31371-1

Integrating a VHDL Dialect into the AMPLO Design Framework

por

Flávio Rech Wagner

RP nº 137   DEZEMBRO 1990

Trabalho desenvolvido com apoio do CNPq
e da IBM Brasil

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PóS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Av. Osvaldo Aranha, 99
90.210 - Porto Alegre - RS - BRASIL
Telefone: (0512) 271999
Telex: (051) 2680 - CCUF BR
FAX: (0512) 244164
E-MAIL: silvia@sbu.ufrgs.anrs.br
        PGCC@sbu.ufrgs.anrs.br

Correspondência: UFRGS-CPGCC
                 Caixa Postal 1501
                 90001 - Porto Alegre - RS - BRASIL

Editor: Ingrid E. S. Jansch Pôrto

Sistemas digitais - SBU/II
AMPLO
Linguagens: Descrição: Hardware

Contents

Abstract

This report compares the data representation and management models of the AMPLO design framework and of the hardware description language VHDL. It is argued that the AMPLO concepts are superior regarding important framework requirements, such as the tool integration process, the management of the design methodology and of the design refinement processes, and the efficiency of the design tools. The AMPLO data model is oriented to an extensible family of uniform languages, that are dedicated to various design levels and models. The main features of a VHDL dialect which holds the full language modelling range and fits the AMPLO data model are defined, so that the language can be integrated into the framework. The dialect imposes a more restricted discipline to the hierarchical decomposition of systems.

Sumário

Este relatório compara os modelos de representação e gerência de dados do ambiente de projeto AMPLO e da linguagem de descrição de hardware VHDL. É argumentado que os conceitos de AMPLO são superiores em relação a importantes requisitos de ambientes de projeto, tais como o processo de integração de ferramentas, a gerência da metodologia de projeto e do processo de refinamento de projeto e a eficiência das ferramentas de projeto. O modelo de dados do AMPLO é orientado a uma família extensível de linguagens uniformes, que são dedicadas a vários níveis e modelos de abstração. São estabelecidas as principais características de um dialeto VHDL que mantém todo o espectro de modelagem da linguagem e se adapta ao modelo de dados do AMPLO, de modo que a linguagem possa ser integrada ao ambiente. Este dialeto impõe uma disciplina mais restrita para a decomposição hierárquica de sistemas.

1 Introduction

CAD frameworks have been proposed in recent years to support the integration of suites of tools for different design levels. They should ideally guarantee automatic data consistency, offer mechanisms for uniformly integrating new tools, and support design and data management. Among them, we can mention ADAM, from USC [1], FACE, from GE [2], OCT, from Berkeley [3], the CADLAB Workstation CWS [4], and the IMEC open system architecture [5].

Hardware description languages are valuable tools in design environments. They are needed in order to cope with the growing complexity of digital systems. HDLs help design the system architecture from a high abstraction level and guide the synthesis process down to the physical implementation. VHDL [6] is an IEEE proposed standard which is gaining wide acceptance in industry and academia.

The AMPLO framework [7] [8] has been under development at the Federal University of Rio Grande do Sul, Brazil, since 1987. Currently, it integrates a family of uniform HDLs, that are oriented to discrete, structural and behavioral design levels, and offers a general, multi-level simulation mechanism for these levels.

This report compares the AMPLO and VHDL data representation and management models. It is shown how the AMPLO concepts are related to important framework requirements (the tool integration process, the management of the design methodology and of the design refinement processes, and the efficiency of the design tools). The AMPLO framework is oriented to an extensible family of compatible design languages that are specialized for the various abstraction levels. A VHDL dialect which holds the full language modelling range and fits the AMPLO data model is defined, so that it can be integrated into the framework. It imposes a more restricted discipline to the hierarchical decomposition of systems. A simulator for the dialect, to be integrated into the AMPLO multi-level simulation environment, is under development [9].

Although both systems have been already described elsewhere, a brief review of their underlying data models is first presented, in order to set the basis for the comparison.

2 The AMPLO data model

Every digital system is modelled in AMPLO as a **net of agencies** [10]. An agency can be a module of any complexity, from a single logical gate to a processor. This is a strong structural model. Agencies can communicate with each other only through their interface signals. Data types are assigned to all interface signals.

Agency descriptions can be either **primitive** or **composite**. AMPLO requires the assignment of a design level to a primitive agency as an attribute of it. Specialized HDLs define design levels. Constructs from different languages cannot be used inside the same primitive agency. Composite agencies are nets of occurrences of other agencies. Occurrences of primitive agencies that are described at various design levels can be used together in composite descriptions. In order to interconnect signals at the interfaces of different agencies, type compatibilities between signal data types of distinct design languages must be defined. Composite agencies are described through special language constructs, based on the CASCADE language [11].

In the current AMPLO implementation, three languages are available for primitive descriptions: LAÇO (a version of LASSO [12]), which supports behavioral descriptions through control graphs; KAPA (a version of KARL [13]), for structural RT descriptions; and NILO, oriented to the logic and switch levels. These languages have uniform syntactical and semantic structure.

Composite descriptions are stored in the database as complex objects [14] so that the database system explicitly handles all information about the modularity and hierarchy of design objects. The data structure representing the internal function of primitive agencies is not represented in the data model.

Each agency can have many associated **design alternatives**, corresponding to different interface definitions. For each design alternative, any number of **design versions** can be described. Versions can be either primitive or composite descriptions.

Composite descriptions can contain either occurrences of alternatives (**dynamic configurations** [15]) or occurrences of versions (**static configurations**). In the former case, versions must be assigned to the occurrences before the agency description can be used by some design tool.

In a top-down design approach, alternatives can temporarily remain without associated design versions. An agency, with its first alternative, can be declared inside the composite description which instantiates it.

The version concept of AMPLO is used with three different purposes: a) representations of a module at various design levels; b) multiple implementations for a module (e.g. a fast ALU, a small ALU, etc); c) design evolution in time, i.e., consecutive improvements of the same implementation. Other environments, such as the DAMASCUS system [16], offer separate management concepts for supporting these distinct situations.

3 The VHDL data model

VHDL is a hardware description language developed under a DoD contract and proposed as a standard by the IEEE. Several commercial products supporting the language are now available.

Design objects in VHDL are modeled as **design entities**, that are described through an **interface** and one or several **architectural bodies**, corresponding to many possible representations or implementations for them. Three different design styles are available in describing architectural bodies. The **behavioral** style describes an entity as a collection of communicating processes. The **dataflow** style uses concurrent assignment statements. The **structural** style describes an interconnection of components, that are occurrences of other entities. These three design styles can be used together inside an architectural body. **Regular** descriptions can be generated through iteration. Classes of similar entities can be defined through **generic parameters**, to which values are assigned when the entity is instantiated. Generics can be used to describe variable structures, such as interface signals with variable bit lengths and regular structures of variable size.

Components used in an architectural body can be bound to a given design entity through either a configuration specification inside this body or a separate **configuration body**. The architectural body can be **partially bounded**, if only a design entity is selected for a component; **fully bounded**, if also an architectural body of the design entity is selected; or **open**, if the binding is done later through a configuration body.

4 Comparing the design languages

VHDL has been defined as a unique language for supporting many design dimensions and levels, whereas AMPLO supports an extensible family of languages, each one specialized for a distinct design level. While in VHDL the user can describe an architectural body by mixing three different design styles, a specific language (or level) must be assigned to each AMPLO object version. This restriction should not be meant as a drawback of the AMPLO languages. VHDL has in reality only two built-in "primitive" design levels (behavior and dataflow), since its structural style corresponds to the composite versions of AMPLO. Other primitive levels can be defined in VHDL through user-defined data types, which have no intrinsic semantics, thus resulting in lesser tool efficiency. This easy-to-use and powerful extension method, although very useful for obtaining more modelling power, can lead to a proliferation of dialects, each one representing a particular design level which is defined for a more efficient tool building. A synthesis program which accepts any VHDL description as input, for example, is today still unavailable. Synthesis packages have been implemented, however, for special VHDL subsets or modelling strategies. As another example, VHDL doesn't have built-in primitive RT structural constructs, as KARL [13], which would be ideal for expressing the outcome of a high level synthesis program.

AMPLO, in turn, allows the integration of any number of problem-specific languages, that can be defined to simultaneously achieve modelling power and maximal efficiency of the design tools. The proliferation of languages (or dialects) is as possible in AMPLO as in VHDL, since the user tends to prefer language constructs and models that are problem-oriented. The fundamental differences between both language approaches lie in two facts. First, the set of languages in AMPLO is known by the design environment, which can thus use this knowledge for management purposes, as discussed in the next section. Second, the AMPLO languages have built-in primitives, that are not implemented by user-defined data types and operations. This approach promotes language efficiency at the cost of implementation complexity.

Although it does not offer any mechanism for defining new design languages with uniform grammar and semantics, AMPLO does not restrict the use of such mechanisms. The CONLAN approach [17] could be used for that purpose, as it has been done for building the CASCADE [11] family of languages.

The languages that are now implemented in AMPLO have been defined, however, with the restricted goal of validating the framework principles. Therefore, they lack offering more powerful language mechanisms. Packages and regular structures (of fixed size), that are for example supported in VHDL, would be valuable in extending the modelling power of the languages and their user-friendliness.

5 Comparing the data models

Table 1 summarizes the relationships between the AMPLO and VHDL data representation and management concepts.

CONCEPTS	VHDL	AMPLO
Main design objects	Design entities	Agencies
Associating many interfaces to the same design object	Not possible	Alternatives of an agency
Associating many representations to the same design object	Architectural bodies of an entity	Versions of an alternative
Static configurations	Full binding: selection of architectural bodies of design entities	Instances of versions
Dynamic configurations	Partial binding: selection of design entities	Instances of alternatives
Open configurations	Components without binding	Not available
Mixing structure with primitive constructs in a design object description	Possible	Not possible
Defining the design level of an object representation	Not possible	Mandatory
Parameterized objects (including variable structures)	Available (generics)	Not available

Table 1: Relationship between data management concepts of AMPLO and VHDL

The AMPLO data model shows two main modelling restrictions when compared to VHDL. First, "structural" descriptions cannot be used together with primitive constructs from some design level inside the same object description. Second, a specific language (level) must be associated with each primitive object description, so that constructs from distinct levels cannot be mixed.

These restrictions have been defined so that the data model better responds to important framework requirements, namely the tool integration process, the design methodology management, the design refinement process management, and the efficiency of the design tools. This last issue has been already discussed in the previous section. The other topics are considered below.

5.1 Tool integration

The AMPLO data model shows "coarse" granularity. It is based on general objects (agencies), which do not have intrinsic semantic power, and handles only external agency attributes (name, interface signals, design level) and interconnections between agencies in composite descriptions. The integrity constraints related to the design primitives are checked by the tools. The AMPLO database is accessed through an object-oriented interface [18], which allows the manipulation of design objects (agency alternatives and versions), while maintaining integrity constraints that relate them to each other.

The coarse granularity of the AMPLO data model implies a "loose" tool integration, as in the IMEC Open Architecture [5]. The data model is not influenced by the design levels at which primitive agencies can be described. A loose tool integration mechanism avoids a severe re-definition of the data schema when a new design level, with its repertoire of primitive constructs, is integrated.

Although the AMPLO framework does not yet offer high level user interface facilities for allowing an easy extension of the design environment, when a new design level (language) is to be integrated, they could be provided at a low implementation cost. A new language should be inserted into a list of available design levels and, eventually, new interface signal data types should be declared, together with their compatibility with already defined data types.

A "tight" tool integration would be possible with a "fine grain" data model, which expresses design primitives such as registers and gates and relationships between them. In this case, each tool access to a data item is mapped into database accesses. The database system can check integrity constraints related to the design primitives. In order to integrate a new design level, the data model must be severely extended. A fine grain data model can be specified only for geometric and structural representations, because it is almost impossible to establish relationships involving behavioral primitives.

Since the structural and functional design styles can be used together inside an architectural body, a fine grain data model cannot be specified for VHDL. Furthermore, the VHDL language extension mechanism, based on user-defined constructs, avoids the definition of a semantically rich coarse data model.

5.2 Design refinement management

In VHDL, components can be incrementally added to a behavioral description. Component ports can be connected to any signals declared inside the architectural body, also when these signals have no structural meaning. Therefore, an underlying data model could not exactly represent the system hierarchy, including the precise interconnections between signals at the interfaces of components, and the associated database system could only partially control the consistency of the structural refinement process. This is not the case in AMPLO, where the data model forces a complete structural decomposition in a one-step procedure.

```
architecture structure_1 of traffic_light_controller is
  signal ...;
  signal Start_Timer : Data_Type := '0';

  component Timer_Section
    generic (...);
    port (Start : in Data_Type; ...);
  end component;

begin
  Controller_Process:
  process
  begin
    case ... is
      when ... =>
        if ... then
          Start_Timer <= transport not Start_Timer;
        end if;
  ...
    end case;
  ...
  end process;

  Timer_Struct : Timer_Section
    generic map (...)
    port map (Start_Timer, ...);

end structure_1;
```

Figure 1: Mixing structure with behavior in VHDL

The VHDL fragment in Figure 1, extracted from [19], illustrates this point. The architectural body *structure_1* of the entity *traffic_light_controller* contains a component *Timer_Struct* which is bound to the entity *Timer*. A component port of *Timer_Struct* is connected to signal *Start_Timer*, whose value is set in behavioral statements of a process declaration, which is not structurally defined inside the architectural body.

As an additional point, AMPLO does not allow regular structures of variable length. Although this is a powerful modelling mechanism, it complicates the management of the design refinement process and inhibits the exact representation of the design hierarchy.

5.3 Design methodology management

A **design methodology** [20] is a sequence of transformations in a design space which is defined by three axes (behavior, structure, and geometry). Points in the design space correspond to abstraction levels. **Design managers** have been proposed to enforce particular design methodologies (e.g. in the CMU Cadweld [21] and USC ADAM [22] environments).

Research is being carried on in order to support in AMPLO the definition of design methodologies through the specification of integrity constraints to be automatically verified by the database system. Such a scheme for implementing design managers has also been proposed in [23], based on an event-triggering mechanism.

In AMPLO, a design methodology to be specified with such a technique can only refer to the values of agency attributes that are externally visible. The main visible attribute of an AMPLO object is its description level. The example in Figure 2 shows a script, inspired by the DECOL (Design Control Language) [24] templates of the OASIS system from MCNC, which defines the initial part of a design methodology based on the AMPLO data model. In this script, the '*' is to be replaced in a particular design by an object name. The '$' means that any version can be selected for the referenced object. Design level identifications (or 'comp' for composite descriptions) are used as suffixes for the object names.

```
PROGRAM  DataPathSynthesis
INPUT    *.LACO
OUTPUT   *_dp.KAPA

PROGRAM  ControlFlowSynthesis
INPUT    *.LACO, *_dp.KAPA
OUTPUT   *_cf.LACO

PROGRAM  Composition
INPUT    *_dp.$, *_cf.$
OUTPUT   *.comp
```

Figure 2: Design methodology management in AMPLO

Suppose one wants to design a microprocessor *mp*, which is initially specified by *mp.LAÇO*. A program DataPathSynthesis generates, from this description, an object *mp_dp.KAPA*, which contains a structural RT description. From the original specification *mp_LAÇO* and the data path description, a program ControlFlowSynthesis generates *mp_cf.LAÇO*, which contains a control flow graph. A composition tool finally creates a composite version for *mp*, containing occurrences of *mp_dp* and *mp_cf* (any versions of both objects could be selected).

Although appropriate for coarse data models, such design methodology control cannot be achieved in VHDL. Since the language permits the use of many description styles inside the same architectural body, the design level of an object cannot be identified.

5.4 Configuration management

VHDL is superior to AMPLO with regard to configuration management. It supports open configurations, which have no counterpart in AMPLO, but this feature could be added to AMPLO without conflicting with its remaining data representation and management concepts. Generics could also be supported in AMPLO, but in a restricted way: since the AMPLO data model does not handle variable structures, generics could not be used with such a goal.

6 Definition of a VHDL dialect for AMPLO

A VHDL dialect, called rVHDL (for "restricted" VHDL) is defined in order to be integrated into the AMPLO framework. Essentially, the language must match the AMPLO data model. This is accomplished by imposing to VHDL a main modelling restriction: the functional (behavior and dataflow) and structural description styles must be used in two separated, different types of architectural bodies. The keyword "architecture" is replaced by "structure" or "function" in the body identification, depending on its description style.

Functional descriptions can contain the full repertoire of statements that are related to the behavioral and dataflow styles, but cannot use components. The structural descriptions cannot use behavioral and concurrent statements, only component declarations and instantiations.

Since VHDL entities have interfaces attached to them, they correspond in fact to AMPLO alternatives. Entity names are thus always followed by an alternative number. The rVHDL analyzer must recognize identical entity names and associate the corresponding alternatives to the same agency.

Because AMPLO does not support open configurations, configuration declarations must necessarily be added to the structural descriptions. At least an entity name must be bound to a component, but an architectural body does not need to be selected (this corresponds to the AMPLO dynamic configurations). As an additional restriction, generics cannot be used with the purpose of declaring variable structures.

Since AMPLO identifies design versions by numbers, the rVHDL analyzer must maintain separate tables for converting architectural body identifiers into version numbers.

Declarations and statements that can be contained in the VHDL entity declarations are stored as AMPLO user-defined attributes, that are associated with agency alternatives. These attributes are handled by the rVHDL analyzer.

The rVHDL analyzer must also be charged of the management of packages. These objects, which can be used by several entities, cannot be effectively integrated into the AMPLO database system without changing its associated data model.

Although the languages that are already integrated in AMPLO have a specific, uniform grammar, the basic VHDL grammar has not been changed. So, for example, the designation 'entity' is still used instead of 'agency'.

The use of rVHDL is illustrated by showing its impact on the design example presented in [19]. An agency *TLC* ('traffic_light_controller') is to be designed. The initial specification for *TLC* (see Figure 3) creates the agency with its first design alternative (an interface definition). There are no versions for this agency / alternative at this moment.

A first version is then created by the architectural body ***specification*** (see Figure 4). It is a primitive description which uses both processes and concurrent

```
entity TLC.1 is
  generic (...);
  port (...);
end TLC.1;
```

Figure 3: Definition of agency TLC

assignments.

```
function specification of TLC.1 is
  signal ...;
begin

  Controller_Process:
  process
  begin
    ...
  end process;

-- concurrent assignments

  Timer_Process:
  process
  begin
    ...
  end process;

end specification;
```

Figure 4: Initial functional specification of TLC

The agency is then partitioned into two components, that are bound to alternatives of two new agencies, *Timer* and *TL_Controller* (see Figures 5 and 6). AMPLO supports a top-down approach as VHDL. Therefore, both agencies could have been declared and created (together with their first design alternatives) later in the component declaration inside the *TLC* description. Since a composite version can refer to occurrences of alternatives, these two new agencies do not need to have versions when they are instantiated inside *TLC*.

Before the new *TLC* description can be simulated, however, these versions must be created. Each agency will have a primitive version, which in fact does not add new design information to the first system specification, since they only encapsulate functions that were already specified (see Figures 7 and 8). Because the *struct* version of *TLC* did not select versions for the agencies that are bound to its components, this must be made through a configuration body.

```
entity Timer.1 is                  entity TL_Controller.1 is
  generic (...);                     port (...);
  port (...);                      end TL_Controller.1;
end Timer.1;
```

Figure 5: Agencies used in the partitioning of TLC

```
structure struct of TLC.1 is
  signal ...;

  component Timer_Section:
    generic (...);
    port (...);
  end component;

  component Controller_Section:
    port (...);
  end component;

  for Traffic_Light : Controller_Section use
    entity TL_Controller.1
      port map (...);
  end for;

  for Timer_Struct : Timer_Section use
    entity Timer.1
      generic map (...);
      port map (...);
  end for;

end struct;
```

Figure 6: Structural partitioning of TLC

```
function Behavior of Timer.1 is
begin

  Timer_Process:
  process (Start)
  begin
    ...
  end process;

end Behavior;
```

Figure 7: Functional specification of Timer

```
function Behavior of TL_Controller.1 is
  signal ...;
begin
  Controller_Process:
  process
  begin
    ...
  end process;
-- concurrent assignments

end Behavior;
```

Figure 8: Functional specification of TL_Controller

The partitioning method adopted in [19] could not be used here. It first encapsulated the function of *Timer* inside a component, and then created a first structure for *TLC*. In a second step, also the *Controller* function was isolated into another component. The first step cannot be replicated in AMPLO, because the *TLC* description (see Figure 1) mixes the structural and behavioral styles. This modelling restriction can be partially avoided. The incremental addition of structure can be "simulated", if, at each new design step which introduces structural information, the remaining behavior is encapsulated within an auxiliary component. Besides this restriction, however, the design process using rVHDL is essentially the same as in VHDL, because AMPLO supports concepts that are very similar to entities, architectural bodies, and components.

The example also shows that rVHDL holds the full modelling range of VHDL, since all behavioral statements are still available. VHDL user-defined constructs can still be used, if handled by the rVHDL analyzer, without knowledge of the AMPLO database system. Also VHDL packages can be used in such a way, although it would be surely better that the AMPLO database handles the relationships between entities and packages.

7 Final remarks

The AMPLO data model has been defined according to important design framework requirements:

- to permit the enforcement of specific design methodologies,
- to allow an easy integration of new tools, specially for new design levels, and
- to control the structural design refinement process.

This model is based on two main modelling properties:

- the explicit separation between primitive and composite objects, and
- the explicit assignment of a design level (or language) to each primitive object description.

The framework is thus oriented to a family of dedicated, compatible design languages, that have more intrinsic semantic power and allow the construction of more efficient design tools than systems based on a unique language. The framework also offers a general simulation mechanism for integrating discrete levels [25] [26], which is consistent with the data model properties.

Since VHDL does not fit these data representation and management concepts, a dialect rVHDL has been defined in order to be integrated into AMPLO. This dialect holds the full modelling range of VHDL, but imposes a more restricted discipline to the hierarchical decomposition of systems. The development of a rVHDL simulator to be integrated into the AMPLO simulation environment is now underway.

The report emphasized the properties of the AMPLO data representation and management models. On the other hand, VHDL is a much more powerful language than the AMPLO family of languages, that have been defined with the restricted purpose of validating the framework principles.

References

[1] J. Granacki, D. Knapp, and A. Parker. The ADAM Advanced Design Automation System: overview, planner, and natural language interface. In *22nd Design Automation Conference*, ACM/IEEE, 1985.

[2] W.D. Smith et al. FACE Core Environment: the model and its application in CAE/CAD tool development. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[3] D.S. Harrison et al. Data management and graphics editing in the Berkeley Design Environment. In *International Conference on Computer Aided Design*, IEEE, 1986.

[4] K. Gottheil et al. The CADLAB workstation CWS – an open, generic system for tool integration. In F.J. Rammig, editor, *Tool Integration and Design Environments*, North-Holland, 1988.

[5] L. Claesen et al. Open framework of interactive and communicating CAD tools. In F.J. Rammig, editor, *Tool Integration and Design Environments*, North-Holland, 1988.

[6] *IEEE Standard VHDL Language Reference Manual.* IEEE, New York, 1988.

[7] F.R. Wagner, C. Freitas, and L.G. Golendziner. The AMPLO system – an integrated environment for digital systems design. In F.J. Rammig, editor, *Tool Integration and Design Environments*, North-Holland, 1988.

[8] L.G. Golendziner and F.R. Wagner. Modeling digital systems as complex objects. In *9th International Symposium on Computer Hardware Description Languages and their Applications*, IFIP, 1989.

[9] J.H. Rech. *Um Simulador VHDL para o Sistema AMPLO.* Master's thesis, CPGCC / UFRGS, 1990. (under development).

[10] S. Wendt. On the partitioning of computing agencies into communicating agencies. In *GI-NTG Fachtagung - Struktur und Betrieb von Rechensystemen*, Springer-Verlag, 1980.

[11] D. Borrione and C. LeFaou. Overview of the CASCADE multi-level hardware description language and its mixed-mode simulation mechanisms. In *7th International Symposium on Computer Hardware Description Languages and their Applications*, IFIP, 1985.

[12] D. Borrione and J.F. Grabowiecki. Informal introduction to LASSO – a language for asynchronous systems specification and simulation. In *EURO-IFIP*, 1979.

[13] R. Hartenstein. *Fundamentals of Structured Hardware Design.* North-Holland, 1977.

[14] R.L. Haskin and R. Lorie. On extending the functions of a relational database system. In *SIGMOD Conference*, ACM, 1982.

[15] R.H. Katz et al. Design version management. *IEEE Design & Test*, February 1987.

[16] J.A. Mulle, K.R. Dittrich, and A.M. Kotz. Design management support by advanced database facilities. In F.J. Rammig, editor, *Tool Integration and Design Environments*, North-Holland, 1988.

[17] R. Piloty and D. Borrione. The CONLAN project: concepts, implementations, and applications. *IEEE Computer*, February 1985.

[18] K. Becker and L.G. Golendziner. Database support for a CAD environment for digital systems design. In *8th International Conference on Computer Science*, SCCC, Santiago do Chile, 1989.

[19] R. Lipsett, C. Schaefer, and C. Ussery. *VHDL: Hardware Description and Design.* Kluwer Academic Publishers, 1989.

[20] D. Gajski and R. Kuhn. Guest's editors introduction. *IEEE Computer*, December 1983.

[21] J. Daniell and S.W. Director. An object-oriented approach to CAD tool control within a design framework. In *26th Design Automation Conference*, ACM/IEEE, 1989.

[22] D. Knapp and A. Parker. A design utility manager: the ADAM planning engine. In *23rd Design Automation Conference*, ACM/IEEE, 1986.

[23] A.M. Kotz, K.R. Dittrich, and J.A. Mulle. Supporting semantic rules by a generalized event / trigger mechanism. In *International Conference on Extending Data Base Technology*, Venice, 1988.

[24] K. Kozminski. *Design Control in MCNC's Open Architecture Silicon Implementation System OASIS.* Technical Report TR89-54, MCNC, 1989.

[25] F.R. Wagner. Um ambiente integrado para a simulação de sistemas digitais. In *IV Simpósio Brasileiro de Concepção de Circuitos Integrados*, SBC, Rio de Janeiro, 1989.

[26] J.P. Figueiró. Simulador mestre para o sistema AMPLO. Departamento de Informática Aplicada, UFRGS, 1990. (Trabalho de Diplomação).

Relatórios de Pesquisa

RP-137: "Integrating a VHDL Dialect into the AMPLO Design Framework", dezembro 1990.
F.R. WAGNER

RP-136: "O Ambiente de Execução do Experimento em Programação Diversitária", novembro 1990.
R. CANANI; S.P. ZANI

RP-135: "Teste Prático do Circuito MPC e o seu Ambiente Técnico", novembro, 1990.
L. ROISENBERG; T.V. WAGNER; D.A.C. BARONE

RP-134: "SÍNTESE AUTOMÁTICA DE PARTES OPERATIVAS - Ferramentas resultantes da implementação do protótipo do sintetizador automático de partes operativas (SAPO), agosto 1990.
C. De ROSE; J.L.S. JÚNIOR

RP-133: "Preliminar Thoughts on Agents and their Development", novembro 1990.
A.C.R. COSTA

RP-132: "Towards a Complete Conceptual model: Petri Nets and Entity-Relationship", agosto 1990.
C.A. HEUSER; E.M. PERES

RP-131: "PC como Terminal do ED680 (Uso e Implementação)", agosto 1990.
S.D. OLABARRIAGA

RP-130: "Ferramenta para Apoio à Análise de Requisitos e Modelagem de Sistemas de Banco de Dados", julho 1990.
M.H. YAMAGUTI; S. LOH; J.M.V. CASTILHO

RP-129: "Biblioteca de Células Tranca. Regras CMP - Parte II", julho 1990.
A. REIS; E. E. GIFFONI; F. MORAES; R. REIS

RP-128: "Nets of Places and Links: a coherent presentation of Petri Nets for systems modeling", Julho 1990.
G. RICHTER; C.A. HEUSER

RP-127: "Integração do Método T.A.& A. e do Ambiente YPY na Especificação e Prototipação de um SIE", junho 1990.
N. EDELWEISS; J.P.M. OLIVEIRA

RP-126: "Uma Abordagem para Prototipação de Sistemas de Banco de Dados", junho 1990.
H. STROGULSKI; J.M. CASTILHO; O.J.C. SOUZA

RP-125:	"Edição Revisada da Biblioteca de Células Tranca", maio 1990. F.G. MORAES; A.I. REIS; E.E. GIFFONI; M.S.LUBASZEWSKI; R.F. GOMES; R.A.L. REIS
RP-124:	"Controle de Aprendizagem no Nível do Meta--Conhecimento", abril 1990. F.M. OLIVEIRA
RP-123:	"Documentação da Concepção de Percanta, Memória Ram Estática de 1/4 kbits", março 1990. J.L.K. HOLSBACH; L.O.S. FREIRE; R.P. RIBAS e D.A.C. BARONE
RP-122:	"Seminário de Epistemologia da Inteligência Artificial, 1989: IA e Racionalidade", fevereiro de 1990. A.C.R. COSTA
RP-121:	"GAMINE: Gerenciador para um Ambiente de Minimização e Edição de Funções Combinacionais", janeiro de 1990. S.J. RIGO; S.P. ZANI e T.S. WEBER
RP-120:	"Necessidades Básicas em Sistemas Operacionais Tempo Real", dezembro de 1989. R.M. FRICKS e I.J. PÔRTO
RP-119:	"O Modelo e Revisado", dezembro de 1989. C.S. DOS SANTOS
RP-118:	"Descrição Comportamental de Hardware Concorrente: Uma Introdução a BABEL", dezembro de 1989. T.S. WEBER
RP-117:	"Um Algoritmo de Reconhecimento Incremental", outubro de 1989. L.G. ESPERANÇA; E.L. FAVERO; R.T. PRICE
RP-116:	"Documentação da Concepção de PEALO (CADM1C) um Circuito Integrado Pré-Difundido Controlador de Acesso Direto à Memória de um Canal para Sistemas Baseados no Microprocessador 8085A", outubro 1989. L.O.S. FREIRE; J.L. GÜNTZEL; R.P. RIBAS, A.L. COSTA, A.A.M. FRÖHLICH, D.A.C. BARONE
RP-115:	"Vetor Descrição do Projeto de Hardware de um Sistema Multimicroprocessador para Processamento Numérico", agosto 1989. F.R. NASCIMENTO
RP-114:	"Introdução do Tempo no Ambiente para Desenvolvimento de Bancos de Dados Dedutivos", julho 1989. N. EDELWEISS