FMC: An Approach Towards Architecture-Centric System Development

Frank Keller, Siegfried Wendt

Hasso Plattner Institute for Software Systems Engineering
P.O. Box 90 04 60, 14440 Potsdam, Germany
{keller,wendt}@hpi.uni-potsdam.de

Abstract

The architectural level plays a major role in the engineering of computer based systems. Having proper means for representing the architecture of a system is a crucial element of large system development efforts as it reduces the amount of uncertainty among the involved stakeholders.

This paper presents a systematic approach called FMC (Fundamental Modeling Concepts) to describe the conceptual architecture of software-intensive systems. FMC is a decision-making and planning tool, facilitating the communication between the architect and further stakeholders. We will give a brief overview of FMC and how it can be applied throughout the development life cycle. Finally, we will give some examples of where FMC has been successfully applied in the industry.

1. Introduction

The engineering of large computer based systems (CBS) requires the collaborative effort of many people from a variety of engineering disciplines. Only if all participants share a similar understanding of the overall concepts, can the system be developed in an efficient and successful manner. This is the driving motivation of this paper. The results are based on the work of Wendt [1],[2],[3] and his research group [4],[5],[6],[7],[8], who have been tackling this problem for more than 25 years in applied research projects with Siemens, SAP AG and other industry partners.

There is a general consensus on the importance of the architectural level of systems development [9]. As pointed out by [10], it is an architectural challenge to reconcile the integration needs of software and hardware to produce an integrated system. Finally, it is the system and not the software inside which customers wish to acquire.

For this reason, research in the field of architecture is an important and rapidly growing discipline as it offers promise for more effective systems development [10],[11]. Current activities cover a wide variety of studies beginning with the role of architecture in industry [12], through architectural analyses [13] up to architecture description languages, aiming at a high-level automated assembly of systems [14]. Nevertheless, research regarding the conceptual system architecture remains very limited (e.g. [15],[16]). In this paper we will focus mainly on the conceptual aspects of system architecting in order to support the system architect throughout the project.

The representation of a system's architecture plays an important role throughout the whole development life cycle [11]. In the early stages, architecture descriptions serve as a major decision-making tool, providing a necessary level of abstraction which aides in dealing with the complexities associated with large CBS. Thus, they allow the studying of alternative solution strategies and the reasoning about their feasibility at a point in the development process when little is known about the final realization of a system. After the planning stage, architecture representations guide the implementation, documentation, deployment and evolution of a system. For this reason, we will present an approach of how architectural representations can be used throughout the entire life cycle.

Architecture descriptions are means of facilitating communication among the various stakeholders in systems development. Especially in the case of CBS, architecture descriptions are important in supporting a holistic overall system planning, aiming at an integrated software-hardware codesign. Thus, one of the goals of our approach is to reduce the gap between the software and the hardware development.

Despite the need for effective exchange of knowledge at the architectural level, this kind of communication remains a major problem in most development projects. An ongoing field-study of multiple industry projects indicates [17] that the communication and documentation of architectural concepts is handled very poorly in the daily life of most architects. This begins with an uncertain understanding of the different architectural categories and ends with an unawareness of appropriate architectural representation techniques.

For this reason, we will discuss these deficiencies in this paper and propose a solution approach: Firstly, we will clarify the role of the architect and his communication needs. Secondly, we will provide a semantic metamodel identifying different architectural categories. Finally, we will present a set of concepts, called *FMC* (Fundamental Modeling Concepts), to describe the conceptual architecture structures of a system along with its application throughout the development life cycle. We will conclude with an outline of some FMC case studies and the lessons learned.

2. The architect's role

In the context of a large CBS development project, the architect[1] plays a central role by having the technical lead of the project. The architect is responsible for defining the overall concepts and thus the architectural structures of the system (Figure 1).

For this reason, the architect has to function as a mediator between the customer side and the developers (engineers). The architect acts to translate between the problem domain concepts of the client and the solution domain concepts of the developers [10]. The architect conceives the architecture and describes his vision about the various system structures with architecture models from different perspectives. The process of architecting is based on the collected requirements, the experience of the architect and several domain specific heuristics[2] the architects applies.

The foremost role of the created architectural models is to communicate by providing an evocative picture of the system in development. At the architectural level, the architect discusses the system concepts with the customer[3] at an early stage thereby ensuring that the system to be built reflects the customer's value judgements of his objectives and constraints.

Therefore, the architect has to explain how the system will operate from the customer's point of view, providing the client with confidence in the progress of design and construction. In the same manner, the architect discusses the current state of the planning with his developers, explaining the overall architecture and evaluating its technical feasibility from the developers' perspective. In this way, architectural models become the documentation of decisions (system rationale) and help to provide the overall design integrity. Models that connect the customer and the developers are particularly helpful in bridging the gap between the

1. The role of the architect represents either a single person or a small team. Such a team is usually small sized to ensure architectural integrity.
2. [10] provides an extensive list of heuristics being applied during architecting.
3. Here, the customer role includes users, operators, etc.

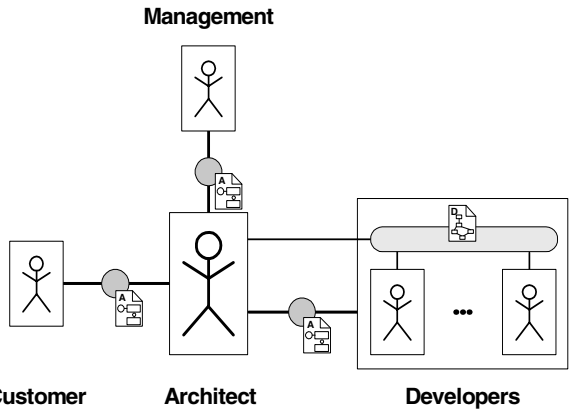

Figure 1: The architect acts as communicator (compositional structure - see section 4)

developers' technical capabilities and the client's objectives.

The task of project reporting on development progress, development costs, resource needs and upcoming problems to the upper management can be supported by architectural models showing the overall system structures. Those models have to be decorated to show the needed facts properly.

When moving from architecture to design, the architect has to define the work breakdown structure based on the architecture defined so far. Then, the developers go into detailed design of the system parts they are assigned to. The design decisions in this stage are constrained by the architecture defined earlier. The design discussion takes place among the developers, involving the architect whenever there is an upcoming mismatch with the imposed architecture. This kind of communication involves models on a lower level of abstraction, providing greater details than the high-level architectural models provided by the architect.

Communication at the level of detailed software design works sufficiently well while developing CBS. For this level of abstraction, the Unified Modeling Language (UML) [18] has become a widely accepted de facto standard in research and industry.

This statement holds for the hardware domain as well where hardware design is a well understood process using methods such as VHDL etc. [19].

On higher levels of abstraction, communication concerning the overall architecture of a CBS remains difficult and uncertain [20]. The commonly used object-oriented approach for the software part reflects conceptual problems on this level of abstraction [21].

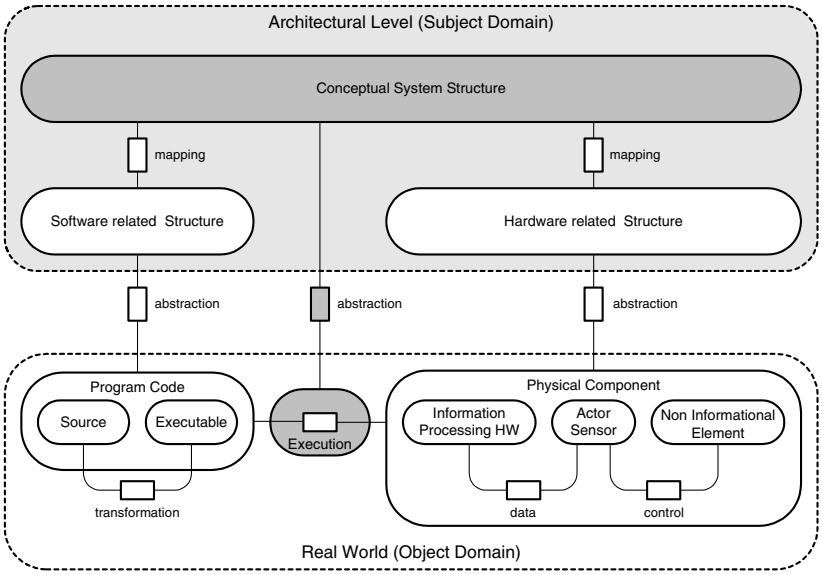

Figure 2: Semantic categories in the context of the term system architecture (E/R diagram - see section 4)

3. Architectural Categories

The term architecture is a very vague one in the context of computer based systems. Even though there is a huge variety of definitions [22], there exist a few major characteristics on which most people agree:

3.1 Architecture definitions

The ECBS Architecture Working Group [11] outlines the term *system architecture* as "a system's fundamental, abstract structure which determines its behavior defined in terms of components, connectors and constraints."

This definition describes the *CCC metaphor* of system architecture, where:

- *Components* are the major elements of the architecture (functional, physical or logical).
- *Connections* are the relationships between those components.
- *Constraints* impose restrictions on the components and their connections.

The IEEE Standard 1471 defines the term *architecture* in a similar way: "(System architecture is) the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution." [9]

These definitions - among others - indicate that architecture is always related to the composition of parts to a larger whole. This implies the idea that architecture deals with structural aspects though the character of those structures is not further specified.

3.2 A conceptual metamodel

A precise definition of the fundamental semantic categories at the architectural level of CBS remains an open issue. The *conceptual metamodel* shown in Figure 2 proposes a set of fundamental semantic categories which have to be differentiated during the discussion of the term *system architecture* for CBS.

Primarily, it is important to distinguish between the entities existing in the *"real world"* (the philosophical *object domain* – the world of things) and the architectural relevant abstractions of those entities. These abstractions form the *architectural level* (the philosophical *subject domain*) comprising different purely abstract structures.

The "real world" (object domain)

The perceptible "real world" entities of a CBS can be partitioned into three distinct categories:

1. ***Program code:*** All program artefacts contributing to the behavior of the system during execution can be subsumed in this category. This means that this category includes the complete *source code* (e.g. C-files) and any of its transformations (e.g. all the *executables* compiled from the sources)

   The program code category represents the programming artefacts, which will finally contribute to the system behavior.

2. ***Physical Components:*** Any physical device of the CBS belongs to this category. This category is partitioned into three distinct subcategories:

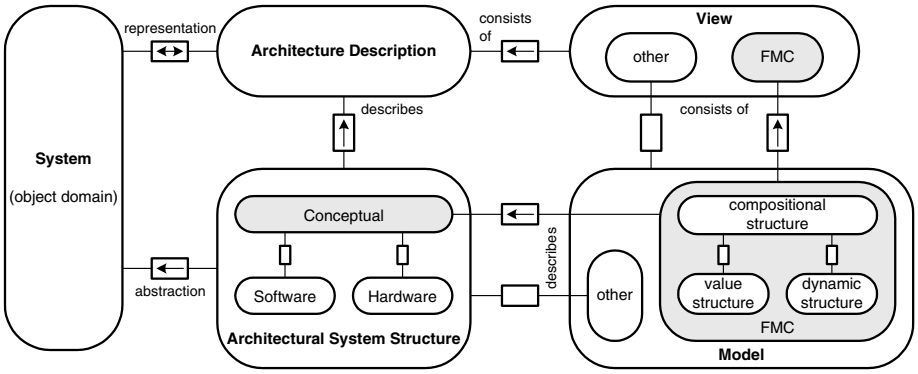

Figure 3: Relationship between FMC and IEEE 1471 views and models

- *Information processing hardware* embodies components which in fact execute the program code (e.g. a CPU).
- *Non informational elements* represent the components of a system that are not related to information processing (e.g. an oil pump).
- A*ctors* and *sensors* of the system represent the interface between the informational components and the environment. On the information processing side, they exchange control information (data) with the information processing hardware. On the non informational side, the *sensors* collect information about physical quantities whereas the *actors* control physical components (e.g. a contactor controlling the energy supply of an oil pump).

3. ***Execution:*** This category represents the desired system itself which only exists when the program code is executed by the information processing hardware during system runtime. The system can be observed by measuring the different physical quantities changing over time during system operation. Thus, the *execution* category delineates the relation between program code and the system's hardware.

The architectural level (subject domain)

Having identified the three fundamental categories in the "real world" domain, it is straight forward to identify three corresponding categories at the architectural level. The architectural level differentiates from the "real world" as it deals with abstractions of the real entities. Reducing the complexity embodied in the "real world" is the intention of this abstraction. The categories related to physical phenomena (*physical component* and *execution*) have an infinite set of properties. The program code category of any large system has a practically infinite set of properties as it is almost impossible for the human mind to be aware of the complete source structures (imagine keeping an overview of more than 1 MLOC for a large system).

1. ***Software related structures***: This category comprises any structure that can be conceived when abstracting from the program code alone. This includes structures such as module/package hierarchies, class hierarchies, call graphs, directory structures and many more which are related to the structural aspects of the program code.

2. ***Hardware related structures:*** These structures are obtained by abstraction from the concrete physical devices being part of the system. These are, for example, electronic hardware architecture structures, plumbing structures, wiring layouts, floor plans and many more.

3. ***Conceptual system structures:*** The missing link between the two other architectural categories is the abstraction of the system itself. This category represents the system when it is performing its operations. It represents the structures needed to understand the system at runtime. For this reason, hardware aspects and program code aspects are conceptually related to each other. This category encompasses component structures and behavioral structures which are tightly coupled. As an example, one can imagine the distribution of operating system processes onto hardware along with their communication protocols.

   Besides the abstraction from the "real world" (the executing system), there must exist a mapping between the other two architectural categories (software and hardware) and the conceptual category in order to trace the conceptual elements to their realization. Conceptual system structures are covered by the *FMC* approach being discussed in the subsequent sections.

3.3 Views and models

It is common practice to describe the architecture of a system from different *viewpoints* where the application of a

viewpoint to a concrete system results in an *architectural view* [9]. Views address one or more aspects of a system (e.g. the security view), usually relating structures from different categories with each other (e.g. [23],[15]).

Views are represented by one or multiple *architectural models* where each model may be part of multiple views. In most cases the models are represented using graphical notations. Nevertheless, there are models expressed only by mathematical expressions (e.g. some performance models).

It is important that the different system models are carefully coupled with each other. Otherwise, the various models can not be aligned to create an unambiguous picture of the whole system.

The FMC approach provides three distinct model types representing the different conceptual system structures at the architectural level. They are well aligned and tightly coupled to describe the conceptual system structures of informational systems. Figure 3 shows how FMC fits into the IEEE 1471 conceptual model of architectural description [9].

4. The FMC concepts

The conceptual system structures play a key role in attaining a common overall understanding of a system. It explains how the requirements from the application domain are transferred onto the technical structure of the system [24]. Particularly in the context of CBS, the conceptual system structures relate the software structures with the hardware structures.

Thus, it enables the stakeholders to discuss various system aspects such as compliance with nonfunctional requirements, general concepts and the interaction of the system's components, without getting lost in implementation-specific details.

It is worthwhile to use comprehensive means for this kind of communication, i.e. to provide a coherent set of concepts whose sole purpose is to express the knowledge related to the conceptual system architecture. This represents the guiding idea behind the FMC approach [25],[26].

FMC is based on [1],[2],[3] and has been further refined in [4],[5],[8]. Even though the notational aspects are not the primary concern of FMC, the concepts of FMC correspond with a semiformal graphical notation. While relying on bipartite graphs, it is optimized for the knowledge exchange between the various architecture stakeholders.

It is not mandatory to use this notation when applying the FMC concept to describe the conceptual structures of a system. Thus, it is possible to use most concepts of FMC with other means of representation (e.g. by using stereotyped UML diagrams). Despite this possibility we do not encourage this step, because by using the same notation (e.g. UML) to describe software structures and conceptual structure, one runs the risk of blurring the distinction between the different semantic categories [15].

4.1 The FMC structure types

One of the primary problems encountered when dealing with descriptions of large systems is their embodied complexity. One strategy of FMC for reducing complexity is to clearly distinguish between the following types of structures:[1]

- *Compositional structures*
- *Dynamic structures* (behavior)
- *Value structures* (data)

A FMC system view is the representation of a system aspect 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.

Compositional structures

Any system can be seen as a composition of collaborating components called *agents* [28]. 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.

With the FMC notation, agents are drawn as rectangular nodes, whereas locations are symbolized as rounded nodes[2]. In particular, channels are depicted as small circles and storages are illustrated as larger circles or rounded nodes (see left of Figure 4). The possibility to read or write information at a location is indicated by arrows.

The distinction between *active* agents and *passive* locations (channels and storages) is a key element of FMC as it is a major mechanism of FMC to reduce complexity: The value structures existing at the various locations are described separately with value structure models. This leads to the second model type describing the system's behavior expressed by the read and write accesses of the agents at the various locations defined in the compositional structure model.

Dynamic structures

The main purpose of FMC dynamic structure models is to describe the *flow of control* through the system.

1. See [25] for a more detailed description of the FMC structure types and the guiding ideas behind FMC.
2. This notation originates from [29]

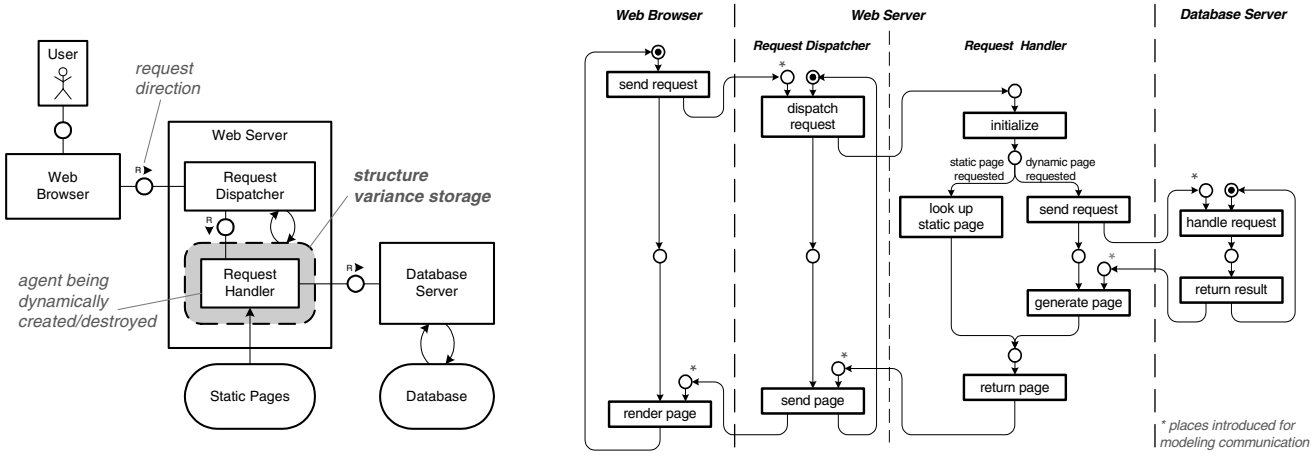

Figure 4: FMC compositional structure model (left) with associated dynamic structure model (right)

Each agent has a well defined functionality represented by the operations the agent can perform. An *operation* is defined by the values an agent reads from connected locations and the resulting value it writes to a certain location. Operations can be triggered by *events* and in turn produce events. This leads to causal dependencies of events.

FMC separates the system's *control states* from the system's *data states*.[1] This leads to a great reduction of the states that have to be described explicitly (only the control states). The differentiation between active and passive components in the compositional structure models enables this classification in an intuitive manner. At a point in time during operation, the system's data state is embodied in the locations of the compositional structure, while the control state of the system is hidden within the agents. Thus, the control states are described explicitly with the associated dynamic structure model.

The FMC notation uses *Petri nets* [33] to represent dynamic structure models (e.g. right of Figure 4). *Transitions* symbolize *operational behavior* (event types, operation types or complex activities) whereas *places* symbolize the *control states*. The text label of each transition node briefly describes the semantics of the corresponding activity.

Each transition belongs to an agent of an associated compositional structure. To show the responsibilities of each agent, 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 4).

1. The usefulness of this separation has already been known by Turing [30]. Throughout the 60s and 70s, it was intensively used by hardware developers [31] and programming language developers [32]. From the late 80s on, this separation of states has been known as extended finfite state machines (EFSM).

Control states are symbolized by places, whereas data states are handled differently to reduce complexity: 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[2] (see Figure 4, Request Handler: below "initialize").

Concurrency can be described for two cases: either multiple agents working sequentially 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 responsible for incoming requests (see Figure 4). 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. With the FMC notation, such storages are symbolized using a dashed border. Thus, dynamic compositional structures are the outcome of special types of operations creating and destroying agents.

Value structures

Each location of the compositional structure has the ability to hold a *value* (data). 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.

The FMC notation offers a dedicated diagram type for the description of value structure types. It is based on entity/relationship diagrams [34]. The primary symbols are

2. 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.

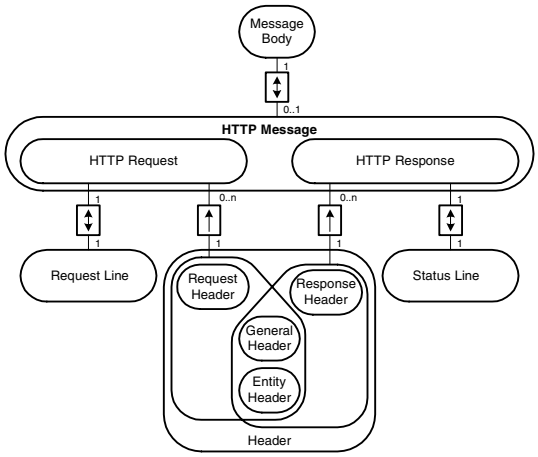

Figure 5: FMC value structure model

rounded nodes for entity sets and rectangular nodes for relations. Entity nodes and relation nodes are connected by undirected edges along with cardinalities (see Figure 5).

FMC conceptual metamodel

All elements of the presented FMC structures and their interdependencies are well defined by the FMC metamodel [25]. Furthermore, there exists a concept of ports [35] aiming at a stronger tool support for FMC.

4.2 Model hierarchies

Each FMC view is a set of FMC models concerning one level of abstraction. Such a view describes the conceptual system structures of a system from a certain viewpoint using the three FMC model types. The compositional structure model connects all FMC models at the same abstraction level. This is due to the fact that value structures describe the data content of locations and dynamic structures describe the way agents interact via channels and storages.

With CBS, it is usually essential to provide system models on different levels of abstraction. In this case, multiple views are given which are hierarchically ordered by a *refinement relationship*. Refinement implies that high-level model elements are substituted by lower-level elements. Such refinement decisions can affect all three FMC model structures and systematically increase the level of detail or show certain implementation decisions. An example of refinement is shown in Figure 4, where the web server agent is refined by the structure shown inside. On a higher level of abstraction one would not show this internal structure thus leading towards a less detailed behavior description.

5. Applying FMC

FMC models can be applied throughout the life cycle of a project, serving as the integral planning part for a CBS.

We suggest that architects use FMC to describe and discuss their conceptual ideas when planning the architectural level of a CBS prior to concurrent engineering of the parts of a system.

This means, e.g. that conceptual system models are prepared in order to decide which components should be implemented using hardware and which ones using software. Furthermore, FMC models can help to determine the technologies best suited to realize the planned concepts. Based on a risk analysis of the suggested models it is possible to determine which parts of a system have to be evaluated with prototypes before accepting a certain solution strategy.

It is important to map the collected requirements to the elements of the architectural models as early as possible. This mapping allows the tracing of requirements to architectural and design elements. It is usually helpful to prepare a “business view” which describes only those components that are relevant to a specific application scenario. Those models usually allow an easy mapping of use cases onto the conceptual architecture. Thus, they can be applied to represent the customer’s view of the system. Furthermore, they are useful to explain the planned solution and its implications on cost and timing issues to the client.

Moreover, it is important to describe the technical core as well (e.g. middleware services). Those models either provide a detailed description of certain concepts or they give a global overview of all components participating in the system. Such models (particularly the composition structure models) can easily facilitate certain project management tasks:

The architect can define the work breakdown structure based on the architecture description rather than on feature lists as it is commonly the case in the industry. After assigning personnel for the realization of the architectural components, the project manager can keep track of the development progress by attributing those elements with a task completion status. Such a “project status map” helps to efficiently communicate the project status and upcoming difficulties or delays to the upper management and the customer. Based on this information, the concerned stakeholders can agree on a solution strategy (e.g. to omit a certain component in the current release or to choose a different technology).

When the architecture documentation is well aligned with the current status of the development, it can be used as a reference for newcomers and during maintenance and evolution. For this purpose, it has been identified that it is useful to review the architecture documentation at the end of a release cycle in order to keep it up to date.

6. FMC case studies

The FMC approach has been applied to facilitate the analysis and synthesis of various systems:

A recent FMC case study took place during the development of a vending machine for refilling prepaid accounts of mobile phones. We were involved in the middle of the architecture definition phase of the project. Until then, the usual "Power-Point box-and-line diagrams" had been used to represent the system's architecture. After a short FMC training (1 day), the team was able to adopt the FMC approach. Some developers had general difficulties in describing the system at a higher level of abstraction as they had to omit many details in their models. Nevertheless, the consequent use of FMC (especially the compositional structure models) soon indicated that the previous models were extremely ambiguous and the intentions were not well understood. This led towards extensive discussions based on the FMC models in order to establish a well-suited conceptual architecture. It was said by the head of development that "FMC helped a lot to facilitate communication and increase team integration" [36].

As another example, FMC has been applied successfully in the evolutionary development of a large CBS in the telecommunications sector [25],[37]. 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 who were developing new functionality. The architects used FMC models and sequence diagrams to describe the conceptual architecture which was well accepted by the developers after a short while. The precise description of the conceptual architecture has been very beneficial for project planning, communication between the two remote sites and finally providing a useful frame for the detailed design. Thus, FMC is currently applied to further development projects by the same group.

FMC has been used as well to recover the conceptual architecture of systems:

One example is a very large-scale business application - the *SAP System R/3* ®. The architecture of the system core (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 [38] have been written which serve as conceptual reference for developers and SAP training courses.

Another example is the architecture recovery of a medium scale open-source application, the Apache HTTP web server [39],[26]. The conceptual architecture models serve now as reference models in FMC training courses and furthermore to study architectural patterns in student courses.

7. Lessons learned

Having conducted several case studies in the industry, the FMC approach has shown that it is well suited to cover the description of the conceptional system structures of even large and complex systems. With the focus on the communication needs of the architect with other stakeholders, FMC addresses a major problem in the daily life of many system architects, especially in the context of CBS.

Scalability

We have applied FMC in the analysis of existing systems and as well as the synthesis of new systems. These studies clearly show the scalability of FMC in describing the architectural concerns of medium to large applications. In the case of very small projects, the usefulness of any architectural description technique is questionable. This is due to the fact that there are usually not many people involved, reducing the need for communication drastically. Thus, one should consider the application of agile methods (e.g. Extreme Programming [40]) in those cases.

Development process

Currently, there are no well suited process models available, neither supporting the use of FMC for development nor for architecture analysis.

Based on our experiences we suggest that an architecture driven development process should consider an explicit architecture phase, dealing with conceptual issues. 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 the 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. Thus, the compositional structure models can be used to guide activities throughout the development process (e.g. to trace requirements to architectural components or to define the work breakdown structure).

Notational aspects

A clear distinction between the notations used for describing conceptual architecture and low-level design is beneficial to 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.

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 sessions.

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 a few coherent concepts.

FMC is not bound to an implementation paradigm which enables conceptual modeling for any kind of system. With a semantic extension of the compositional structure diagrams, it is even possible to describe material flows besides the purely informational intention of FMC. When moving toward implementation-specific design, an appropriate notation has to be selected (e.g. UML).

Model consistency

The problem of keeping architecture and design models in a synchronous state with the "real world" artefacts is often mentioned. Our experiences show that the low level design models of the software and the hardware categories should reflect the current state of the development. Those models serve the purpose of detailed communication among the developers or engineers. If they are not aligned with the sources or the hardware implementation, they do not fulfill their purpose. This is in contrast to conceptual models on higher levels of abstraction. Once those models are in place, they usually do not change drastically over time. Experience shows that the integrity and consistency of architecture models can be sufficiently ensured by an appropriate development process [37]. Nevertheless, a well suited tool support would help to keep the architectural models up to date.

8. Concluding remarks

We have discussed the shortcomings of the state of the praxis regarding the architectural description of CBS. Thus, we have presented an approach to describe the conceptual system structures at the architectural level.

As a first step, we have defined three different architectural categories in the context of CBS (software structures, hardware structure and conceptual structures). Next, we have described the communication needs of a system architect concerning the conceptual system structures.

This led us to the FMC approach which facilitates the architectural description of conceptual system structures. FMC focuses on the support of the architect in his communication with other stakeholders concerning conceptual issues. The conceptual basis of FMC provides several mechanisms to reduce complexity when describing a system's architecture (hierarchical decomposition, separation of component structure from behavior and data, distinguishing active and passive components and isolating the control states from the data states).

Even though it is possible to use the FMC concepts with any appropriate notation (like UML), we encourage the use of the dedicated FMC-notation. The strict use of bipartite graphs and the clear graphical distinction of the conceptual system structures from the other architectural structures has proven to be beneficial in practice.

Finally, FMC has been successfully applied to facilitate the architecture recovery and development of medium-sized to large CBS where it has proven its scalability and usefulness. Nevertheless, there are still some open issues concerning the application of FMC:

- Current development processes do not address the conceptual architecture of CBS properly, thus hampering the successful application of FMC. Therefore, we are currently developing an architecture driven process (ADP) tailored for the use with FMC.
- The current tool support for FMC is still relatively weak (i.e. a drawing tool based on Visio® stencils). Particularly, an integration of FMC with existing tools (e.g. project and risk management) and a repository based storage system would be desirable.
- It may be useful to attribute and analyze FMC models for estimating certain quality attributes of a planned system architecture.

9. References

[1] S. Wendt, "The Programmed Action Module: An Element for System Modelling", *Digital Processes,* vol. 5, 1979, pp.213-222.

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

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

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

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

[6] F. Gales, *Ein Beitrag zur Schaffung programmierter Systeme,* Shaker, Aachen, 1998.

[7] W. Kleis, *Konzepte zur verständlichen Beschreibung objektorientierter Frameworks,* Shaker, Aachen, 1999.

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

[9] IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems,* 2000.

[10] E. Rechtin, M. Maier, *The art of systems architecting,* CRC Press, Boca Raton FL, 1997.

[11] H. Simpson, et al, "Computer Based System Architecture - An IEEE ECBS TC Architecture Focus Group Discussion Paper", *International Conference on the Engineering of Computer Based Systems (ECBS'98),* IEEE Computer Society, 1998.

[12] K. Smolander, et al, "What is Included in Software Architecture? A Case Study in Three Software Organizations", *Proceedings of 9th annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'00)*, IEEE Computer Society, 2002. pp 131-138.

[13] J. Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach,* Addison Wesley, 2000.

[14] N. Medvidovic, R. Taylor, "A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering,* vol. 26-1, 2000.

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

[16] H. de Bruin, H. van Vliet, "Top-Down Composition of Software Architectures", *Proceedings of 9th annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'02),* IEEE Computer Society, 2002.

[17] F. Keller, *On the Role of Architecture Descriptions in Industry Projects,* Hasso Plattner Institute for Software Systems Engineering (unpublished manuscript), Potsdam, Germany, 2003.

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

[19] K. Ten Hagen, H. Meyr, "Generic Design: Its Importance, Implementations and Limitations", VHDL International Users' Forum (VIUF), IEEE Computer Society, 1993, pp 297-309.

[20] 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.

[21] 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.

[22] Software Engineering Institute, *How Do You Define Architecture,* http://www.sei.cmu.edu/architecture/definitions.html, Carnegie Mellon University, 2002

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

[24] A. Ran "Software Isn't Built From Lego Blocks", *Proceedings of the 1999 Symposium on Software Reusability,* Los Angeles, California, United States, 1999, pp. 164-169.

[25] F. Keller, P. Tabeling, et al, "Improving Knowledge Transfer at the Architectural Level: Concepts and Notations", *The 2002 International Conference on Software Engineering Research and Practice,* Las Vegas, CSREA, 2002, pp. 101-107.

[26] B. Gröne, A. Knöpfel, R. Kugel, "Design recovery of Apache 1.3 - A case study", *The 2002 International Conference on Software Engineering Research and Practice*, Las Vegas, CSREA, 2002, pp. 87-93.

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

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

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

[30] A.M. Turing "On Computable Numbers, with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society,* vol. 42, 1936.

[31] S. Wendt "Eine Methode zum Entwurf komplexer Schaltwerke unter Verwendung spezieller Ablaufdiagramme", *Elektronische Rechenanlagen,* vol. 12, no. 6, 1970, pp. 314-323.

[32] R.M. Burstall "Formal Description of Program Structure and Semantics in First Order Logic", *Machine Intelligence,* vol. 5, 1969, pp. 79-98.

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

[34] 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.

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

[36] R. Irsigler et al, *TobiAutomat,* Project Documentation, Intervista AG, 2002.

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

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

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

[40] K. Beck, *Extreme Programming Explained,* Addison Wesley, 2000.