicable graph types on the upper design levels and their suitable extension during the refinement process.

We believe this work complements the existing CFI, EDIF, and VHDL information model. Future work will investigate generic applicability [6] of this information model to a wider domain of synthesis environments.

## References


[1] N.N.:“Product Data Representation and Exchange - Part 11: The EXPRESS Language Reference Manual”, ISO-Standard TC184/SC4 N151, Int. Organization for Standardization, Subcommittee 4, NIST-Secretary, 1992.

[2] Knapp, D. W., Parker, A. C.: “A unified Representation for Design Information”, in Koomen, C.F., Moto-oka, T. (eds.) “Computer Hardware Description Languages and their Applications” , Elsevier, 1985.

[3] Siepmann, E., Zimmermann, G.: “An Object-Oriented Datamodel for the VLSI Design System PLAYOUT”, Proc. Design Automation Conference, 1993.

[4] Lau, R.Y.W.: “Proposal for an Information Model for EDIF”, Technical Report UMCS-91-6-2, Dept. Computer Science, University of Manchester, U.K., 1991.

[5] CFI: “Design Representation, Electrical Connectivity Information Model and Programming Interface”, CAD Framework Initiative Inc., Version 0.9.4-071092, 1992.

[6] Bredenfeld, A., Camposano, R.: “Tool Integration and Construction using generated Graph-Based Design Representations”, Proc. of Design Automation Conference, 1995.

[7] Schürmann, B., Altmeyer, J., Schütze, M.: “On Modeling Top-Down VLSI Design”, Proc. IEEE International Conference on Computer-Aided Design ICCAD-94, 1994.

[8] Wendt, S.: “Nichtphysikalische Grundlagen der Informationstechnik, Interpretierte Formalismen”, Springer, 1989.

[9] Wendt, S., et al.: “Producing and Managing Technical Product Documentation for Users of the R/3 Basis System”, SAP Info, No. 40, SAP publication, October, 1993.

[10] Gajski, D.D., Vahid, F., Narajan, S., Gong, J.: “Specification and Design of Embedded Systems”, Addison Wesley, 1994.

[11] Narajan, S., Vahid, F., Gajski, D.D.: “System Specification with the SpecCharts Language”, IEEE Design of Computers, Dec. 1992.

[12] Vahid, F., Gajski, D.D.: “SLIF: A Specification-Level Intermediate Format for System Design”, Proc. European Design and Test Conference, 1995.

[13] Rundensteiner, E., A., Gajski, D.D.:“A Design Representation model for High-Level-Synthesis”, Technical Report, University California at Irvine, 1990.

[14] Becker, J., Buijs, F.: “Das SIR-Datenschema”, Report 34/94, Cadlab, University Paderborn, SNI, 1994.

[15] Kumar, S., Klenke, R. H., Aylor, J. H., Johnson, B. W. , Williams, R. D., Waxman, R.: “ADEPT: A Unified System Level Modeling Design Environment”, Proceedings of the 1st Annual RASSP Conference, pp. 114 - 123, 1994.

[16] Rammig, F.: “System Level Design” in Mermet, J.P. (ed.): “Fundamentals and Standards in Hardware Description Languages”, Kluwer, 1993.

[17] Giumale, C.A., Kahn, H.J.: “A Core Information Model of VHDL”, Proc. EuroDAC (same conference), 1995.

[18] Conradi. P., Dutt, N.: “A Compound Information Model for High-level Synthesis”, 4th International Working Conference on Electronic Design Automation Frameworks (EDAF), Gramado, Brazil, Nov. 28 - 30, 1994.

[19] Bergé, J.-M., et al.: “VHDL’92”, Kluwer, 93.

[20] Conradi. P.: “Information Analysis in High-level Synthesis”, Technical Report 94-39, Dept. Computer Science, University California, Irvine, 1994.


4. trademark of Versant Object Technology Corporation, USA