Improving Knowledge Transfer at the Architectural Level: Concepts and Notations

Frank Keller, Peter Tabeling, Rémy Apfelbacher, Bernhard Gröne,
Andreas Knöpfel, Rudolf Kugel and Oliver Schmidt
Hasso Plattner Institute for Software Systems Engineering
P.O. Box 90 04 60, 14440 Potsdam, Germany
{keller,tabeling,apfelbacher,groene,knoepfel,kugel,schmidt}@hpi.uni-potsdam.de

Abstract

This paper presents a vocabulary of concepts and a corresponding graphical notation called Fundamental Modeling Concepts (FMC) to describe the conceptual architecture of large and complex software systems. The main objective of this approach is to improve knowledge exchange, at the architectural level, between the participants of software development projects using a semiformal graphical notation. The focus of FMC is not to represent code structures but rather to describe system concepts and architectural rationale. FMC is not bound to a specific programming paradigm and thus enables communication about any type of software systems. We present both a metamodel and a graphical notation suitable for the intended use. Furthermore, we will summarize our results achieved in applying FMC to software projects of different sizes and scopes.

Keywords: software architecture, conceptual view, architecture description, system modeling

1. Introduction

The development of large software systems requires the collaborative effort of many people. Today, it is commonly the case that multiple or even distributed groups work together on a common system. Hence, many persons need to understand and communicate technical issues on different levels of abstraction. Only if all participants in software development share a similar understanding of the system can software be developed in an efficient manner. Thus, communication between humans represents a crucial aspect for the success of large software development projects.

Communication on the level of detailed software design is well understood. For this level of abstraction the Unified Modeling Language (UML) [1], [2] has become a widely accepted de facto standard in research and industry.

On higher levels of abstraction, communication concerning what is generally known as the *architecture* of a system remains difficult and uncertain [3]. The commonly used object-oriented approach reflects conceptual problems on this level of abstraction [4].

Research and practice have shown that the architecture of a complex system includes different high-level structures spanning from conceptual structures to source code structures. Thus, the representation of system architectures evolved into what is known as *architectural views* [5], [6].

In this paper we will focus on the *conceptual architecture view* which plays a key role in attaining a common overall understanding of a system being planned or already built (for evolutionary purposes). It embodies the structures in the minds of the system designers during the planning and constructing of a system - in contrast to the source structures of the implemented system. It explains how the requirements from the application domain map are transferred onto the technical structure of the system. Thus, it enables one to discuss system aspects such as compliance with nonfunctional requirements, general concepts, the system's components and their interaction, without getting lost in implementation-specific details.

2. Requirements of conceptual architecture descriptions

It has been pointed out that transferring knowledge about conceptual system architecture plays a key role in large-scale software development. Therefore, it is worthwhile to use comprehensive means for this kind of communication, i.e. to provide a uniform set of concepts and corresponding notations whose sole purpose is to deal with knowledge related to the conceptual system architecture. This represents the guiding idea behind the approach presented in this paper.

To use UML for this purpose was rejected, because we share the opinion that "(UML) *is ultimately designed to support the object-oriented view of software design*" and that "*UML currently does not provide a satisfactory*

solution to the modeling of architectures." [7] Hence, "*by using the same notation* (UML) *to describe software architecture, we run the risk of further blurring the distinction between architecture and implementation*" [6].

A technique for describing conceptual software architectures has to meet certain requirements in order to facilitate interpersonal communication at this level:

Abstraction: The ability to describe the conceptual architecture on different levels of abstraction is of major importance. In addition, it has to be possible to describe different conceptual aspects on the same abstraction level separately. This should lead to a system representation reducing the complexity embodied in the technical details of the system implementation.

Simplicity: If a notation and its underlying concepts are highly complex, it will not be easily adopted as means for daily communication between software developers. It should be easy to learn and easy to apply for the majority of software developers. For that reason, a description technique should be reduced to a few elementary concepts and notational elements.

Universality: A sound trade-off between simplicity and conceptual richness has to be made in order to explain the desired architectural concepts of a system. The result should be a description technique which is still simple but also offers enough expressive power to cover a wide range of system types. This implies that neither the notation nor the conceptual basis is bound to a specific implementation paradigm. Furthermore, modeling of the embedding environment of a system has to be possible as well.

Separation of concerns: The description of complex systems has to include different conceptual aspects. It is important for an architectural description technique to support the separation of these aspects by offering comprehensive means for their illustration. This separation of concerns should be fostered by the conceptual basis. A coherent set of structures which are common to all types of systems should build the core of the technique. The choice of notation should support the distinction of the different views and responsibilities during system development.

Aesthetics and secondary notation: Since descriptions of conceptual architectures are made for the human reader, informal criteria for diagram quality must not be ignored. Valuable layout cues, called *secondary notation*, are crucial for comprehensibility [8]. Architectural diagrams will be studied by many persons, so making a clear and appealing layout is worth the effort. Therefore, the notational elements of an architectural description technique should support proper layout including the easy formation of graphical patterns ("secondary notation").

3. The FMC approach

In this paper we present FMC (Fundamental Modeling Concepts), an approach for describing the conceptual architecture of software systems using a semiformal graphical notation. It is based on [9],[10] and has been further refined in [11],[12],[13].

FMC has been developed to meet the requirements listed in section 2. One of the primary problems encountered when dealing with descriptions of large systems is the complexity embodied in many technical details. One strategy of FMC for reducing complexity is to distinguish clearly the following types of structures which are fundamental aspects of any system architecture (separation of concerns):

- Static structures:
  - compositional structures
  - value structures
- Dynamic structures
- Relationships between different FMC models

FMC uses bipartite graphs to depict the structures. The corresponding conceptual and notational elements of FMC will be discussed below. A complete system model is a representation of the whole system on a certain level of abstraction. If we restrict our interest to structures appearing at a certain point in time, two types of structures have to be distinguished - compositional structures and value structures. On the other hand, we can observe system behavior over time.

3.1 Compositional structures

Any system can be seen as a composition of collaborating components called *agents* [14]. Each agent serves a well-defined purpose and communicates via *channels* (and *shared storages)* with other agents. If an agent needs to keep information over time, it has access to at least one *storage* where information can be stored. Channels and storages are (virtual) *locations* where information can be observed.

The agents are drawn as rectangular nodes, whereas locations are symbolized as rounded nodes[1]. In particular, channels are depicted as small circles and storages are illustrated as larger circles or rounded nodes (see Figure 1). The possibility to read information from or write information to a location is indicated by arrows. Agents and locations are identified by descriptive textual labels. As the example shows, it is useful to stretch agent or location nodes in order to achieve a cleaner layout facilitating "secondary notation".

An agent must have *write-access* to at least one shared storage or channel in order to be able to partici-

1. This notation originates from [15]

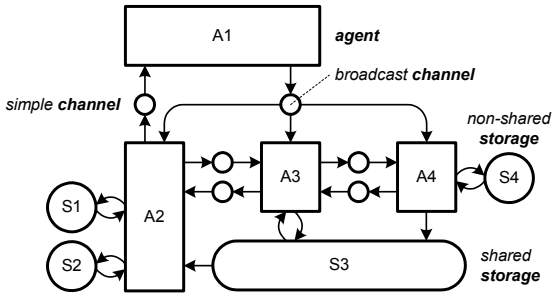

Figure 1: A FMC compositional structure diagram

pate in the system. Each location must be *accessible* (by *reading* and *writing*) to at least one agent. Arbitrary complex structures can be described because there are no further restrictions regarding the number of connections between locations and agents. For example, it is possible to describe *simple channels* (connecting only two agents) as well as *broadcast channels* (connecting more than two agents -see Figure 1). Storages can be used by one or more agents, allowing the description of *shared* and *non-shared storages*.

In general, agents and locations are not related to the system's physical structure. The compositional structure facilitates the understanding of a system, because one can *imagine* it as a physical structure (e.g. as a team of cooperating persons). Nevertheless, on lower levels of abstraction a direct mapping to physical parts of the system might be possible.

3.2 Value structures

Each location of the compositional structure holds a unit of information, called a value. A value can be a simple, *unstructured value* such as an integer as well as a *structured value* such as a tree or the whole content of a database.

FMC offers a dedicated diagram type for the description of value structure types. It is based on entity/relationship diagrams [16] with several modifications and additions. Again, the primary symbols are simple - circles or rounded nodes for entity sets and rectangular nodes for relations. Entity nodes and relation nodes are connected by undirected edges (see Figure 2). This choice of symbols was made because rounded nodes resemble the elliptic nodes often used to symbolize sets (Venn diagrams), whereas a rectangular node can be interpreted as a simplified relational matrix.
Further diagram elements frequently used with FMC E/R-diagrams (see also Figure 2):

- Relation names, entity names and role names can be placed within or near the corresponding graphical element. If an entity has attributes these can be listed below the entity name using smaller typefaces (see upper left of Figure 2).

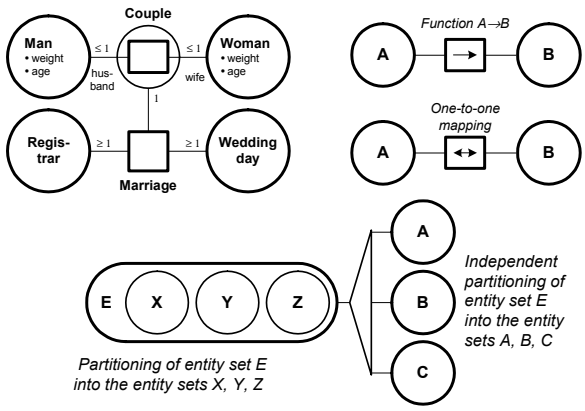

Figure 2: Notational elements of FMC value structure diagrams

- Cardinalities can be specified by placing textual constraints near the respective edges connecting entity nodes to the relation node. A constraint text placed near an entity node E specifies how often each element of E participates in the relation.
- A relation participating in another relation (thus forming an entity set itself) is surrounded by a rounded node (see "Couple" in Figure 2).
- Functions and one-to-one-mappings are identified by simple arrows and bidirectional arrows, respectively (see Figure 2).
- An entity set can be partitioned in disjunctive subsets (see bottom of Figure 2). Usually this is shown by placing the subset nodes within the entity set being partitioned. Whenever there is another independent partitioning of the same entity set, this can be depicted using a triangular partitioning symbol connecting these partitions with the whole.

3.3 Dynamic structures

If we look at a system over a period of time, its dynamic behavior can be observed. A fundamental concept is the *event*, an instantaneous value change occurring at a certain location and at a certain point in time. Whereas events are elementary with respect to *observability*, *operations* are elementary with respect to *activity*, i.e. an operation is the "smallest" activity an agent can perform. When performing an operation an agent reads values from several locations and writes a derived value (the operation's result) at a certain location. Operations can be triggered by events and in turn produce events. This leads to causal dependencies of events.

This notion of an operation covers a variety of operation types. For example, a state change is an operation where an agent reads a value from a certain storage and

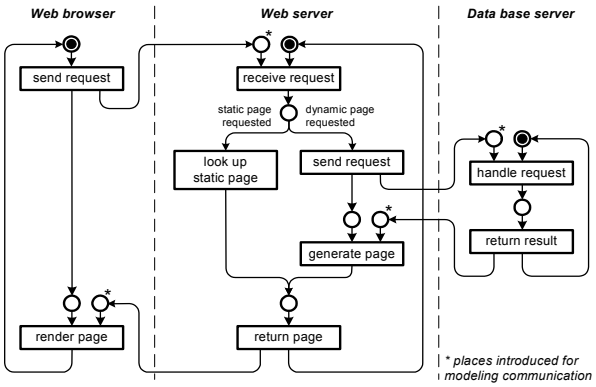

Figure 3: A FMC dynamic structure diagram

writes a new value at the same storage. An agent performs an output operation when the operation's result is written to a channel. If an agent reads a value from one location and writes a copy of it to another location, a pure transport operation is given, etc.

FMC diagrams for dynamic structures are based on Petri nets [17], namely place/transition nets with several enhancements. Transitions symbolize event types, operation types or complex activities, whereas places (mostly) symbolize control states. The text label of each transition node briefly describes the semantics of the corresponding operation or activity. Each transition belongs to an agent of an associated compositional structure. If a diagram covers the behavior of different agents, additional places and edges are introduced to describe causal dependencies resulting from communication, see Figure 3.

Each transition belongs to an agent of an associated compositional structure. To show the responsibilities of agents, it is possible to partition the set of transitions and place them in distinct areas ("swim-lanes"). These areas are separated by dashed lines, symbolizing the domains of the corresponding agents (see Figure 3).

Control states are symbolized by places, whereas data states are handled differently to reduce complexity: operation/activity descriptions identify data state changes occurring at the corresponding storages of the compositional structure. In addition, branch conditions are expressed as data state predicates placed at edges leading to conflicting transitions[1] (see Figure 3, below "receive request").

Because Petri nets can easily be used to describe *concurrency* they are a first choice for the description of complex system behavior. FMC offers an improvement over conventional place/transition nets, one which describes *recursion* as well. This extension has been

1. An extended firing rule has to be applied where transitions are only ready for firing when both the correct control state and the correct data state is given.

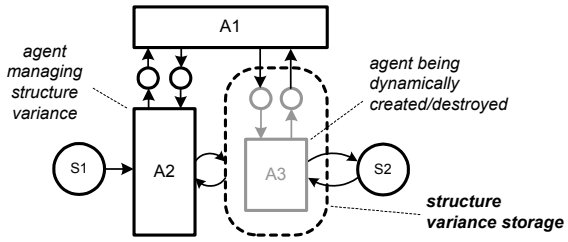

Figure 4: A FMC dynamic compositional structure

suggested in [18] and will not be discussed here further. Concurrency can be described for two cases: either multiple sequential agents working independently or a single agent showing concurrent behavior.

Dynamic Compositional Structures. Complex systems frequently show a compositional structure which changes over time. These changes are usually caused by activities of certain system agents. For example, a dispatcher agent in a server system creates and removes agents which are processing incoming requests. FMC facilitates the modeling of such systems by allowing a part of the compositional structure to appear as a (structured) value of a storage. By changing the "value" of such a storage, an agent can alter the system's compositional structure. (Such storages are symbolized using a dashed border - see also Figure 4). Thus, dynamic compositional structures are the outcome of special types of operations creating and destroying agents.

3.4 The FMC conceptual metamodel

The FMC E/R diagram in Figure 5 presents an overview of the terms and their relationships presented in the previous sections. All elements of the presented FMC structures and their interdependencies are covered by this metamodel. The concept of ports has been omitted here for simplicity. It is described in detail in [19].

3.5 Relationships between FMC models

At one level of abstraction the conceptual architecture of a system can be modeled using the three different FMC structure diagram types. The compositional structure diagram connects all FMC diagram types of one abstraction level to each other.

With complex systems it is usually essential to provide system models on different levels of abstraction. In this case multiple models are given which are hierarchically ordered by the *refinement relationship*: model A is refined by model B if B is derived from A by applying a refinement decision to A. This means iteratively replacing certain model elements with elements of a lower level of abstraction. (This idea does not imply "top down" development.) Such refinement decisions can affect all three FMC model structures.

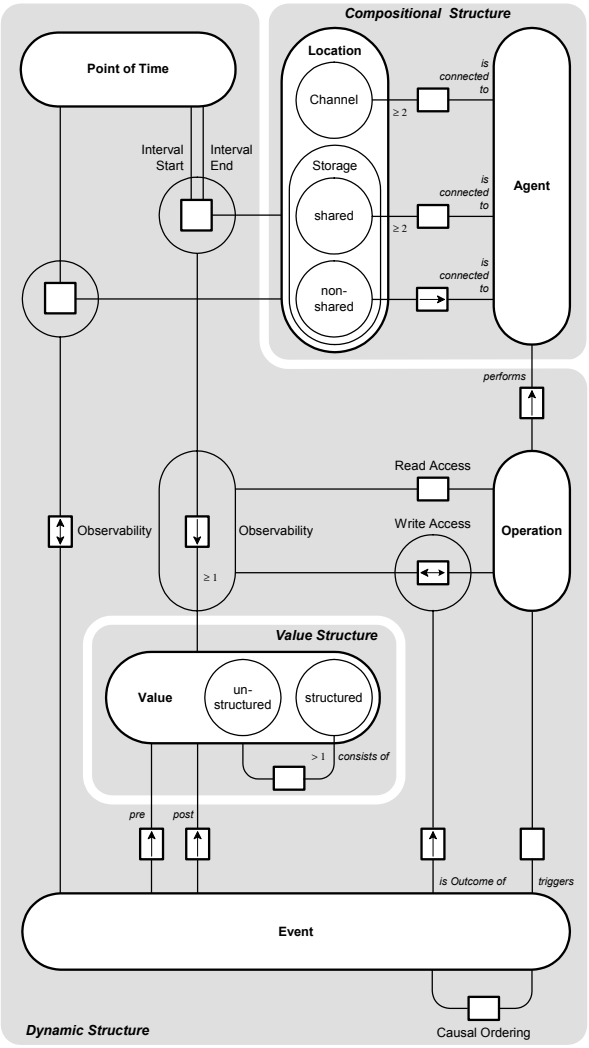

Figure 5: The FMC conceptual metamodel

The simplest type of refinement decision is given if a data type of a higher-level model is refined by data types of a lower-level model. For example, a real number (an unstructured value) can be refined by a pair (mantissa, exponent) of integers (a structured value).

Each operation of a dynamic structure can be refined by a complex activity at a lower level. FMC allows this activity to be a sequence of operations, a set of concurrent operations, or even a mixture of both. Thus, a transition representing a single operation in one diagram can be described in another (refined) diagram as a complex dynamic structure. When refining operations, the corresponding events, read or write accesses, are mapped to their respective lower level events.

In the case of complex systems, many interesting refinement decisions affect the compositional structures of a system. For example, data replication means that one storage is being refined by a set of storages containing (usually) identical values. In general, a location can be refined by one or more locations, whereas an agent can be refined by an agent or a compositional structure.

A simple case of agent refinement is the functional decomposition of agents, where each agent is replaced by a corresponding compositional structure. FMC is not limited to this simple case but allows the description of more sophisticated designs. One example is the refinement of a set of agents (of identical type) by a single multiplexed agent. Interesting types of refinement relationships are given when the refined structure of an agent changes over time, as for example, in the case of load balancing.

Refining elements of the compositional structure often implies the need for refining corresponding dynamic structure elements as well. For example, if an agent is replaced by several cooperating agents, it might be necessary to split operations of the original agent into several operations performed by multiple refined agents.

4. Application of FMC

The FMC approach has been applied to facilitate the analysis and synthesis of software systems of different sizes and different application domains.

FMC was useful in providing a quick overview of the planned conceptual architecture and the core concepts during the design and development of smaller object-oriented system (25-50K LOC). Once the FMC models were in place implementation was straight forward [20]. The FMC models were very helpful during later evolution of a system because the architecture was easy to understand, even though the students who developed the first system release had already left the project by that time [21].

FMC has been applied successfully in the evolutionary development of a large embedded system in the telecommunications sector [22]. The entire system was developed by more than 300 engineers (hardware and software) over the last 10 years (>2.5M LOC in the current release). The FMC case study took place in a subproject of about 20 developers at two different locations. They had been developing new functionality for a system of about 400K LOC for more than one year. Capturing the conceptual architecture in an ad hoc fashion was an explicit part of the group's development process for several years. Thus, it was a straightforward process to apply FMC in this case. The architecture documentation consists of FMC diagrams supplemented with sequence diagrams as commonly used in the telecommunications sector. Furthermore, the required detailed design documentation was done using UML class and sequence diagrams. FMC was introduced to the project through the system architects. After a short time the individual

developers responsible for detailed design and implementation became familiar with the notation. The ability to show different levels of abstraction using a notation that is not related to code structures has been pointed out as one major advantage of FMC. Moreover, the clear distinction between conceptual architecture and object-oriented design through the usage of different modeling notations has been mentioned as one strength by the developers. This distinction gave the developers the freedom to carry out their creative design activities within the given restrictions of the conceptual architecture without being "demoted" to programmers who are responsible for the unalluring tasks concerning a preconceived design. The success of the FMC approach in this case is reflected in the fact that FMC is currently being applied to further development projects in the same group.

Furthermore, FMC has been used to analyze and document the core of a very large-scale business application - the SAP System R/3. This project took place over several years in the industry environment. The architecture of the System R/3 Basis (about 5M LOC) has been analyzed to get an overview of the system and to illuminate many single concepts. As a result of the analysis a set of architectural description manuals [23] have been written, serving as conceptual reference for SAP developers and delivering the foundation for internal and external training in System R/3 core technology. The main challenge of this project was to extract the core architectural elements from such a large system and to present them on an appropriate level of abstraction.

Finally, FMC has been applied to recover the architecture of a medium scale open-source application, the Apache HTTP web server. This architecture analysis and FMC modeling has been carried out in the context of a student course. The results were later presented to academia and industry [24],[25]. These presentations have shown that it is possible to explain architecture and technical concepts in an efficient manner. After a brief introduction to FMC an audience is able to follow the presentations and grasp an understanding of the system's conceptual architecture.

5. Discussion of the FMC approach

The application of FMC has shown that FMC is suited to capture the conceptual architecture of large and complex software systems. FMC has been used to facilitate communication between the project members on a conceptual level.

Based on our experiences we suggest that an explicit architecture phase, dealing with conceptual issues, should be introduced to software development processes. This helps to define robust and stable architectures before applying detailed design. During this stage, FMC is able to provide high level abstractions to describe software systems in order to get a common understanding between project members. The FMC compositional structure diagrams can be seen as mental maps which assist in getting a quick overview of a system's structure. From this point the reader may explore associated behavioral aspects, value structures or refinements of agents in further detailed diagrams. This leads to the ability to vary the level of detail according to the desired needs without being lost in a mass of description detail.

We consider a clear distinction between the notations used for describing conceptual architecture and low-level design as crucial for the development of a system. Our experiences show that this approach has worked well when using FMC to represent the conceptual architecture and UML for the object-oriented design. In some cases it has been difficult to identify which level of detail should be covered by the architecture description and when to move towards design. For the case study in telecommunications, the description at the architectural level stopped when the work package of a single developer was well defined. UML and IDL descriptions have been used to define common interfaces for guiding the transition towards low-level design.

The use of bipartite graphs is fundamental for the FMC notation. The simplicity of the notational elements has the advantage of being easy to draw and to distinguish on a white-board or on paper. This is important since no computer-based tool such as a diagram editor is usually used between software developers during architecture and design sessions.

FMC provides the freedom to stretch and bend the defined notational elements in order to improve layout quality and to enhance their expressiveness supporting secondary notation.

We have recognized that some readers interpret any bipartite graph as a Petri net and are confused by other applications of this kind of graphical pattern. Nevertheless, the simplicity of the FMC notation means that only little effort is required in learning to read and draw the diagrams. This is possible because the terminology of the conceptual basis is restricted to the few coherent concepts shown in Figure 5. Unlike other modeling techniques, the FMC approach does not offer multiple alternatives to express the same aspect. For each of the fundamental structure types (compositional structure, value structure and dynamic structure), there is one comprehensive type of diagram providing strict separation.

FMC is not bound to an implementation paradigm which enables conceptual modeling for any kind of sys-

tem. When moving toward implementation-specific design, an appropriate notation has to be selected (e.g. UML).

One problem in FMC models is that it is not possible to automatically keep the source code and models in a synchronous state. This has to be ensured by an appropriate software development process. However, the high level abstractions of a system - once in place - do not usually change drastically over time.

6. Conclusion and future work

FMC provides a notation and conceptual basis for describing the conceptual architecture of software systems to enhance human comprehension. Furthermore, FMC has been successfully applied to facilitate architecture recovery and development of software systems of different sizes and scopes. FMC is primarily focused on human comprehension of complex conceptual architectures. Thus, we advocate using FMC within an explicit conceptual architecture phase before applying low-level design (e.g using UML).

Besides the results already mentioned, our future work will address the following issues:

Architectural Patterns: Typical large architectural patterns can be observed in many systems. Using FMC diagrams to visualize these patterns would help to model systems in a more efficient way. This would also enhance system synthesis and diagram comprehension due to the advantages of secondary notation.

FMC Methodology: In order to use FMC efficiently in the industry environment it is necessary to provide a development methodology (supported by tools) with guidance on how and when to deploy FMC in large scale development. Here it is important to clarify the coupling between FMC, at the architectural level, and UML, for fine design.

7. References

[1] G. Booch, J. Rumbaugh, I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1999.

[2] OMG. *OMG Unified Modeling Language Specification Version 1.4.* Object Management Group Document formal/01-09-67, 2001.

[3] J. Bargary, K. Reed, "Why We Need A Different View of Software Architecture". *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01).* 2001.

[4] P. K. Laine, "The Role of SW Architecture in Solving Fundamental Problems in Object Oriented Development of Large Embedded SW Systems". *Proceedings of the Working IEEE/ IFIP Conference on Software Architecture (WICSA'01).* 2001.

[5] P. Kruchten, "Architectural Blueprints - The "4+1" View Model of Software Architecture". *IEEE Software.* vol. 12, no. 6, November 1995, pp. 42-50.

[6] C. Hofmeister, R. L. Nord, D. Soni, "Describing Software Architecture with UML". *Proceedings of the First Working IFIP Conference on Software Architecture.* 1999.

[7] T. Weigert, D. Garlan, B. Selic et al, "Modeling Architectures with UML". *UML 2000, LNCS 1939.* Springer, 2000.

[8] M. Petre, "Why Looking Isn't Always Seeing: Readership Skills and Graphical Programming". *Communications of the ACM.* vol. 38, no. 6, June 1995, pp. 33-44.

[9] S. Wendt, "Einführung in die Begriffswelt allgemeiner Netzsysteme", *Regelungstechnik,* vol. 30, no. 1, 1982.

[10] S. Wendt, "Der Kommunikationsansatz in der Software-Technik". *Data Report,* vol. 17, no. 4, 1982.

[11] W. Zuck. *Ein Beitrag zur konsistenten Mitdokumentation von Systementwürfen auf der Basis von Strukturplänen.* Dissertation, Universität Kaiserslautern, 1990.

[12] A. Bungert. *Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle.* Shaker, Aachen, 1998.

[13] P. Tabeling, *Der Modellhierarchieansatz zur Beschreibung nebenläufiger, verteilter und transaktionsverarbeitender Systeme.* Shaker, Aachen, 2000.

[14] S. Wendt, *Nichtphysikalische Grundlagen der Informationstechnik - Interpretierte Formalismen.* 2nd Ed. Springer, Heidelberg, 1991.

[15] Deutsches Institut für Normung e.V.: *Betrieb von Rechensystemen - Begriffe, Auftragsbeziehungen.* DIN 66200, Beuth, Berlin, 1968.

[16] P. Chen. "The Entity-Relationship Model - Towards a Unified View of Data". A*CM Transaction on Database Systems*, vol. 1, no. 1, 1976, pp. 9-36.

[17] W. Reisig. *Petrinetze.* 2nd Ed. Springer, Heidelberg, 1986.

[18] S. Wendt. "Modified Petri Nets as Flowcharts for Recursive Programs". *Software - Practice and Experience,* vol.10, 1980, pp. 935-942.

[19] P. Tabeling, "Ein Metamodell zur architekturorientierten Beschreibung komplexer Systeme". *Proceedings of Modellierung 2002*, GI-Lecture Notes in Informatics, Tutzing, 2002.

[20] M. Kappel, *Entwurf und Implementierung eines Simulators für Register-Transfer-Netze.* Diplomarbeit, Universität Kaiserslautern, 1995.

[21] R. Kugel, *Entwurf und Implementierung eines Codegenerators zur Integration neuer Bausteintypen in einen RTN-Simulator.* Studienarbeit, Universität Kaiserslautern, 1997.

[22] M. Kappel, P. Monz. *TND Architecture Documentation of Segment ITMF.* Project Documentation, Alcatel SEL AG, Stuttgart, 2001.

[23] SAP AG. *Reports of the SAP Basis Modeling Group.* SAP-AG, Walldorf, 1990-2001.

[24] B. Gröne, A. Knöpfel, R. Kugel. *The Apache modeling project.* http://www.hpi.uni-potsdam.de/apache, 2002.

[25] B. Gröne, A. Knöpfel, R. Kugel. "Design recovery of Apache 1.3 - A case study". *(to appear) The 2002 International Conference on Software Engineering Research and Practice*, 2002.