Component-based vs. Model-based Development: A Comparison in the Context of Vehicular Embedded Systems

Martin Törngren[1], DeJiu Chen[1], Ivica Crnkovic[2]
[1]KTH, Stockholm, [2]MdH, Västerås
martin@md.kth.se, chen@md.kth.se, ivica.crnkovic@mdh.se

Abstract

Component based and model based development (CBD vs. MBD), in their various interpretations, are in focus in many efforts in order to better handle the efficient development of increasingly complex embedded systems. We elaborate on what CBD and MBD represent, on their differences and similarities. Although CBD represents a bottom-up approach whereas MBD is more top-down in nature, it turns out that the concepts have much in common and can benefit from cross-fertilization. In particular, MBD requires improved handling of 'model' components, and CBD requires improved component models to assure component composition and reuse. We discuss their mutual opportunities and other relationships.

1. Introduction

The motivation behind this paper arose in the context of the SAVE research project [1], dealing with component based development of software for safety critical vehicular systems, such as aircraft and trucks.

In research communities working on these and related topics, two buzzwords that frequently appear are Component based and Model based development (CBD and MBD), see e.g. [2]. Such buzzwords cause confusion because the terms are quite generic and are consequently used in a variety of contexts for different purposes. We believe, however, that there are needs to correlate the two, one reason being to facilitate communication and reduce unnecessary redundant work among researchers. Misconceptions such as "MBD is only about code generation" can then be avoided. Moreover, we believe that cross-fertilization is possible by reusing concepts and results across the two approaches. In this paper we elaborate on what CBD and MBD represent, on their differences and similarities, and discuss how they can be combined. More underlying material and case studies can be found in [3].

Embedded systems constitute an essential enabling technology for vehicular systems, providing possibilities for high performance and entirely new functionality such as diagnostics, active safety systems and passenger entertainment. As a consequence, embedded systems carry an increasing portion of the total vehicle cost as well as the value-added.

Key characteristics of vehicular embedded systems include their environmental dependence and increasing connectivity. The "embedding" dimension strongly affects the requirements on the software and hardware portion of a product, on the one hand since this portion will inherit requirements related to the product itself (such as constraints on safety and performance) and on the other hand, since the embedded systems have to withstand the harsh environments typical for vehicular systems. The embedding dimension among other things leads to real-time constraints and tight relations to the environment, see e.g. [4], complicating system design and component reuse. In providing more functionality and answering demands relating to performance and flexibilities, embedded systems have over the past 30 years evolved from stand-alone microprocessor based systems to distributed embedded systems, which today also are increasingly connected to the external world. The flip-side of the coin is the increasing complexity, which comes in many forms including a large amount of entities and relationships of various types, variants, and many (and conflicting) requirements.

This paper includes 6 main sections. In the next section, we present the major ideas behind the characterization and comparison. Key characteristics of these two approaches are described in the section 3 and 4, and compared in section 5. Finally, a discussion on their mutual opportunities is provided in Section 6.

2. Comparison approach

To be able to profile and compare CBD and MBD we have developed a framework that codifies important issues. The framework considers three related dimensions of engineering as depicted in Fig. 1 to address the following questions, further discussed in Sections 3 and 4: *Q1. Why is an approach attractive*? *Q2. What are the concepts and techniques*? *Q3. When and where is it applied*? *Q4. How to apply it*?

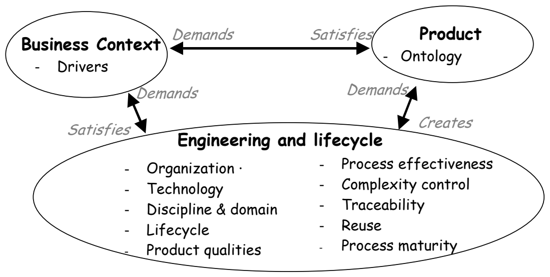

Figure 1. Context and content of engineering in three dimensions.

Relating to the question *Q1*, the dimension of *business context* characterizes the motivations or aims of a particular approach with respect to customer- or enterprise-needs, referred to as its *drivers*.

The dimension of *product* relates to the question *Q3* with respect to the target systems being developed or managed by an approach. One key issue is *ontology*, referring to the explicitly considered means, relationships, and environmental issues of a complex whole constituting a product [5]. Due to its purpose(s) and role(s), an approach can have different focuses and adopt different interpretations of system synthesis.

The dimension of *engineering and lifecycle support* relates to questions *Q2*, *Q3*, and *Q4* with respect to the supported development activities. Such activities are managed into processes, including technical processes for creations, evaluation, and synthesis of products, and management processes for resources, changes, configurations, work planning, and integration of teams [6, 7]. For complex products, multiple processes, organized into levels and domains, are involved, targeting different phases and concerns (e.g., safety and performance). Within each process, iterations are often necessary for rectifying design errors or taking changes into account.

In this dimension, we highlight a set of factors, as listed in Table 1. For the development of complex products, one central issue is support for the ease of cognition and knowledge transfer, i.e., *complexity control*. A closely related issue is support for information *traceability*, over time, from requirements to solutions, within or across engineering tasks. Since engineering seldom starts from scratch, support for *reuse* is also an issue of concern. This requires not only run-time compatibility, but also support for variant management, IP-protection, and design-time integration, etc. To perform development tasks, it is always preferred that explicit methodology and tool support are provided by an engineering approach. This is covered by the *process maturity* factor. The *process effectiveness* is affected by factors like the number of iterations within each process. To this end, a parallel organization is often advocated, such as in concurrent engineering where the expertise associated with a specific stage is made available at every stage [8].

Table 1.Comparison factors in engineering and lifecycle support.

Factors of Engineering and Lifecycle Support	Definitions
Organization	Organizational prerequisites and consequences of adopting the approach.
Technology	Required technologies for performing the approach.
Discipline&Domain	Intended disciplines and engineering domains.
Lifecycle	Stages of the product lifecycle.
Product qualities	Support for evaluation criteria and evaluation related tasks in technical processes.
Complexity-control	Explicit support for the ease of comprehension and communication.
Traceability	Explicit support for managing information dependencies.
Reuse	Explicit support for managing and using existing solutions.
Process effective-ness	Applied principles and concepts for promoting engineering efficiency.
Process maturity	Completeness and formality of provided engineering methodology and tools.

3. CBD

A basic idea in component based software development is building systems from existing components as opposed to building the entire application from the scratch. This idea has two important consequences:

- The components are built to be reused in different systems. It allows reuse of development effort by allowing components to be reused across products and in the longer term it enables building a market for software components. In particular for the development of many variants of products the component-based approach is attractive.
- The component development process is separated from the system development process. This allows significantly shorter time-to market as products are integrated from the already existing components.

The latest trends show that different component technologies are being developed for different domains. Similarly to the object-oriented paradigm that is exploited in different OO languages, a component-based paradigm based on certain common principles is slowly built and used in different component technologies [2, 9].

The main features of CBD originate from business requirements:

- Short time to market – large savings in time can be achieved by constructing applications from already existing parts.

- Distribution of work among dedicated experts for development of components. Since components are developed independently of the products the experts in particular domains can develop them.

From the engineering point of view the advantages of CBD are based on standardisation and reusability. Standardisation plays a crucial role as it enables independent development and seamless integration of components. By reusing the same entities the confidence of their behaviour and properties increases.

Similar to other engineering domains, CBD aims for targeting complexity: By reusing existing solution not only on the component level but also on the system structure level CBD enables better understanding of complexity; the implementation details of components are hidden and only component services that are exposed through component interfaces are visible. In this way the abstraction level is increased which is a key factor in managing complexity.

Further, in the CBD approach the maintenance is focused on replacement similar to replacement of spare parts of components rather than on re-implementation of specific parts.

3.1 Basic characteristics of CBD

The central terms in CBD are the component and the component model. In classic engineering disciplines, a component is a self-contained part or subsystem that can be used as a building block in the design of a larger system. It provides specified services to its environment across well-specified interfaces. Ideally, the development of components is decoupled from the development of the systems that contain them. Components are assumed to be reusable assets in different contexts.

A *component model* specifies the standards and conventions that are needed to enable the independent development of components, and the composition of independently developed components. More concretely, a component model defines the set of rules that the components and application obey in order to make it possible to interact and to be composed. These rules include component specification, component deployment and execution and communication between components. The specifications of component interface and component interactions determine the architectural framework of component-based systems. Indeed different component models assume different architectural styles.

The key consideration of CBD is *reusability* and *adaptability*. The component models must have a mean by which component can be reused and adapted to the requirements.

Reusability is an ability of reusing the same component in different applications. This desired property includes a number of assumptions:

- The architectural framework of the applications must be the same.
- Non-functional run-time properties of the applications are similar.
- The component functionality is general enough to be reusable, but simple enough to be understandable.

Adaptability is an ability to respond to changing requirements, either in different applications or in an application during its lifecycle. In CBD it is realized by the following properties: *Substitutability and Expandability*.

CBD aims to provide support for performing safe substitution of a component. Substitutability is related to contractual interfaces and dependencies. The contractual interfaces distinguish “require” interface from “provide” interface and for them they specify the pre-conditions that must be valid for a correct execution of the component and post-conditions which are valid for output values after the component specification. The contractually specified interface also identifies the states (invariants) that hold after the component execution. The dependences, expressed through the “require” interface indirectly specifies on which other components a particular component is dependent of. However, while the principles of functional substitutability are known, theses principles are much more diffuse for non-functional properties.

For expandability CBD provides support either in an ability of adding new interfaces to existing components, or ability to add new components in the system that provide the new functionality.

Since CBD assumes of existence of components before the development of an application or system, many activities in the development process follow bottom-up principles (within given frames defined by the component model). For this reason *composability* is an important concern in CBD.

Since CBD aims for reusing existing components the important issue is a composition of components – not only for providing the mechanisms for the composition but also for modeling applications reasoning about different properties. The main goal is to provide theories, methods and tools to predict the composition properties from the properties of components.

$$P(c_1 \circ c_2) = P(c_1) \circ P(c_2)$$

Again, there is much more work done and there is more experience in predicting compositions of functional properties; functional composability often it is immanent part of the component models. Composition of non-functional properties is in general more complex; There are properties that are results of composition and architectural solutions, there are properties that are visible on the system level, but do not exist for the components, and there are properties that are nit composable at all.

3.2 MBD related efforts and challenges

The primary drivers of the CBD approach come from the business motivations and from engineering in different domains: Components should be reused and only specifications of the components are necessary for their usage. This requirement states the main challenge: How to specify components? The specification include identification of different aspects – from the functional formal specification, to the specification of different properties such as resource usage, detailed documentation about the component usage, test support, etc.

For component specification, different modeling or specification languages are used – for example UML, or other Architectural Definition Languages, or IDL (interface definition Language). While functional specifications follow certain standards or are an integrated part of the component technology, for non-functional properties there are is a wide range of different notations that originate from different theories, not directly related to CBD. One systematic approach is given in PECT (Prediction-Enabled Component Technology) [10] by which the component specification is implemented via interface specifications classified in *(i) constructive interface,* and *(ii) analytical interface*.

The constructive interface specifies how the components can be connected ("wired") and it is an integrated part of a component model. The analytical interface originates from different property theories (for example theories of timeliness, or reliability) For a particular component model this interface provides the mapping between specification of the components and the notation from particular theories (for example a component may be an executable unit with specification of execution time, which is then used by scheduling theory to design a time-correct system).

A challenge related to a component specification is the verification; How to verify that a component will behave correctly in a new environment? The example of Ariane 5 [11] illustrates the case in which a component has been reused in a new environment, resulting in a major system failure . The question of verification is related to testing and formal verification. In both cases it is a challenge – what specification is needed and which methods can be applied.

From the business perspective the main challenge is that of trustworthiness related both the run-time performance and lifecycle issues. For example, there is a question, how trustworthy the specification is? Or a question related to the lifecycle of a product: Who is responsible for the maintenance, or how the evolution of the component will be related to the evolution of the product, and the evolution to other components?

Another challenge of the business model is the reusability: If one of the goals is to develop a component for reuse, how much additional efforts will be required to achieve this, and how much overhead will be encapsulated in the component and how much additional system resources this will require.

These challenges make the progress of the CBD approach slower, particularly in the domains in which the non-functional properties are the primary concerns.

4. MBD

Modeling and models are strongly associated with engineering [7, 12]. They are used for *communicating* information, for *analyzing* a system, thus providing answers to certain questions about a system, and for *synthesizing* a system, e.g. through transformations. One traditional definition of a model is: "A model is a simplified representation of a system intended to enhance our ability to understand, predict and possibly control the behavior of the system" [13]. For example, consider a body sliding on the ground and affected by an external force *F*. Based on Newtonian laws and certain assumptions we can obtain a behavioral model in the form of a differential equation: $d^2x/dt^2=F-kdx/dt$. The resulting model can be used to predict the body's motion and for synthesizing a feedback control system, as long as the assumptions are valid with respect to the design tasks at hand.

MBD has been introduced and evolved in many engineering disciplines. For embedded system we find an extraordinary amount of modeling languages and tools that have been introduced for a wide variety of purposes. This fragmentation mirrors the broad applicability of embedded systems, but also the increased specialization and, possibly, the immaturity of the area [14]. It is consequently not surprising to find a number of interpretations of what comprises an MBD approach. These interpretations are related to the usage of models and tools in different industrial domains and specialist disciplines. For example, a control engineering approach to MBD typically uses languages such as Simulink and Modelica, with modeling activities related to function design, prototyping, implementation and testing. In software development, the UML is in frequent usage. Yet other approaches use SDL, Statecharts, or hardware description languages. It is clear that these efforts reflect different design objectives and target different stages of the development process.

4.1. Basic characteristics of MBD

A typical model based development process starts in the very early stages, using models with well defined syntax and semantics for capturing requirements and for designing the functionality. These models are then successively refined, incorporating more detail and peculiarities of the implementation. While formal abstractions are important for analysis and synthesis, visual representations of models are often important for comprehension and communication of designs.

Figure 2 depicts a model based embedded systems development, using abstract descriptions for the design, and tools for generation of executable code or hardware configurations from these descriptions. The figure indicates the trend to synthesize not only functionality, but also parts of the (software) platform. For embedded systems, code generation constitutes an interface between function and implementation specialists. With code generation, some of the manual implementation steps can be removed, leading to a *Y*- or even a *T*-cycle, instead of the traditional *V*-model [2]. However, it is important to understand that there is much more to MBD than code generation. Models are also essential for early design validation and verification, for architectural design, for testing and integration. In many cases, executable environment models also form part of the embedded code, for example for diagnostics or estimation purposes. Other usages of models include information and process modeling. Developed in the right way, models become assets that can be reused within and between projects.

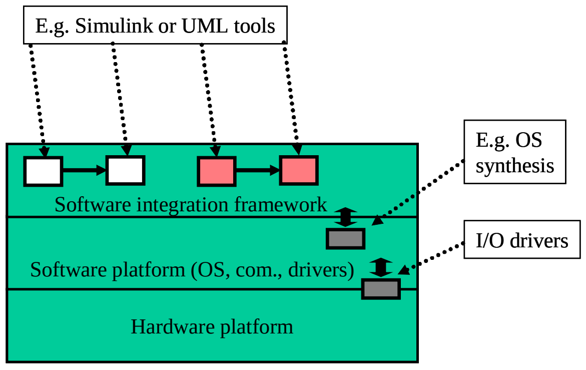
E.g. Simulink or UML tools
E.g. OS synthesis
Software integration framework
I/O drivers
Software platform (OS, com., drivers)
Hardware platform

Figure 2. One illustration of MBD.

MBD provides possibilities to develop and use a multitude of models for different system aspects (i.e. product properties and constraints) such as failure modes of components and their propagation, required timing behavior of a real-time implementation, power consumption, and the behavior of the expected environment. As discussed previously, some models, e.g. describing the intended behavior of a product, can also be the basis for direct synthesis of the implementation, whereas others instead implicitly affect the resulting product structure/behavior by providing one basis for design decisions.

Cornerstones for the successful adoption of an MBD approach include *theoretical foundations* – as a basis for modeling languages, analysis and synthesis, *tool support* – to assist in complexity management by automating design tasks, and *modeling methodology* – providing guidelines for how to model and how to choose an appropriate level of abstraction.

The use of models and supporting tools must always - as the use of any tool - be evaluated within the given context. Increasingly complex products with tough requirements require more powerful tools – thus increasing the need for an MBD approach. The handling of complexity causes the resulting need to strive for abstract descriptions, formal methods and tools which support information management, reuse, automated analysis and automated/ assisted synthesis. With respect to reuse, MBD approaches in principle enable reuse of earlier developed models and of tooling efforts.

4.2 Challenges and CBD related efforts

A challenge for MBD is that of handling the increasing gap to reality and the availability of suitable tools. Methodology and techniques for model validation and verification are therefore essential.

The introduction of MBD is strongly related to and affected by organizational, process and technology constraints. For example, the personnel of the organization needs to be prepared and/or trained to use a certain modeling methodology (their tradition may make it difficult to use a certain approach). The process needs to recognize and take the use of models supporting the development into account. It is important to assess the maturity of the available supporting technology to avoid mismatches between needs and support tools [14].

The increasing exploitation of MBD highlights two important challenges, the *integration of multiple models,* and the *management and reuse of models*.

MBD tools today only support the handling of a limited number of aspects (or views), consequently model and tool integration is today an intense research area. A complete development chain typically involves a multitude of tools and pieces of information that are loosely integrated. Some efforts try to develop 'all-encompassing' languages to cover as many aspects as possible. Examples in this direction include the EAST-ADL, the AADL and the UML. The former two are specialized for vehicular embedded systems development and provide constructs and properties required for analysis of safety, reliability and timing, as well as explicit models of software components [15, 16]. While the UML was initially intended for general purpose software systems, work is being undertaken to provide more capabilities for non-functional aspects by defining UML profiles [2]. Other OMG related efforts focus on establishing the Model Driven Architecture for which a key ingredient is research on the systematization of transformations between models.

Yet other efforts in different ways strive for integration platforms that allow models and tool services to be combined – thus still allowing existing tools and modeling languages to be used, see e.g. [18]. The need to describe multiple aspects is also strongly relevant for CBD and software systems in general. Here it should be of interest to further evaluate and compare model integration approaches with aspect-oriented approaches in software. It does at least seem that the techniques can be equally well applicable to models [19].

The model management problem becomes apparent with an increasing usage of MBD. Working with MBD tools has, in fact, many similarities to CBD. A designer will work by composing model components from libraries. Newly developed models or just components thereof can be stored there as components, for reuse. However, models appear in many versions and variants, are used by many persons, and associated with even more information such as parameters, verification status etc. Reuse of model components and models (component assemblies) faces a reuse problem not far from that in CBD. That is, a pure textual description and interface definitions are not sufficient. Additional contextual information is also required. Keeping track of all model information requires explicit consideration, and tool support for larger systems. Proper configuration management thus has to be in place. This is a rather new problem for MBD in embedded systems, however less so for the more mature area of mechanical engineering [20, 21].

A key challenge for MBD in embedded systems is to combine such best practices with the emerging results in embedded systems MBD.

Several efforts today address model management and techniques similar to those used or advocated in CBD, see for example [22-24]. Within the European project NFCCPP [25], a system reference model has been developed for future generation of automotive fuel cell systems. The model defines standardized system components and sub-systems in terms of well-modularized functional simulation model blocks (e.g., air compressor). To promote continuous system evolution and a component market, the cross-enterprise exchange and integration of functional solutions are supported, covering issues such as independent evaluation of individual components, protection of proprietary information and prevention of unauthorized component usage, as well as configuration management.

5. Comparison

Table 2 compares CBD with MBD, based on the comparison approach established in Section 2.

Table 2. CBD vs. MBD

	CBD	MBD
Drivers	Emphases on reuse across products, time-to market, and component market	Support for design and evaluation; emphasis on complexity control.
Ontology	Targeting software programs, interactions, design- and run-time composition. Structure emphasis.	Targeting entire system including functional solutions and implementation, as well as their run-time relationships. Structure and behavior emphasis.
Organiza-tion	Assuming separated component development processes from system development process	Assuming multiple teams normally in the same organization.
Technology	Relying on component technologies for component specification, integration, management, and code generation.	Relying on modeling techniques and tools for specification, analysis, realization and information management.
Discipline& domain	Software engineering	❶
Lifecycle	Targeting software design, implementation and maintenance.	Targeting a broad spectrum of stages in system development. ❶
Product qualities	Mainly concerning the functionality related effects of reuse (e.g., composability).	Providing description and formalization support for many types of analysis, and of many qualities - ❶.
Complexity-control	Increasing composition granularity, encapsulating and hiding details, and standardizing system components.	Abstracting, separating concerns in views, standardizing in languages, tool automation.
Traceability	Supporting standardization and versions/variants management.	Forming a basis for documentation and information management.
Reuse	Component specifications, components, integration architecture/infrastructure	Providing global context of solutions hence forming a basis of reuse.
Process effective-ness	Bottom-up integration, distribution of work, quality assurance from reuse.	Early V&V!, risk reduction, automation, including techniques such as executable requirements, rapid prototyping, and auto-coding
Process maturity	Well-established in engineering of general purpose software systems	Application domain dependent (e.g., automotive vs. avionics).

❶ The precise scope is defined by the MBD domain/ approach as discussed in section 4.

The differences in emphasis are clearly seen. The two approaches have originated to address different needs with different technology assumptions. While CBD tries to combine a bottom-up and a top-down approach, where the “run-time thinking is strong”, MBD constitutes a top-down approach where knowledge about the systems under development is a critical aspect. While MBD adopts code generation, CBD relies on the reuse of already existing code.

The focus of MBD is traditionally on the early life-cycle stages where the forthcoming product is modeled, evaluated and optimized with respect to the intended functionality and quality attributes. In contrast, CBD emphasizes the creation, integration and maintenance of reusable software solutions, where the functionality related effects of reuse traditionally have been in focus. Formal techniques play an increasingly important role for both approaches to increase their capability for analysis. In MBD, it is assumed that the design teams have a much more complete access to system information. For CBD, however, the applicability of using formal techniques depends on the completeness of the provided and derivable information from components.

Both approaches adopt many concepts found in systems engineering and utilize some fundamental software engineering concepts and technologies.

The architecture concept appears in both approaches, relating to the design of the overall system structures. Although differing in scopes (i.e., software or entire system), architectures in both approaches constitute the basis for solution integration, change control, concurrent engineering and outsourcing. The component concept exists in many modeling efforts at different levels. As a generic term for “system constituent units”, components can have different interpretations. A CBD component, most often refers to binary code (e.g., COM-components) and a concrete execution platform. In MBD, components exist in the form of for example “function blocks”. However, these may, or may not have direct relationships with the final implementation. For example, a block may represent an environment model, defining the context rather than the target system itself. A block may also only be a visual representation, grouping several primitive blocks together. In contrast, in some modeling languages like Rapide and AADL/MetaH, components have a direct link to the coding structure.

Both approaches address complexity control using general principles like information-hiding, separation-of-concern, and standardization. The main difference is that these principles are applied at different levels. Using such measures in CBD is directly related to changing the product itself (e.g., changes in product decomposition structure). However, in MBD there is the additional degree of freedom to use these principles for other purposes such as usability and for analyzing different aspects.

While reuse in CBD targets a complete solution in terms of a run-time executable (e.g., .dll), reuse in MBD is not restricted to a certain level and can hence be applied to various partial solutions such as algorithms, structuring styles, etc.

A further connection point, from MBD to CBD is that of code generation. For example, code generation from models can be used to generate code corresponding to component-internal functionality, where CBD provides the mechanisms for gluing components together, see e.g. [2].

Typically, MBD approaches provide quite a lot of flexibility, with options to generate only pieces of code, or even complete systems. This flexibility also allows incorporating and reusing legacy code. Some CBD efforts, on the other hand, provide the possibility to configure a system and generate glue code.

6. Discussion

To summarize many of the topics previously discussed, consider Fig. 3, which depicts system development in terms of a number of conceptual levels (or design stages), each treating a particular level of abstraction. As indicated, the product and its constituents are, at each design level, typically represented by a multitude of models representing different aspects, concerns or views. design models are successively refined and mapped (e.g. one to one, or many to one), with varying precision and concreteness.

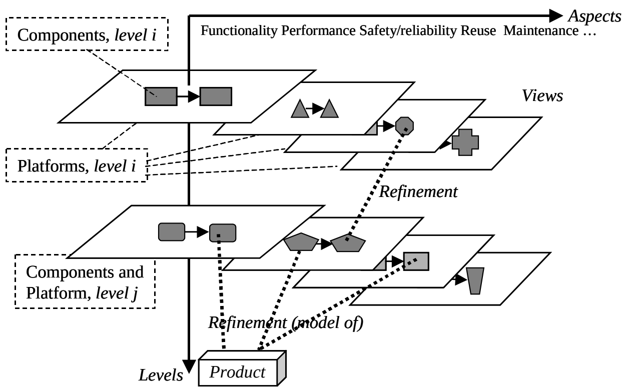
Aspects
Components, level i
Functionality Performance Safety/reliability Reuse Maintenance …
Views
Platforms, level i
Refinement
Components and Platform, level j
Refinement (model of)
Levels
Product

Figure 3. A conceptual view of CBD and MBD.

Each level can be seen to be composed of components and one or more platforms. It is however important to note as discussed previously that a model component is a much more general notion compared to a CBD component. At higher levels, platforms can correspond to a simulation environment and other analysis facilities, together with a database for storing the models. At the bottom level (the product), the platform corresponds to hardware and system software such as an operating system, and executable software components. Each platform provides its own services to its components.

MBD is related to all higher levels. Since multiple models are used a key challenge becomes the correlation and integration between the different models. CBD corresponds to the bottom – product level – and to (at least) one higher level, the description of the component models. As discussed in section 3, we can say that there is a trend of CBD to treat higher design levels to ensure reuse and composability. As a consequence, the boarder-line between CBD and MBD is becoming fuzzy.

The differences between CBD and MBD are related to the mappings between the levels. One assumption in MBD is that, under given constraints and formally specified conditions, a transformation from a higher level to the lower level can be achieved more or less automatically. The properties of the lower level are the results of a synthesis from models on the higher level. CBD assumes that the automatic transformation is not the most efficient way of reuse; rather all levels should be reused in its entirety. There is no need to regenerate the lower level; it is more efficient to reuse it directly. However, problems occur when certain conditions change, functionality is to be extended or requirements change. A related problem is that of keeping documentation and code consistent. The strength of MBD is then the ability to reuse, but at a higher level. However, for this purpose, MBD is strongly relying on tool support compared to CBD. CBD has to handle changes by component adaptation or by developing new components.

The MDB approach is obviously more efficient in domains in which the transformations are feasible and well defined. Examples of such domains are control and signal processing. The CBD approach is more efficient for applications which are more difficult to formalize, such as Human-Machine Interfaces.

There are a number of ongoing efforts that investigate the integration of CBD and MBD. Such efforts indicate the potential and feasibility of such integration, see for example [16, 18], where a strong point is the ability to utilize and combine the component based approach of “systems architecting” together with the MBD approach of analyzing different aspects and synthesizing different parts of the system, thus promoting complexity management.

CBD addresses issues such as component and architecture specifications, reuse and product evolution. These techniques from CBD can be applied in MBD on different levels and are important to support complexity management at the model level. However, it must be realized that CBD techniques are currently mainly delimited to functionality aspects at the implementation level. For the organization of a particular structure, component identification and specification is important. The techniques in CBD for component specifications can be adopted – in particular the ability of using strong (e.g. including application specific protocols) and contract-based interfaces. In addition, CBD highlights the needs for rigorous specifications at the lower levels.

From the above discussion, it is clear that there are many efforts within CBD, striving to develop specifications (models) of components, their interactions, etc. to support configuration and composability. There are obviously many issues here where MBD can play an important role for CBD. Composability is a key issue for CBD. Many modeling techniques support compositional analysis for example dealing with logical, performance, timing, reliability etc. properties of composed systems. This relates back to the possibility with MBD to describe different views of a system. An MBD approach is also required to ensure reuse by providing context models, model-based verification and model-based testing.

7. Conclusions

While CBD traditionally is concerned with handling binary software units, their integration and reuse, MBD uses the concept of formalized descriptions for the purposes of supporting communication, analysis and synthesis during the product development. From the analysis, it is clear that the approaches complement each-other, and that they to some extent overlap.

It is concluded that concepts from MBD and CBD can reinforce one and another, and that both concepts are required for the development of complex systems. A remaining central challenge is the full integration of the two, taking multiple aspects into account.

8. Acknowledgements

This work was supported in part by the Swedish Strategic Research Foundation (the SAVE project) and by the European Commission through the ARTIST2 Network of Excellence. We wish to acknowledge useful feedback from the anonymous reviewers.

9. References

1. The SAVE project: <http://www.mrtc.mdh.se/SAVE/>
2. B Bouyssounouse,; J Sifakis, (Eds.). Embedded Systems Design - The ARTIST Roadmap for Research and

Development Series: *LNCS*, Vol. 3436 2005, Springer.

3. M Törngren, DJ Chen, I Crncovic. *Component based and Model based development in the context of embedded systems: Characterization, comparison and case studies*. Tech Rep, KTH 2005. TRIA-MMK-2005-15, ISSN 1400-1179, ISRN/KTH/MMK/R-05/15-SE
4. B Artemis. *Report by the High-level Group on Embedded Systems. European Commission*, 2003 - <ftp://ftp.cordis.lu/pub/ist/docs/dir_c/ems/everything-final_en.pdf>
5. E Rechtin, MW Maier. *The Art of System Architecting,* CRC Press. 1997.
6. AT Bahill, B Gissing, Re-evaluating systems engineering concepts using systems thinking*, IEEE Trans on Systems*, Man, Cybernetics, Part C, Vol: 28, Issue:1, Nov.1998.
7. DW Oliver, TP Kelliher, JG Keegan Jr., *Engineering Complex Systems with Models and Objects*, McGraw-Hill, 1996.
8. B Prasad. *Concurrent Engineering Fundamentals, Volume I: Integrated Product and Process Organization*. New Jersey: PTR Prentice Hall, 1996.
9. I Crnkovic and M Larsson. (editors), *Building Reliable Component-Based Software Systems*, Artech House Publishers, ISBN 1-58053-327-2, 2003.
10. KC Wallnau. Volume III: *A Component Technology for Predictable, Assembly from Certifiable Components*. Tech Rep, SW Eng Institute, CMU, April 2003.
11. JM Jazequel, B Meyer. Design by contract: the lessons of Ariane. *Computer*, Volume: 30, Issue: 1, page(s): 129 – 130, Jan. 1997. IEEE Computer Society.
12. R Stevens, P Brook, K Jackson & S Arnold. *Systems Engineering - coping with complexity.* Pearson Education 1998. ISBN 0-13-095085-8
13. F Neelamkavil. *Computer simulation and modeling*. John Wiley & Sons Inc, 1987.
14. M Törngren and O Larses. *Characterization of model based development of embedded control systems from a mechatronic perspective - drivers, processes, technology and their maturity*. Tech Rep, KTH 2004. TRITA-MMK 2004:23. ISSN 1400-1179. ISRN/KTH/MMK/R-04/23-SE.
15. U Freund, O Gurrieri, J Küster, H Lonn, J Migge, M-O Reiser, T Wierczoch and Weber: An Architecture Description Language for developing Automotive ECU-Software. *INCOSE 2004.*
16. M Törngren, N Adamsson, and P Johanessen. Lessons Learned from Model Based Development of a Distributed Embedded Automotive Control System. *SAE World Congress*, Detroit, 2004. SAE paper no. 2004-01-0713.
17. Atkinson C., Bunse C., Wust J., Driving component-based software development through quality modeling. Component-based software quality, *LNCS 2003*. Springer-Verlag Berlin Heidelberg, A. Cechich et al. (Eds).
18. *Workshop on Tool Integration in System Development at ESEC/FSE 2003* 9th European Software Eng Conf and 11th ACM SIGSOFT Symp on the Foundations of SW Eng, Helsinki, Finland, Sept. 1-5, 2003.
19. J Gray, T Bapty, S Neema, J Tuck. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, Vol. 44, Issue 10, October 2001.
20. I Crnkovic, U Asklund. and DA Persson, *Implementing and integrating product data management and software configuration management,* Artech House Publishers, 2003.
21. U Sellgren, C Hakelius. A Survey of PDM Implementation Projects in Selected Swedish Industries. *ASME Design Eng Tech Conf*, August 18-22, Irvine, California, 1996.
22. M Tiller. Model Management Tools and Technologies. *Modelica Automotive Workshop 2002*, Dearborn, MI, USA, 2002.
23. K Steppe, D Garlan, G Bylenok, B Schmerl, K Abirov, and N Shevchenko. Tool Support for Model Based Architectural Design for Automotive Control Systems. *First European Workshop on Model Driven Architecture with Emphasis on Industrial Application*, Enschede, The Netherlands, March 17-19, 2004.
24. S Neema, J Sztipanovits, G Karsai1, and K Butts. Constraint-Based Design-Space Exploration and Model Synthesis. *Proc. Of EMSOFT 2003*, LNCS 2855, pp. 290–305. Springer-Verlag, Berlin Heidelberg 2003
25. CJ Sjöstedt, DJ Chen, I Faye, T Huelshorst, A Kells, I Harkness, C Schönfelder. Virtual Component Testing for PEM Fuel Cell Systems – An Efficient, High Quality and Safe Approach for Suppliers and OEM´s, *3rd European PEFC Fuel Cell Forum*, Lucerne, Swiss, July 4-8, 2005.