17 CoCoME Jury Evaluation and Conclusion

Manfred Broy[1], Johannes Siedersleben[2], and Clemens Szyperski[3]

[1] Technische Universität München, Germany
[2] T-Systems International, Germany
[3] Microsoft, USA

Abstract. The CoCoME Jury attended the two-day seminar with presentations from all participating teams. The jury provided individual feedback to each of the teams. This chapter is an attempt at both organizing the contributions into groups of related themes and summarizing some of the more salient feedback. The jury concludes with a few observations about the CoCoME contest as such and its overall outcome.

17.1 Introduction and Overview

As already described in Chapter 2, the main goal of the CoCoME contest is to evaluate and compare the practical appliance of the existing component models and the corresponding specification techniques. Based on a UML-based description of CoCoME, a provided sample implementation, and test cases, the participating teams elaborated their own modeling of CoCoME, applying their own component model and description techniques. After an internal peer review each team presented their results in a joint GI-Dagstuhl Research Seminar, where an international jury–the authors of this chapter–was present.

The jury's task was to evaluate and compare the different modeling approaches based on the respective team's presentation at the seminar as well as the team chapter, which was provided to the jury ahead of the seminar. It became clear that it is not possible to come up with a unified ranking of the presented approaches, based on how well each modeled CoCoME. This is because the approaches varied widely with respect to each approach's target objectives, used component model, applied description technique, modeled part of CoCoME, and the resulting benefits and outcomes of the modeling effort.

Rather, it turned out that each approach has its individual strengths and drawbacks. For that reason, the jury and the organizers decided during the GI-Dagstuhl Research Seminar not to come up with a ranking, but instead to provide for each approach an individual non-comparing evaluation and assessment that could be one starting point to further improve the specific approach.

In the following sections these individual jury evaluations will be presented. To enable some comparative reading, the jury organized the various approaches into a number of groups. The first group covers *semi-formal modeling approaches*. Instead of an own underlying formal semantic model, these approaches are UML-extensions and use the semi-formal model and informal and more or less intuitive semantics of UML.

A. Rausch et al. (Eds.): Common Component Modeling Example, LNCS 5153, pp. 449–458, 2008.

The next group, also the largest, comprises *formal models focused on functional correctness*. This group can be subdivided into three subgroups with respect to the underlying formal semantics: The first subgroup has *formal state semantics*, the second has *formal algebraic semantics*, and the third subgroup has an *formal object semantics*. All approaches in this subgroup share the same overall goal: The resulting models are used to verify and validate the functional correctness of the resulting models. Therefore, these approaches provide to varying extend forms of consistency checking, refinement calculus, simulation support, and code generation.

The third group is about *formal models focused on behavior and quality properties*. The main goal of these models is to enable the modeler to analyze, simulate, and predict the capabilities of the modeled solution with respect to specific behavior or quality properties, such as performance or reliability.

The last group contains *models tailored for a specific application domain*, such as grid computing. Hence, the used component model and the applied description techniques are domain specific and not general purpose. The main goal of these approaches is it to come up with a more specific and restricted model and description technique with respect to the target domain and thus provide results on higher semantic level, such as functional correctness or prediction of quality properties.

17.2 Semi-formal Modeling

KobrA. KobrA is a UML-based method for describing components and component-based systems. The acronym (derived from German) stands for "Component-based Application Development". The approach makes use of UML as a specification and realization language, but other languages can be used. The basic goal of KobrA is to allow the design of a complex system to be split into separate parts (i.e., components), which can be tackled separately. KobrA focuses on describing the architecture of components and component-based systems at the design level. Therefore, the benefits of applying KobrA include the ability to experiment with potential designs before committing to concrete implementations, the availability of easy-to-read descriptions (e.g. system specification) that can be understood by (and discussed with) customers and users, and the creation of a platform-independent record of the system structure and project design decisions.

The recognized biggest obstacle to practical adoption of KobrA at the present time is the lack of a dedicated tool that understands KobrA and is able to ensure that UML is applied in a KobrA-compliant way. Other aspects to be considered/improved are: the specification of the non-functional properties of components and the provision of clear guidelines on how to use KobrA in a more formal style. The jury was quite perplexed about the actual usefulness of KobrA because of its strong resemblance to standard UML.

Rich Services. Rich Services is an approach leveraging the SOA (Service-Oriented Architectures) principles. It models an application as a set of services

communicating via a bus. Services may be hierarchically decomposed–a service is recursively modeled as a set of services connected to a bus. In addition to functional decomposition, Rich Services allows for capturing cross-cutting concerns (security, policies, failure management, QoS, etc.) using dedicated services that intercept and route messages flowing through a bus. From the deployment point of view, Rich Services may be mapped to the existing ESB middleware (e.g. Mule).

Rich Services utilizes Message Sequence Charts (MSCs) to define interactions (including temporal properties). The existing tool support is then able to synthesize executable code from MSCs. Verification is possible by translating MSCs to Promela.

Another important aspect of Rich Services is that they come with a service-oriented development process, which clearly separates a system's logical model from its implementation. This allows, for example, to introduce performance optimizations at deployment time by flattening the system's architecture.

In modeling CoCoME, the authors followed the Rich Services development process from requirements to actual deployment. It has to be noted, that this resulted in an architecture quite different from the original CoCoME architecture to be modeled. Where the other approaches tried to model the orginal architecture as intended, the Rich Services approach is hard to compare to the other approaches, as it remains unclear which benefits the other approaches would have had, if they had also changed the architecture of CoCoME to optimize for their methods. However, the Rich Services approach proved to be usable in capturing service contracts and different cross-cutting concerns.

17.3 Formal Modeling, Focused on Functional Correctness

Formal State Models

rCOS. rCOS is a relational calculus for object and component systems based on Hoares and Hes Unifying Theories of Programming (UTP) for object-oriented and component-based programming. It allows to model the system at different level of abstraction. Between these levels a clear refinement relation is provided. At each level, the model can be described by a set of different views–the static structural view, the interaction view, the dynamic view– as well as their timing apspects.

Thereby the rCOS approach claims to support a formal development process starting out with analysis and in an stepwise refinement approach finally ending up with a component model ready for an object-oriented implementation. The main promised benefits are: consistency rules between the various views in a modeling step, guided refinement rules from one modeling step to the next one, and finally a formally founded specification technique to support verification and analysis on the model level.

The rCOS team has modeled two use cases on the analysis model level. This cutout has been refined to an object-oriented model. Finally, the object-oriented model has been encapsulated in a set of components.

The main strength of the rCOS approach lies in its clear and straight-forward object-oriented analysis model. The relationship to the component model should be further improved as it is not clear at all how the component model will be derived from the object-oriented analysis model and what the relationship between these two models formally is. Finally, the authors could not show which properties were actually proved resp. proveable by their model. Hence it is not determinable whether one main goal of rCOS–to verfiy and analyze properties of the system–can be reached or not.

CoIn. The CoIn approach to modeling the CoCoME application is based on the concept of component-interaction automata, which are used to capture the behavior of components in the system using a labeled transition system. For each primitive component, a distinct automaton is created that models the component behavior. In the case of composite components, the automaton is obtained through composition of automata associated with primitive components. The composition operator is parametrized, which allows different kinds of composition to be performed in different scenarios. Through composition, this approach allows for the constructing of complex, whole-picture models, based on much simpler behavioral models of primitive components. Using the model of an entire system, various properties specified in linear temporal logic, modified for the use with component-interaction automata (CI-LTL), can be verified.

The main strength of the presented approach lies in its expressive power and flexibility. The authors of the CoIn approach chose to create a model of the CoCoME application based mainly on its prototype implementation as the most precise specification, using the UML models mainly to extract structural information concerning system architecture. By using different kinds of composition (cube-like, star-like, and handshake-like) depending on the context of composition of primitive components, they were able to create a very detailed model of the whole system. Using the UML use cases and test scenarios for the prototype implementation, they were able to formulate a set for properties, the validity of which has been verified using the DiVinE model checker.

The level of detail, completeness of the model, and the model checking tool support are noteworthy and it is evident that the CoIn approach is very suitable, thanks to its powerful composition operator, for construction of models in a bottom-up manner. Alternatively, the CoIn approach could be used to model the CoCoME application in a top-down manner, which is often more suitable in the initial phases of system design. However, in top-down approach to construction of a system model, the power of the composition operator would play a minor role, leaving the designers alone with the task of creating the model. It can be argued though, that a model created in a top-down manner would not need to be nearly as complex as the one obtained in bottom-up manner. The authors did not model/verify any extra-functional properties because the CoIn approach does not support that. However, this cannot be considered a serious drawback, mainly because the CI automata were designed for the different goal of the verification of functional properties.

Formal Algebraic Models

Focus/AutoFocus. The Focus/AutoFocus approach provides a top-down development process based upon the formally founded mathematical model Focus (timed infinite streams) as the semantic model for component behaviour. The development process (and the formal model) is designed for distributed reactive systems and starts out with informal requirements and ends up–over various refinement steps–with an implementation model.

The main goal of the approach is to provide a development processes that guides the software engineer from given informal requirements via well defined refinement steps to an implementation model, which can be used to simulate the system and finally generate an implementation of the system. As the model is based on a formally founded mathematical system model, the varification of the correctness of the refinement steps from one abstraction level to the the next more detailed level is guaranteed. Moreover analysis, simulation, verification and code generation can be supported by additional tool support or external tool integration.

The modeling of the Focus/AutoFocus team focused on the cash desk part of CoCoME. As Focus/AutoFocus is an approach tailored for reactive systems, the information system part of CoCoME was not modeled by the team. Moreover, the non-functional and extra quality properties of CoCoME were also not modeled by the Focus/AutoFocus approach.

Hence, as Focus/AutoFocus claim to have a rigid and formally founded development process supporting correct refinement steps from requirements to an implementaion model, the results of applying Focus/AutoFocus are mainly focused on the cash-desk part of CoCoME. Therefore, the Focus/AutoFocus team showed a convincing approach following a clear and rigid development process. Moreover, the underlying formal system model enabled the team to generate an executing system out of their model, showing an online simulation of the system. The main deficit of the approach is that Focus/AutoFocus is not being able to model instantiation and thus the dynamic structure of a system–the running system is modeled as having fixed structure. Especially, when modeling the information-system part of CoCoME such a capability to model at instance level would be required. Moreover, the concept of blocking method calls as a basic communication primitive would also be required. Nevertheless, for the reactive part of the system, the Focus/AutoFocus approach is a well tailored and formalized approach providing convincing results to its users.

Java/A. Java/A is a component model based mostly on several concepts of UML 2.0 which it extends by analysis and verification techniques. Development of applications using the Java/A component model is encouraged through its support of a Java-like programming language, which allows embedding structural and behavioral information concerning application architecture directly in the application source code. This tight coupling of architectural description and implementation allows programmers and maintainers of Java/A applications to avoid architectural erosion, which typically plagues approaches where architectural and behavioral description is kept separately and sometimes without direct

association with implementation. This level of integration in Java/A allows for more natural and direct mapping of UML 2.0 component concepts into program code.

Behavioral description of Java/A components is split between a component and its ports containing pairs of required/provided interfaces. The description comes in the form of UML state machines, which are then translated into I/O transition system, on which formal verification is performed.

Concerning modeling the CoCoME application, the authors of the Java/A approach used mainly the UML use-case descriptions and sequence diagrams to create the model of the system. The resulting model captures the architecture of the whole system, but since the authors focused mainly on the embedded part of CoCoME, behavioral model and analysis have only been done for the CashDeskLine component.

The main strength of the Java/A functional analysis lies in the ability to support modular analysis and verification techniques through derivation of global properties (composite components) from local ones (primitive components). The functional analysis supports checking for correctness of primitive components with respect to their port protocol and checking for deadlock-freedom of a composite. However, the notion of observational equivalence, required to support the analysis, may be considered too strong.

The authors have also performed quantitative analysis of the system, using the specification of extra-functional properties. They had explicitly modeled Use Case 1, using a stochastic process algebra (PEPA), and have shown that the benefit of the express-checkout cash desk does not really match intuitive expectation. Since neither the Java/A component model nor the programming language supports capturing quantitative aspects of an architecture, it may be beneficial to consider integrating description of quantitative aspects into Java/A, similar to integration of the structural and behavioral aspects.

Formal Object Models

Cowch. The Cowch approach is based on a hierarchical component model for object-oriented programming, called the Box Model. A box is a runtime entity with state and a semantics-based encapsulation boundary. The implementation of a box consists of a number of classes and interfaces of the underlying programming language. To structure communication between boxes directed ports and channels are used. Ports are named interfaces and can be incoming or outgoing. All box boundary crossing references must be modeled by using ports. A graphical UML-like description technique is provided for the Cowch approach.

The main goal of the Cowch approach is to fill the gap between high-level description techniques and architecture modeling on the level of component-based architectures and object-oriented programming languages. Thus the functional and strcutural features of the designed component architecture can be enforced on the object level by verification as well as static and dynamic analysis.

The Cowch team has completely modeled the functional and structural properties of CoCoME, where the main focus was on the runtime structure. The

functional and behavioural part was not modeled; instead, relying on the Java code for this purpose. The non-functional and qualitiy properties have not been modeled.

The main strength of the approach is the clear semantic foundation on top of an object-oriented programming language. Thus the Cowch team was able to model the complete CoCoME. However, this benefit is also the main drawback of the approach. Cowch cannot provide a high level of abstraction - all is more or less treated at code level. And, as the approach is still under development, the tool support and thereby the achievable verification and analysis support and results are currently not clear at all.

DisCComp. DisCComp (Distributed Concurrent Components) is a formal model based on set-theoretic formalizations of distributed concurrent systems. It allows modeling of dynamically changing structures, a shared global state and asynchronous message communication, as well as synchronous and concurrent message calls. DisCComp provides a sound semantic model for concurrently executed components that is realistic enough to serve as a formal foundation for component technologies currently in use. The approach is supported by tools for code generation as well as simulation and execution of such specifications for testing purposes. At present, the specification technique is adequate for functional properties, while it is not able to deal with non-functional properties. The jury was quite unclear on the actual CoCoME coverage achieved, since the presentation was mainly focused on the formal model.

17.4 Formal Models, Focusing on Behavior and Quality Properties

Palladio. Palladio is a complete component modeling approach including both modeling and performance analysis facilities in an integrated and automated framework. To this end, the Palladio approach includes a meta-model for structural views, component behavior specifications, resource environment, and the modeling of system usage; and multiple analysis techniques ranging from process algebra analysis to discrete event simulation.

Palladio supports hierarchical composite components and is able to model the usage profile of a system, the component behavior, the composition/assembly structure, the deployment context, and the resource environment. For each of these models, the performance relevant aspects (delays, data abstractions, etc) are modeled and analyzed. All analyses can be performed without an implementation, supporting early design time predictions. This way, Palladio enables software architects to analyze different architectural design alternatives supporting their design decisions with quantitative performance predictions. The CoCoME modeling and analysis has been focused on the most complex use case number 8, providing a nice exposure of the Palladio potential. The good discussion of the approach and the extensive performance analysis carried out on the case study have positively impressed the jury.

KLAPER. KLAPER plays a specific role in this context. It does not provide a model to design component-based systems, but it acts as an intermediate language between design meta models and analysis meta models. While design models support the design of a software systems in a notation and terminology familiar with a software designer, analysis models enable the application of specific analysis techniques. As from both kinds of models (design and analysis) several kinds with specific benefits and drawbacks exist, there are good reasons to support the translation of each design meta model to as many as possible analysis meta models. However, the definition and implementation of $m \times n$ transformations (given m design meta models and n analysis meta models) is not realistic. The solution to such a problem is known from compiler construction: an intermediate language defines a common format where the m design meta models define transformations to, as well as transformations are defined from the intermediate language to the n analysis models. By this, only $m + n$ transformations are required.

From a modeling perspective, KLAPER is defined by a MOF meta-model which includes ways to specify components and software architectures as well as resources. When modeling CoCoME with KLAPER, one started to translate an UML model into KLAPER and than transformed an KLAPER model into a layered queuing network (LQN). The prediction results on resource utilization and response time were impressive, although no direct comparison to the implementation was made, as the implementation deviates from the specification in performance-relevant aspects. The jury was impressed by the elegance of the modeling and the ability to perform analyses on response time and utilization. Areas for future work include improvements of scalability as well as a systematic way to feed back the analysis results into the original design model.

Fractal. Fractal is a hierarchical component model which provides means to describe architectures by the definition of component bindings and resources. Specifically, components have membranes (encapsulation of the inner), content (either wired sub-components or an implementation) and controllers with specific control interfaces. The behavioral view can be specified by traces in the FractalBPC language (which originates from the SOFA component model). Modeling the deployment view is not specified in Fractal, but implementation specific. Noteworthy are the various formal consistency checks, such as component interaction, compliance of inner components with outer component specification and consistency between implementation and interfaces. In particular, it also supports the description of dynamically changing architectures. It was initiated by researchers of the France Telekom where also larger case studies were modeled in Fractal. By this, Fractal forms one of the stronger validated models of the CoCoME contest.

CoCoME was modeled well in Fractal, however, the communication bus was modeled as a separate component which was not present in the original specification. This was motivated by the need to explicitly introduce interfaces which were not present in the original CoCoME. Except for that, using Fractal for modeling the CoCoME proved easy and beneficial due to the extensive compliance

checks. Modeling extra-functional properties is not yet possible in Fractal, although in the specific model of the CoCoME also performance data was collected by monitoring.

SOFA. SoFA (Software For Appliances) is a component model which allows to define the structural view of software architectures as well as behavioral specifications of basic and composed components. Specific to it is the frame protocol which specifies the behavior of a composed component and which can be checked against the composition of inner component protocols. Protocols are modeled by regular expressions. However, since sending and receiving a call as well as call and return of an invocation are modeled explicitly, asynchronous behavior can be specified also. Compliance checking (interoperability, substitutability, compliancy between frame-protocol and inner protocols) is realized by a connection to Promela which allows to use the SPIN model checker.

There is also the deployment view where the distribution of compoennts–their mapping to computing resources–can be specified. Finally, SoFA includes a performance view which is linked to the deployment and behavioural view. Interesting is the comparison to the conceptually related Fractal model: while SoFA allows similar checks also in comparable time, its specification is considerably more compact (1500 LOC vs. 2700 LOC), which makes SoFA a well applicable language with good verification possibilities.

17.5 Models Tailored for a Specific Application Domain

GCM/ProActive. GCM is a component model based on Fractal. It uses the Fractal concept of hierarchical components with provided and required interfaces and explicit treatment of control interfaces. GCM focuses on grid computing, thus in addition to standard Fractal features, it adds the possibility of asynchronous method calls using *futures* and defines collective interfaces (multicast and gathercast) to ease synchronization and work distribution. As an ADL, GCM uses FractalADL enhanced by the concept of a *virtual node*–an abstraction of the physical infrastructure to which an application is deployed.

From the behavior point of view, GCM uses a low-level model called pNets. It is basically a network of parameterized LTSs tailored to suit the needs of grid computing. GCM also offers a high-level language called JDC (Java Distributed Component Specification Language). It combines ADL specification with a behavior specification written in a notation similar to Java. The aim of JDC is providing a modeling language from which the pNets behavior models and skeletons of implementation artifacts could be generated. In addition to textual specification, graphical design is also supported via a UML editor called CTTool.

Modeling of CoCoME was possible in GCM. Especially the distribution was rather seamless thanks to the *virtual node* abstraction. The actual behavior specification and analysis was done by employing the CTTool and eventually verified by LOTOS/CADP as the whole toolchain spanning from JDC to pNets has not been completely implemented yet.

17.6 Conclusions

Each of the modeling approaches, when applied to CoCoME, exhibited an interesting set of strengths and weaknesses. While, pointwise, there is much to be learned from each of the approaches, the jury concludes with a mixed message. CoCoME's top-level challenge was and is about all-of-system modeling and none of the approaches enabled comprehensive modeling at that level. To be fair, the approaches at hand are the results of ongoing research and it isn't the purpose of research efforts to deliver fully rounded "complete" solutions. However, the gap between practical applicability and demonstrated modeling capability remains significant, leaving much room for further work.