CEIS based on relational database systems.

All architectural components on the first (internal) level are generic, i.e. they are reusable for implementing other CEIS. The internal relational schema can be generated by RAD[1] tools for relational database design. For COPIA we designed the first version of the CDM with RONDO[11] in order to generate the initial relational schema.

The components of the conceptual level are specific to the application domain. They have to be implemented for each CEIS anew. However, like in the case of the conceptual schema transformations, their functionality can easily be based on the functionality of the generic components. In general, there are different applications on the external level to cover separate aspects of the required functionality of a CEIS.

6. Conclusions and Related Work

This case study shows that a meta schema approach combined with a well-defined set of schema transformations is a practical way to handle evolution in complex engineering information systems. Yet, due to our specific application domain, we could use a number of simplifying assumptions in order to reduce the complexity of the general evolution problem. We assumed that reorganization of circuit data takes place directly after each schema modification session. COPIA always accesses the newest version of the schema. Therefore, there is no need to provide access to instances of different versions of the schema, i.e. to support *backward* and *forward compatibility.* An extensive overview on these aspects of schema evolution is given in [13]. In contrast to *horizontal evolution* (cf. Figure 8) [3] also considers *vertical evolution*, i.e. the integration of single databases into a global federated information system. This approach is based on database view mechanisms to support evolution. [14] tries to prove the completeness of a set of structural schema transformations. However, they do not consider instance mappings and [13] shows examples for schema modifications that cannot be performed by the proposed set of transformations. Our approach is to start with a rather small set of predefined primitive transformations as a basis to constitute more complex application specific transformations but allowing to add further primitive transformations when necessary.

Acknowledgements

We would like to thank Wilhelm Schäfer and Albert Zündorf, Paderborn University, for many fruitful discussions and their great support in editing this paper.

1. Rapid Application Development

References

[1] D. Wagenblaßt and W. Thronicke. An Approach for Classification of Integrated Circuits by a Knowledge Conserving Library Concept. In *Proc. of European Design Automation Conference with EURO-VHDL 1995*. Brighton, UK.

[2] D. Wagenblaßt and W. Rissiek. Layoutanalyse analoger Schaltungen basierend auf einem erweiterten Bibliothekskonzept. In *3. GME / ITG-Diskussionssitzung Entwicklung von Analogschaltun-gen mit CAE-Methoden, Band 3*. 1994. Bremem, Germany.

[3] M. Tresch. Evolution in Objektdatenbanken: Anpassung und Integration bestehender Informationssysteme. Stuttgart; Teubner, 1995

[4] M. L. Fussel. Foundations of Object Relational Mapping. 1220 N. Fair Oaks Ave, #1314, Sunnyvale CA 94089. 408.734-9068. 1997

[5] J. Rumbaugh and M. Blaha and W. Premerlani and F. Eddy and W. Lorensen. Object--Oriented Modeling and Design. Prentice Hall. Englewood Cliffs, N. J. 07632. 1991

[6] U. Nickel. Konzeption einer objektorientierten Bibliothek für gemischt analog/digitale Schaltkreise. Diplomarbeit. Universität Paderborn. Fachbereich 17. 1997

[7] C. Batini and S. Ceri and S. B. Navathe. Conceptual Database Design. Benjamin Cummings; Redwood City, CA, 1992

[8] J.-L. Hainaut. Entity-Generating Schema Transformations for Entity-Relationship Models. *Proc. of the 10th Entity-Relationship Conference*, San Mateo, 1991

[9] A. Schürr and A. J. Winter and A. Zündorf. Graph Grammar Engineering with PROGRES. In W. Schäfer (ed.) *Software Engineering - ESEC '95*. Springer Verlag, 1995

[10] D. Tsichritzis and A. Klug (ed.). The ANSI/X3/SPARC DBMS Framework, AFIPS Press, 1978.

[11] RONDO EE/R-Editor Benutzerhandbuch, Version 2.2g, PRODV Software GmbH, Martin-Schmeißer-Weg 14, 44227 Dortmund, Germany. 1994

[12] D. Wagenblaßt. Vision eines Analysesystems zur Beurteilung komplexer analog/digitaler Schaltungen hinsichtlich der Funktionssicherheit und der elektromagnetischen Verträglichkeit. *C-LAB, Technical Report 08/97,* Paderborn, Germany. 1997

[13] B. Schiefer. Eine Umgebung zur Unterstützung von Schemaänderungen und Sichten in objektorientierten Datenbanksystemen, *Dissertation des Forschungs-zentrum Informatik (FZI)*. ISSN 0944-3037. Karlsruhe, Germany. 1994

[14] B. Banerjee, W. Kim, H.-J. Kim and H. F. Korth. Semantics and Imlementation of Schema Evolution in Object-Oriented Databases, In *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*. pp. 311-322. San Francisco, USA. 1987.

If a graph rewriting rule is applicable, it replaces its upper side by its bottom side. In this sample rule nodes ‘1, ‘2, and ‘3 are identically replaced and the *declares* relationship between node ‘2 and node ‘1 is replaced by another *declares* relationship between node ‘3 and node ‘1. In analogy, graph rewriting rules can be used to formally specify instance mappings of transformations as well. However, a detailed description of the expressiveness of graph rewriting theory is out of the scope of this paper and can be found in [9].

Based on the set of predefined primitive schema transformations we defined more complex application specific schema transformations. Typically, these so-called *conceptual schema transformations* are build by concatenated primitive transformations. They often are more restrictive according to the specific semantics of the application’s schema. An important benefit of this approach is that the instance mapping of conceptual transformations does not have to be defined anew because it is already prescribed by the concatenation of the instance mapping of the constituted primitive transformations.

Conceptual Transformation	IA	IP	IR
Create Variant Component	√		
Merge Variant Components			√
Cereate Typical Pin		√	
Create Necessary Pin			√
Create Forbidden Pin			√

Figure 12. Samples for Conceptual Transformations

Figure 12 shows several conceptual transformations of COPIA to modify a given CDM. For example, the first transformation (*CreateVariantComponent*) creates a variant class of a given class in the CDM. It is based on the concatenation of primitive transformations *GeneralizeClass*, *MoveUpAttribute*, and *SpecializeClass*.

An example application of *CreateVariantComponent* to an original class *SignalOutputPin* is given in Figure 13.

At first a new superclass *SignalPin* is created. Then it moves the declaration of the attributes *resistance*, *capacitance*, and *operatingRange* to the new superclass by repeatedly applying the primitive transformation *MoveUpAttribute*. Finally, it introduces a new subclass *SignalInputPin* of the class *SignalPin*.

5. Lessons Learned

In addition to the requirements given in Section 2.3, COPIA’s design was driven by the demand for reusibility. Hence, we tried to distinguish generic from application

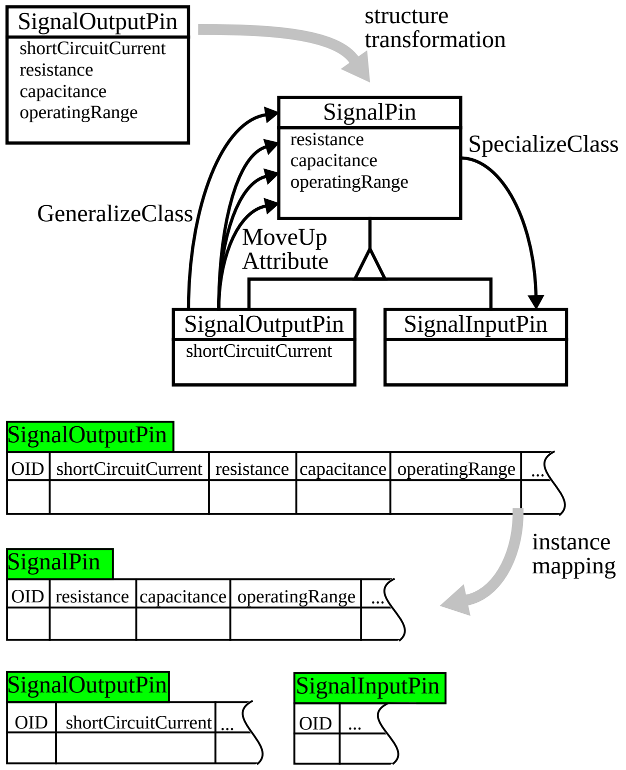

Figure 13. Conceptual Transformation *CreateVariantComponent*

specific components in order to derive a general architecture for complex evolutionary information systems. According to our experiences with COPIA, we derived the following reference architecture which is given in Figure 14. In fact, the proposed architecture is an extension of the general three level architecture for information systems (ANSI/ SPARC)[10] which reflects the specific requirements of

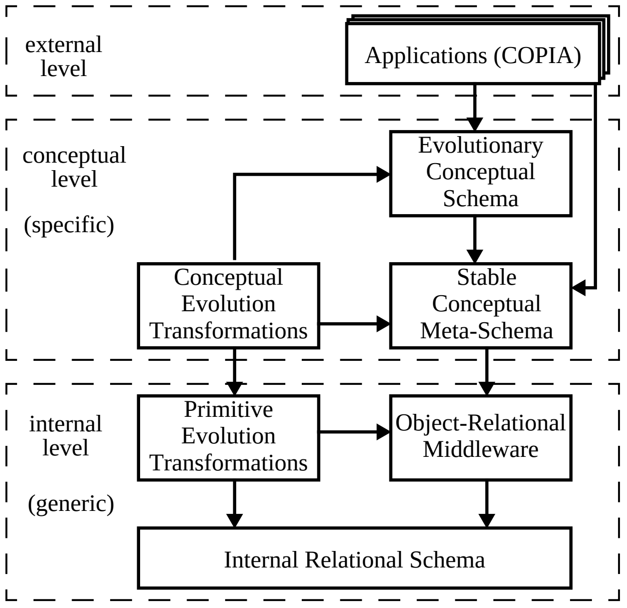
external level
Applications (COPIA)
conceptual level
(specific)
Evolutionary Conceptual Schema
Conceptual Evolution Transformations
Stable Conceptual Meta-Schema
internal level
(generic)
Primitive Evolution Transformations
Object-Relational Middleware
Internal Relational Schema

Figure 14. CEIS Reference Architecture

Primitive Transformation	IA	IP	IR
Specialize Class	√		
Generalize Class	√		
Delete Class			√
Add Association	√		
Delete Attribute			√
Move Up Attribute	√		
Set Attribute Univalent			√
Change Default Value		√	

Figure 9. Samples for Primitive Transformations

the context of schema evolution: it allows to determine when the data has to be reorganized according to an applied transformation and whether an applied transformation can be compensated. Generally, a reorganization has to be performed after each IA transformation because it cannot be simulated. This is in contrast to IR and IP transformations. On the other hand, unlike IA and IP transformations, IR transformations cannot be compensated.

In COPIA we distinguish between primitive and conceptual schema transformations. This distinction stems from the fact that several schema transformations can be combined in order to built new complex schema transformations. Thus, our approach is to define application specific (conceptual) schema transformations based on a relatively small number of predefined (primitive) transformations. Figure 9 shows several samples for primitive schema transformations including their classifications. An application of a primitive transformation (*MoveUpAttribute*) is illustrated in Figure 10. This IA transformation moves the declaration of a certain attribute from a given class to its superclass. In the depicted sample situation the structure mapping moves the attribute *capacitance* from the class *ReferencePin* to the class *ControlPin*.

Consequently, the instance mapping migrates the data about the capacitances of reference pins to the relation *ControlPin*. Now every instance of *FuntionControlPin* (and other subclasses of *ControlPin*) has a capacitance attribute, too. Thus, the instance mapping has to prompt the knowledge engineer for a default capacitance value for these instances.

We formally specify primitive schema transformations using graph rewriting rules[9]. Figure 11 shows the graph rewriting rule that specifies the structure transformation of *MoveUpAttribute, e.g.* Generally, a graph rewriting rule consists of an upper side that corresponds to the precondition *p* of a structure mapping and a bottom side that represents its postcondition *q*. The upper side of the sample

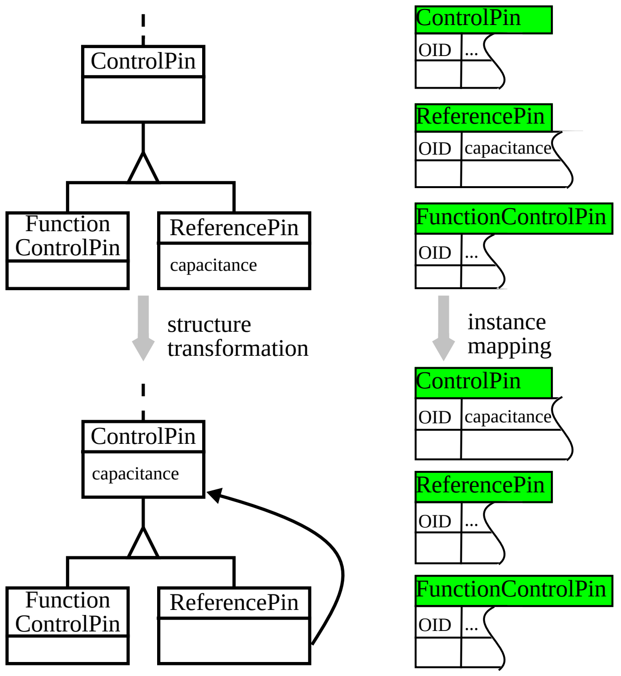

Figure 10. Primitive Transformation *MoveUpAttribute*

rule defines that *MoveUpAttribute* is applicable to a given attribute *attr* if it is declared in a class *c* that has a superclass. Furthermore, this superclass must not have a descendent class (other than *c*) that declares another attribute with the same name. At this, the cancelled attribute (node ‘4) represents a negative condition, i.e. such an attribute must not be present in the original schema. The bold arrow between the superclass (node ‘3) and the forbidden attribute denotes a path expression. It references all attributes which are declared in classes that are in the transitive closure of all subclasses of node ‘3.

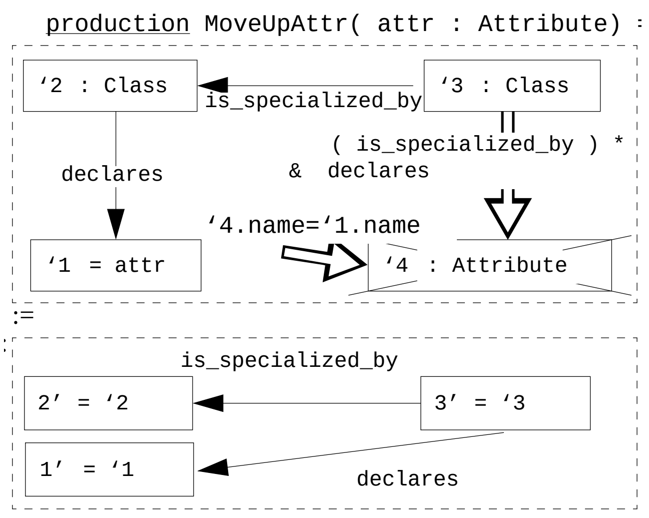

Figure 11. Structure Rewriting Rule for Primitive Transformation *MoveUpAttribute*

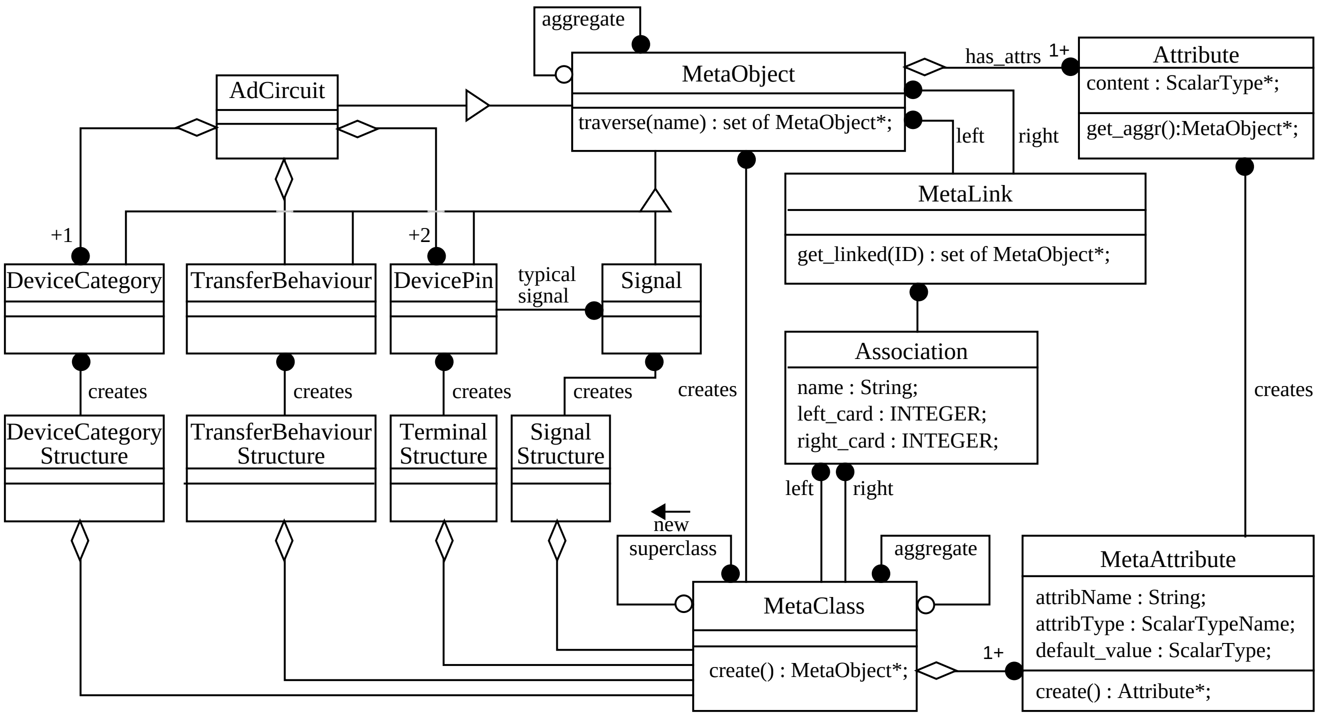

Figure 7. Implementation of the Meta Schema and the CDM (detail)

be modified according to each schema modification.

An important requirement of COPIA is a consistent reorganization of existing data according to modifications of the evolutionary part of the CDM. Furthermore, we require that an evolution step does not entail a modification of application code in order to migrate COPIA to the modified CDM schema.

Like most existing approaches to schema evolution our solution is based on the notion of schema transformations [8,7]. A schema transformation *T* is defined as a tuple *T=(s,i)* where *s* is the *structure transformation* and *i* is the *instance mapping* of *T.* The structure transformation

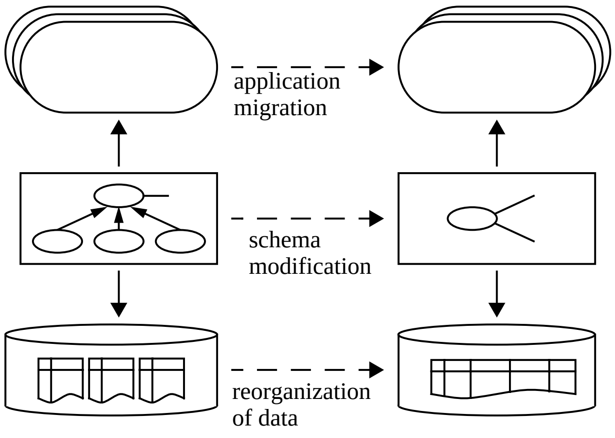

Figure 8. Three Facets of the Evolution Problem

$s{:}S^{*} \longrightarrow S^{*}$ is defined on the set of all schemas S^{*} that can be defined in the data model of discourse. It can be defined by a tuple *s=(p,q)* where *p* is the *precondition* and *q* is the *postcondition* of *s*, respectively. If a structure transformation *s=(p,q)* is applied to a certain schema $S \in S^{*}$ that fulfils *p*, it transforms *S* into another schema $s(S)=S' \in S^{*}$ that fulfils *q*. The instance mapping converts valid database states of an original schema *S* into valid instances of a target schema *S'* by analogy. Formally, the instance mapping $i{:}I \longrightarrow I$ is defined on the union of the *information capacity* of all possible schemas, $I = \bigcup_{S \in S^{*}} \mu(S)$. At this, the information capacity $\mu(S)$ of a given schema *S* is defined to be the set of all valid database states (or instances) of *S*.

If the evolution problem and especially its data reorganization facet is addressed, an important property of a given schema transformation *T(S)=S'* is how it affects the information capacity of the target schema *S'* with respect to *S*. According to [7], a given schema transformation *T=(s,i)* is

- *information-preserving (IP)* if its instance mapping *i* is bijective,
- *information-augmenting (IA)* if *i* is injective but not surjective,
- *information-reducing (IR)* if *i* is surjective but not injective.

Otherwise *T* is called *noncomparable (NC)*.

Although this classification is also used in the area of conceptual database design[8], it is especially important in structure knowledge which is modelled in the CDM. Since the CDM is expected to be subject to frequent modifications, it is important that this interface does not have to be modified according to such evolution steps. Otherwise it would be necessary to adapt the applications using COPIA as well.

COPIA should provide operations to modify the CDM. These operations should automatically change the internal database schema and they should entail a consistent reorganization of already existing circuit data. Furthermore, COPIA should migrate to a modified CDM without the need for changing and recompiling its application code.

An important technical requirement is that COPIA has to be applicable in a heterogeneous environment in order to make the analysis system broadly available. Typically, relational database systems are used in industry today. Thus, COPIA is based on standard relational technology (ODBC) to increase customer acceptance and to avoid additional investments in object databases.

3. A Meta Schema Approach

Another central requirement is that COPIA's application code has to be independent from changes in the evolutionary part of the CDM. Thus, the classes and associations of the CDM could not be directly implemented in the type system of the programming language (C++). Moreover, COPIA has to provide functionality to interactively modify a CDM. In order to meet these requirements we followed a meta schema approach like proposed in [3], i.e. we designed a schema that defines the structure of all possible CDMs.

Figure 6 illustrates the three object levels of this approach: the first level represents the data about concrete circuits. This is exemplified by a detail of an object structure that represents the sample comparator circuit TL331M of Figure 2. The CDM is located on the second level. In this illustration it classifies the TL331M to be a comparator with two analog inputs, a positive analog power supply, and a given transfer function as part of the entire transfer behaviour. Each CDM has to be a valid instance of the meta schema which is represented on the third level. It defines concepts for classes, associations, attributes etc. The reader should note that in our application domain we do not need to represent method definitions in arbitrary CDM classes. All methods are declared in the stable root classes of the four data structures that constitute the CDM (cf. Figure 5). Methods in descendent classes are treated as attributes because typically they define mathematical characteristics like transfer functions of circuits and signals. This is an important assumption that reduces the complexity of the given evolution problem because in general it is an unsolved problem to determine the semantic impact of redefinitions of method bodies within the inheritance hierarchy.We

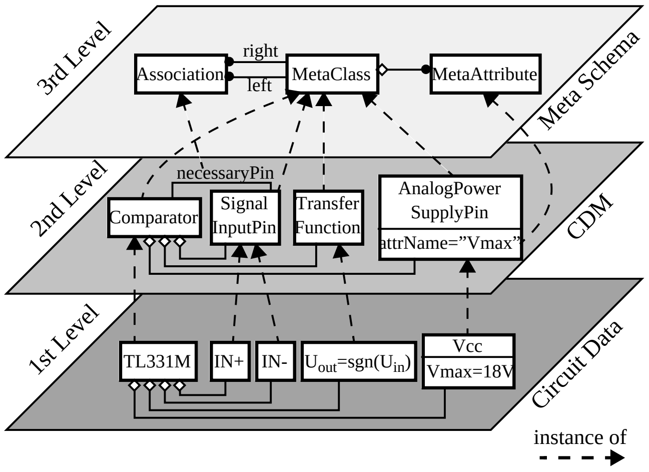

Figure 6. Three Object Levels (detail)

implemented the meta schema and the stable part of the CDM in the type system of C++. The evolutionary part of the CDM (2nd level objects) and the circuit data itself (1st level objects) are represented as C++ runtime objects. Figure 7 gives a more detailed view on the implementation of the meta schema and the stable part of the CDM in COPIA. The roots of the four CDM data structures are represented by the classes *DeviceCategoryStructure*, *TransferBehaviourStructure*, *TerminalStructure*, and *SignalStructure*. Their descendent classes in the CDM (2nd level objects) are represented as instances of the class *MetaClass*. Instances of the classes *Association* and *MetaAttribute* represent associations and attributes which are defined in the CDM, respectively. Circuit characteristics (1st level) are represented by instances of the class *MetaObject*, respectively its subclasses *DeviceCategory*, *TransferBehaviour*, *DevicePin*, and *Signal*. Entire circuits are described by instances of the class *AdCircuit* where each circuit is classified in at least one device category and has exactly one transfer behaviour. Moreover, it aggregates a number of pins where each pin is associated with one or more typical signals.

The described solution provides high flexibility as an evolution step does not entail recompilation of the application code. However, the drawback of this approach is the loss of static type checking for 2nd level objects that has to be performed by the application at runtime instead.

4. Transformation based Schema Evolution

Figure 8 illustrates the three facets of the evolution problem namely *application migration*, *schema modification*, and *reorganization of data* [3,13]. As already mentioned, for most applications these problems are tackled manually in industrial practice, i.e. data conversion programs have to be developed and application code has to

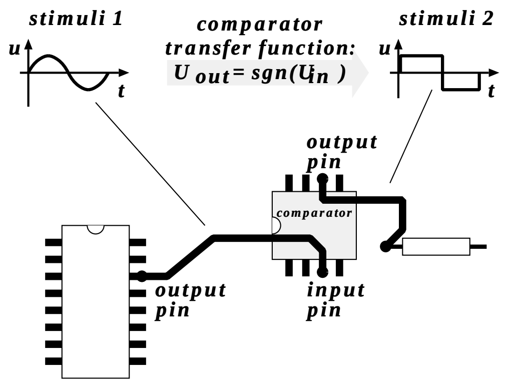

Figure 3. Automatic Stimuli Definition

another component and is possibly calculable, too.

2.2 Data Structures for Modelling A/D Components

The application examples given above show that the component library is a crucial part in a system that performs knowledge based printed circuit board analysis. COPIA maintains four interrelated hierarchical data structures which consider different aspects of analog and mixed analog/digital integrated components as shown in Figure 4. These data structures define an overall *component classification*, the *transfer behaviour classification*, the *terminal classification*, and the *signal classification*. In the following we will refer to these four data structures as the *component description model* (CDM).

The CDM allows the knowledge engineer to specify static (structural) and dynamic (behavioural) characteristics of classes of analog/digital components. Figure 5 shows a detail of COPIA's current CDM[1]. However, the CDM is subject to frequent modifications because in contrast to digital components, analog and mixed analog/digital components are not organized in well-defined circuit families. Even if it would be possible to define a complete CDM for all circuits which are currently available, future inventions with novel functionalities would entail modifications of the CDM. The only stable part of the CDM is represented by the four root classes which are marked by bold boxes in Figure 5.

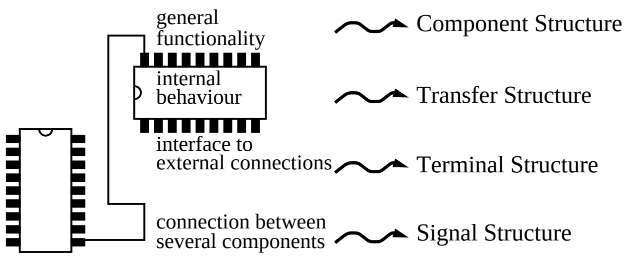

Figure 4. Four Different Aspects to a Circuit

1. For layout reasons Figure 5 shows only class names without their attributes and methods.

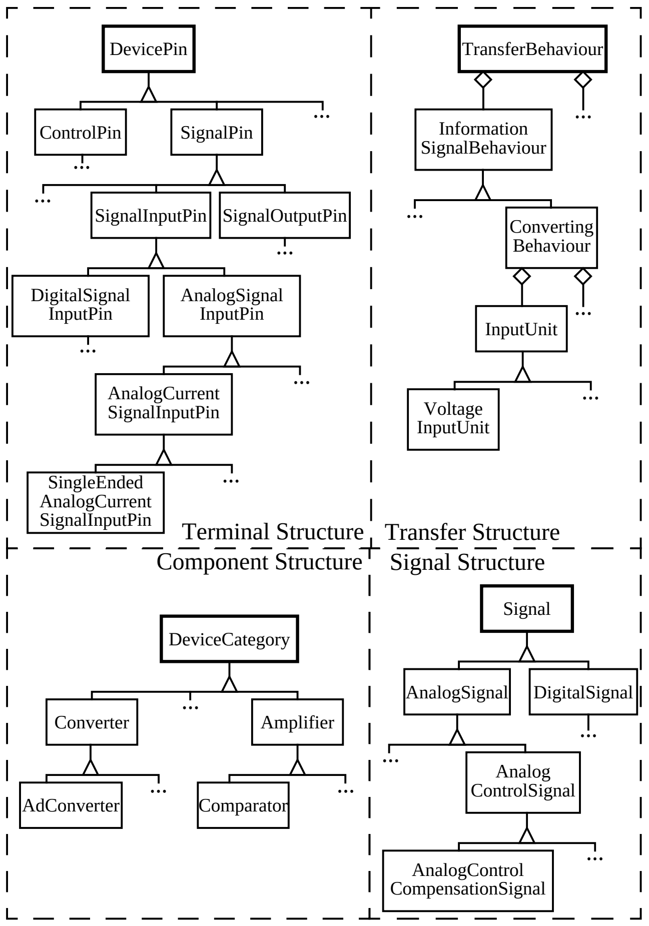

Figure 5. Details of a CDM

The four data structures that constitute the CDM are related with each other to define necessary and forbidden characteristics. This complements the structure knowledge about circuits. For example, a component which has been classified to be a *comparator* (component structure) must have the characteristic *has a one-bit digital signal output* and *must not have an analog signal output*. Such constraints mainly serve three purposes: (1) they ensure the integrity of the application data, (2) they allow a holistic knowledge representation about complex analog circuits, and (3) they enable a context controlled dialogue when new circuits are entered.

COPIA guides the engineer by a context controlled dialogue while entering new circuits, i.e. the system only prompts the user to enter circuit characteristics which are consistent with his previous input.

2.3 Detailed Requirements of COPIA

A major task of COPIA is to maintain the data of mixed analog/digital integrated circuits and to make it available for the analysis program. Therefore, COPIA has to provide an advanced interface which facilitates retrieval of information about concrete circuits as well as information about the conceptual data structure. Based on the meta schema approach, Section 4 describes how conceptual schema transformations are employed to support evolution. Due to our experiences with the case study, Section 5 proposes a reference architecture for similar applications. Finally, Section 6 closes with some concluding remarks and references to related work.

2. Case Study: COPIA - A Complex Information System for Integrated Circuits

2.1 An Analysis System Supporting Printed Circuit Board Design

Developing complex analog printed circuit boards requires a considerable amount of expert knowledge. Correctness and quality of electronic products are difficult to achieve in present of continiously decreasing time to market cycles. Especially the requirements for electromagnetic compatibility are difficult to fulfil in modern high frequency applications. In order to support the designer to meet the specifications in shorter time, we aim to build a computer based analysis system for analog and mixed analog/digital printed circuit boards. This project is carried out at C-LAB[1] in collaboration with HNI[2] and Paderborn University.

Up to now, knowledge about analog designs is not yet structured and formalized for comprehensive usage in analysis or synthesis problems. The designer's work flow is rather dominated by intuitive and unsystematical activities than by a methodical process. The goal of the described project is to structure and formalize this heuristic part of analog knowledge and to make it accessable for computer aided analysis. In contrast to digital designs, the holistic view to an analog design is necessary in order to achieve valuable analysis results, e.g. to perform signal tracing via a vast amount of analog components.

[1,2,12] describe an approach for the solution of the named problems. In these papers we discuss different facets of a knowledge based analysis system. The entire analysis system consists of three main parts shown in Figure 1. The project data (generated by the CAD system) is accessed by the data extractor EXTRAHO. Component data (project independent) is available via the component library COPIA. The analysis tool EXPERIOR combines the project data and component data with (heuristic) analysis rules and (deterministic) algorithms. The output of the analyzer complements the project data and is used for documentation.

The substantial progress of the approach is the

1. Joint venture of Paderborn University and Siemens-Nixdorf Informationssysteme AG
2. Heinz Nixdorf Institute of Paderborn University, research group *System & Circuit Technology*

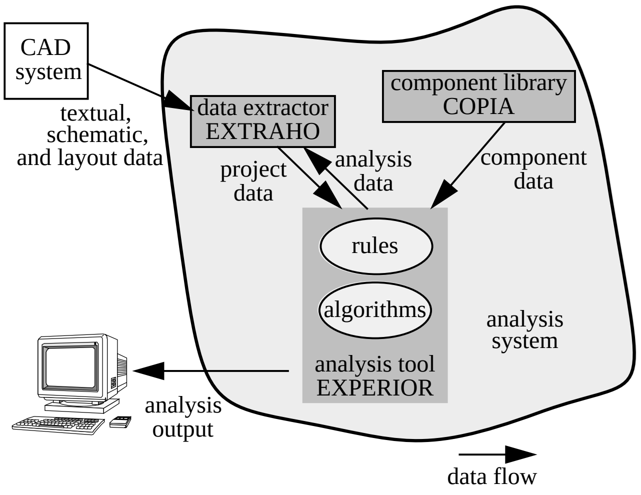

Figure 1. Architecture of the entire analysis system

consequent formalization and modelling of analog knowledge which is an important precondition for knowledge processing. This is done by strictly partitioning knowledge about analog printed circuit boards into component specifications and description of connections. These two complex models were developed separately from other aspects of analog design knowledge. They are implemented in the modules EXTRAHO (layout data extraction) and COPIA (component library).

Often data sheets are an important information source given in textual and graphical form. Especially pinout graphics (Figure 2) contain very important information and typically, they are studied first. Hence, a component library should represent this knowledge. Other requirements like the assignment of pins that belong to a group (e.g. differential input pair) or the selection of a component based on certain characteristics or parameter values are taken into consideration within the component model.

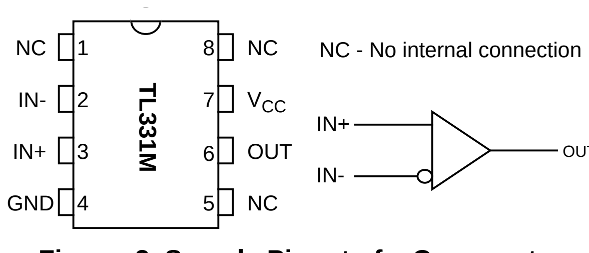

Figure 2. Sample Pinout of a Comparator

Figure 3 shows another application example of the advanced library concept: many analysis aspects need a typical signal for stimulation. For complex circuits, the user desires help for stimuli definition. The example shows that the output signal of components can be calculated automatically if the input signals and the transfer functions are given. Moreover, the signal *stimuli1* may be the output of

A Case Study in Supporting Schema Evolution of Complex Engineering Information Systems

J.-H. Jahnke, U. A. Nickel
Fachbereich 17, Universität Paderborn,
D-33095 Paderborn, Germany;
[jahnke|duke]@uni-paderborn.de

D. Wagenblaßt
C-LAB, Universität Paderborn,
D-33095 Paderborn, Germany;
e-mail: wagen@c-lab.de

Abstract

Information systems have to evolve continually in order to keep up with emerging requirements. Various problems arise with each such evolution step, e.g. the modification of the application's conceptual data structure, the migration of existing data, the adaption of application code, and the modification of technical documentation. Most database systems provide only limited support for schema evolution while problems like data migration and application migration are tackled manually by the programmers. This evolution process is unsatisfactory for a number of novel complex evolutionary information systems (CEIS) in the area of business and engineering applications. This paper describes our experiences with a case study in developing a CEIS in the domain of analysis and design of mixed signal printed circuit boards. We show that a meta schema approach combined with a well-defined set of schema transformations is a practical way to cope with evolution. Based on this case study, we distinguish application specific from reusable architectural components and propose a systematic approach of building CEIS.

Keywords: engineering information system, evolution, meta schema, schema restructuring, graph rewriting

1. Introduction

In time, information systems have to evolve in order to keep up with emerging requirements as the additional customer needs, changing organisational structures, new tax laws, or other hard- and software platforms. Various problems arise with each such evolution step. They may affect the modification of the conceptual data structure, the consistent and efficient migration of existing data, the adaption of application code, the modification of technical documentation and user manuals etc.

Today, most industrial applications are built on top of mature database technology like relational databases. These database systems do not support the evolution of applications sufficiently. Although most relational databases provide simple means to support schema evolution, other problems as data migration and application migration have to be tackled manually by the programmers. In practice this often turns out to be a rather expendable task which is also error prone. This traditional way of implementing evolution steps is unsatisfactory for a number of novel applications in the domain of business and engineering because they require such modifications very frequently. Examples for *complex evolutionary information systems* (CEIS) are enterprise information systems, expert systems for complex engineering tasks, and knowledge based diagnosis systems.

All these applications have three characteristics in common: (1) they manage rather complex conceptual data structures, (2) they have to deal with frequent updates to their conceptual data model, and consequently, (3) they have to provide means to consistently reorganize the data itself. This means in case of an enterprise information system that the production planning system has to be adaptable on the fly when the products have changed. Consequently, it should be possible to migrate existing manufacturing information to the new version of the production plan.

This paper describes our experiences with a case study in supporting evolution in a complex information system in the domain of analysis and design of mixed signal printed circuit boards. We show that a meta schema approach combined with a well-defined set of schema transformations is a practical way to cope with evolution. Based on this case study, we distinguish application specific from reusable architectural components and propose a systematic approach of building CEIS on top of standard relational database technology.

The following section starts with an overview on characteristic problems of computer aided support in the application domain of developing complex printed circuit boards. It describes the role of the component library COPIA as a central part of a novel analysis environment for printed circuit boards. Section 3 defines a meta schema to represent the evolutionary part of the application's