Component-Oriented Programming (WCOP 2004)*

Jan Bosch[1], Clemens Szyperski[2], and Wolfgang Weck[3]

[1] University of Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, The Netherlands
Jan.Bosch@cs.rug.nl
http://segroup.cs.rug.nl
[2] Microsoft, USA
CSzypers@microsoft.com
http://research.microsoft.com/~cszypers/
[3] Independent Consultant, Switzerland

Abstract. This report covers the ninth Workshop on Component-Oriented Programming (WCOP). WCOP has been affiliated with ECOOP since its inception in 1996. The report summarizes the contributions made by authors of accepted position papers as well as those made by all attendees of the workshop sessions.

1 Introduction

WCOP 2004, held in conjunction with ECOOP 2003 in Darmstadt, Germany, was the ninth workshop in the successful series of workshops on component-oriented programming. The previous workshops were held in conjunction with earlier ECOOP conferences in Linz, Austria; Jyväskylä, Finland; Brussels, Belgium; Lisbon, Portugal; Sophia Antipolis, France; Budapest, Hungary; Malaga, Spain and Darmstadt, Germany The first workshop, in 1996, focused on the principal idea of software components and worked towards definitions of terms. In particular, a high-level definition of what a software component is was formulated. WCOP97 concentrated on compositional aspects, architecture and gluing, substitutability, interface evolution and non-functional requirements. In 1998, the workshop addressed industrial practice and developed a major focus on the issues of adaptation. The next year, the workshop moved on to address issues of structured software architecture and component frameworks, especially in the context of large systems. WCOP 2000 focused on component composition, validation and refinement and the use of component technology in the software industry. The year after, containers, dynamic reconfiguration, conformance and

* The title of this report should be referenced as “Report from the ECOOP 2004 Eighth International Workshop on Component-Oriented Programming (WCOP 2004)”.

quality attributes were the main focus. WCOP 2002 has an explicit focus on dynamic reconfiguration of component systems, that is, the overlap between COP and dynamic architectures. Last year, the workshop addressed predictable assembly, model-driven architecture and separation of concerns. The 2004 instance of the workshop had a technical and an industrialisation focus. In the technical focus, predicting performance, testing and aspect-oriented software development in the context of component-based software engineering were discussed. In the second focus, issues including the application of CBSE in the context of embedded systems, an analysis of the past, present and future of components and the results of a large, European initiative, CBSENet, were discussed. WCOP 2004 had been announced as follows:

WCOP 2004 seeks position papers on the important field of component-oriented programming (COP). WCOP 2004 is the ninth event in a series of highly successful workshops, which took place in conjunction with every ECOOP since 1996. COP is the natural extension of object-oriented programming to the realm of independently extensible systems. COP aims at producing software components for a component market and for late composition. Composers are third parties, possibly the end users, who are not able or willing to change components. Several component technologies emerged, including CORBA/CCM, COM/COM+, J2EE/EJB, and .NET. There is an increasing appreciation of software architecture for component-based systems and the consequent effects on organizational processes and structures, as well as the software industry in large. WCOP 2004 emphasizes the dynamic composition of component-based systems and component-oriented development processes. Dynamically composable software needs clearly specified and documented contracts, standardized architectures, specifications of functional properties and quality attributes, and mechanisms for dynamic discovery and binding. A typical example is web services. A service is a running instance that has specific quality attributes, while a component needs to be first deployed, installed, loaded, and instantiated. Comparing service and component composition models is interesting and a proposed workshop focus. Flexible development processes (such as agile ones) and component-based development support each other in that the use of existing components can reduce the development effort. Positions on development processes relating to the use of components are welcome. Finally, we also solicit reports on practical experience with component-oriented software, where the emphasis is on interesting lessons learned.

COP aims at producing software components for a component market and for late composition. Composers are third parties, possibly the end users, who are not able or willing to change components. This requires standards to allow independently created components to interoperate, and specifications that put the composer into the position to decide what can be composed under which conditions. On these grounds, WCOP'96 led to the following definition: A component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.

Often discussed in the context of COP are quality attributes (a.k.a. system qualities). A key problem that results from the dual nature of components between technology and markets are the non-technical aspects of components, including marketing, distribution, selection, licensing, and so on. While it is already hard to establish functional properties under free composition of components, non-functional and non-technical aspects tend to emerge from composition and are thus even harder to control. In the context of specific architectures, it remains an open question what can be said about the quality attributes of systems composed according to the architecture's constraints. As in previous years, we could identify a trend away from the specifics of individual components and towards the issues associated with composition and integration of components in systems. While the call asked for position papers on the relationship to web-services and services in general, we did not receive papers on that subject. An emphasis on anchoring methods in specifications and architecture was noticeable, going beyond the focus on run-time mechanisms in the previous year. Ten papers by authors were accepted for presentation at the workshop and publication in the workshop proceedings; for one paper the authors withdrew the paper from the workshop; for one paper no presenter showed up at the workshop. About 25 participants from around the world participated in the workshop. The workshop was organized into four morning sessions with presentations, one afternoon breakout session with three focus groups, and one final afternoon session gathering reports from the breakout session and discussing future direction.

2 Presentations

This section summarizes briefly the contributions of the nine presenters, as grouped into four sessions, i.e. analysis, performance, direction of CBSE and application of CBSE.

2.1 Analysis

The first paper, presented by Thomas Cottenier, discusses the use of aspect-oriented programming (AOP) techniques for non-invasive component adaptation at load-time, compose-time and runtime. The paper starts from the observation that existing approaches to AOP operate at the code level, rather than at the component level, destroy the locality of runtime control flow and break component encapsulation. These issues complicate the predictability and certification of component-based systems exploiting aspects. The paper proceeds with identifying that different types of aspects are used in different phases of the lifecycle, e.g. resource management aspects at deployment time, adapter aspects at composition time and dynamic adaptation aspects at run-time. The authors propose the use of an aspect-sensitive component profile that provides expressiveness for semantic contracts, temporal properties, quality attributes and access control contracts. This component profile is than used in an MDA-style form of aspect weaving to obtain the desired component and system behaviour. Judith Stafford presented the second paper. This paper starts with the observation that software development is compositional in nature, but that most analysis approaches assume complete system. Consequently, the system needs to be flattened, which is expensive and may be infeasible if external, e.g. COTS, components are present do not allow access to their internals. To achieve compositional analysis, each component in a system should be accompanied with an analytic asset. Once these assets are available, compositional analysis can be performed by analyzing input-output pathways and the specific configuration of the components in the system. The paper raises, as open issues, the selection of analysis algorithms, the representation of analytical assets and packaging of these assets.

2.2 Performance

Viktoria Firus presented the first paper in the performance session. This paper discusses the notion of parametric performance contracts for components and the compositionality of these contracts. The initial observation in the paper is that early evaluation of software architectures is important as the design decisions taken during the architecture design phase have a large impact on performance. The evaluation can be used to select the best design decision. The authors identify three dimensions that are of importance for quality predicting models, i.e. validity, compositionality and parametricity. Parametricity refers to the need for prediction models to reflect the context dependencies of a component. The authors then proceed to define parametric performance contracts. The approach taken models the performance of a component service by a distribution which itself depends on the performance distribution of the external services called. Then an overview of existing work was given. The discussion following the paper presentation focused on the difficulty of providing accurate performance predictions, especially since many performance issues result from resource contention rather than execution. An alternative approach was presented by Trevor Parsons. The authors present a framework for automatically detecting and assessing performance anti-patterns in component-based systems using run-time analysis. The authors start by highlighting the importance of performance in the context of component-based systems. Problems of performance analysis in these systems results from the large amounts of data, the lack of support for achieving solutions and the lack of bottleneck identification. The authors attack the problem by identifying performance anti-patterns. These patterns are identified by proposed framework. The framework consists of three main parts, i.e. monitoring, detection and assessment and visualization. The authors employ data mining algorithms to deal with the large amounts of data generated during monitoring. During the discussion, the main topic was that the approach seems promising, but needs to be employed in practice before its applicability can be determined.

2.3 Direction of CBSE

Different from the first two sessions that were very technical, the third session was of a more conceptual nature, evaluating the notion of component-based software engineering (CBSE). The first paper, presented by Jean-Guy Schneider, discusses an assessment of the past, present and future of CBSE. The authors analyze the achievements of the CBSE community using six factors, i.e. functionality (what is/does a software component?), interaction (how do we compose components?), quality (what results in a composition of components?), management (how to publish and retrieve existing components?), evolution and tools (what support is needed for application development?) and methodology (how to use CBSE, e.g. process and development models). The conclusion of the authors is that much has been achieved, but that especially the methodology, quality and management factors require more research before these can be considered solved. The discussion around the paper centred on the question whether progress actually had been made in the CBSE community or not! The second paper is this session, presented by Stefano de Panfilis, discusses the results of the CBSEnet network of excellence and open issues and concerns of CBSE. CBSEnet, as a project, has resulted in several valuable outcomes, including a classification model, a landscape document and a portal. The landscape document presents concepts and process issues, business concerns, product related topics, issues surrounding COTS components, domain specific concerns of component systems, e.g. business information systems, geographical information systems and embedded systems, and related paradigms, such as service-oriented computing and model driven engineering. The presenter stresses the importance of realizing that CBSE is not a goal in itself, but a means to improve productivity, quality and time-to-market.

2.4 Application of CBSE

The first paper in the final session, presented by Jasminka Matevska-Meyer, addressed the description of software architectures for supporting component deployment and dynamic reconfiguration. The authors define the requirements that an architecture description language (ADL) should satisfy, i.e. expressiveness for supporting dynamic configuration, runtime component behaviour description, timing constraints, runtime dependencies and composition features for describing subsystems. Subsequently, existing ADLs were evaluated against these requirements, leading to the conclusion that no ADL supported all requirements sufficiently. As an initial step towards a solution, the authors presented a meta-model and a reconfiguration manager. The authors identified a number of open issues, including a concrete ADL syntax for their approach, the need for a resource mapping view and a prototype implementation. The final paper was presented by Ivica Crnkovic and addressed the use of software components in the context of embedded systems. The presenter started by explaining the developments in the embedded industry, especially the automotive industry, where the demands on software have changed from basic to complex functions, causing multiple functions to share sensors, networks, processing nodes and actuators. The challenge is to offer an open but dependable platform for automotive applications, i.e. to balance flexibility and predictability. The presenter proceeded to distinguish between large and small embedded systems, focusing on the latter type. The main conclusion from the presentation was that component technology for embedded systems needs to focus on pre-deployment composition, the particularities of the run-time environment, fine-grained components, white-box reuse and source-code level component exchange. Concluding, component-technologies are becoming feasible for embedded systems, but require adaptation to the context of embedded systems.

3 Break-Out Sessions

In the afternoon the workshop participants were organized in break-out sessions addressing three specific topics, i.e. compositional reasoning on quality attributes, runtime re-configuration and re-deployment and, metaphorically, "The Big Picture". Each group had a nominated scribe who, as named in the subsection titles below, contributed the session summaries.

Compositional Reasoning on Quality Attributes by Ralf Reussner

This break-out group was concerned with the question how to predict or guarantee quality attributes of a component based system given the quality attributes of inner components and the architecture of their interconnection. This discussion started by listing and classifying the quality attributes the participants were interested in. A classification and a comprehensive overview on quality attributes is given in the ISO 9126 standard. Quality attributes of particular interest were: availability / reliability, performance (throughput, reaction time, response time) and resource consumption (memory, cache, etc.). We agreed to consider "scalability" a meta-quality attribute, as it describes changes of a quality attribute by varying resources, such as the response time of a clustered database application depending on a changing number of processors. Quality attributes can be classified according to the kind of statements one wants to derive from them. (a) predictions are of concern for quantitatively measured quality attributes, such as reliability or performance. (b) Guarantees are made by non-quantitative attributes, such as correctness, safety or security. (c) Monotonic or preservation statements are of concern if quality attributes can be measured in a discrete ordered scale (such as "low, medium, good"). As a special case, this scale includes all quantitatively measured attributes. Generally, quality attributes can be classified according their kind of metric: stochastic / deterministic and discrete / continuous. That is, the correlation among attributes depends at least also on the system's architecture (in the example, determining the presence or absence of redundancy). One factor of the complexity of reasoning on quality attributes are their interdependencies. A change of a system usually affects more than one quality attribute. For example, improving security by adding encrypted communication channels will most likely affect performance due to the computation costs for encryption and decryption. Remarkably, a pair of attributes is not always positively or negatively correlated. For example, adding replicated computing resources to increase performance may also increase reliability due to an increased fault-tolerance. However, exactly the inverse relationship also exists: by adding on a load-balancer and increasing network-traffic the replication of computing resources improves performance but lowers the reliability of the overall system. An important issue is to understand, that basically all quality attributes of a component are not constant properties of the component itself, but are highly dependent on the component context. Basically, the quality attributes depend on three groups of influence factors:

- The usage profile of the component: The way a component's services are used (the frequency, the parameters, etc) depend on the component's context. However, it is clear that the quality exhibited depend on this usage profile. For reliability, this is true by definition, as software reliability is defined as a function of the profile. However, the performance of a component service will also frequently depend significantly on the arguments provided for service parameters. (A download of 10 KB will be considerably faster than a download of 20 MB.)
- The use-relationship to other components: A component service most often makes use of several other external services. As a consequence, the quality of a service also depends on the qualities of external services. An highly unreliable external service will also influence the reliability of services building upon it.
- The deployed-on-relationship: Each component is deployed on resources, such as processing nodes, network connections or other resources. Obviously, the performance of the deployment hardware, the middleware, virtual machines, and the like influences reliability and performance of a component heavily.

Resource contention, leading to discontinuous quality behaviour, can happen via the use-relationship and the deployed-on-relationship. Any compositional reasoning has to take both relationships into account. The composition of quality attribute values can be either symmetric or asymmetric. Asymmetric composition of quality attribute values happens, if a component "inherits" the quality of another component. For example, an insecure component used within a secured environment (such as a sandbox) will become "secure". Symmetric composition of quality attribute occurs when the quality of an outer component can be computed by a commutative function of the inner components. The composition itself can be performed by many composition-operators. In principle, each architectural pattern or style defines a composition operator. However, one could ask for the minimal set of basic composition operators. Any composition operator should be expressible as a finite combination of basic composition operators. Some candidates for such basic composition operators can be (motivated by operators of process algebras): parallel, sequential, alternative, refinement, extension. Finally, we listed different calculi and their use for prediction models: Queuing Models (performance), Petri-Nets (concurrency), Finite State Machines (protocols), Markov-Models (reliability), Process Algebras (protocol), symbolic (logic) representations (memory, performance), timed-logics (time, liveness). The following participants participated in the break-out group: Steffen Becker, Thomas Cottenier, Viktoria Firius, Trevor Parsons, Ralf Reussner, Sibylle Schupp and Richard Torkar.

Runtime Re-configuration and Re-deployment by Jasminka Matevska-Meyer

The main concern of this break-out group was the definition of the scope of runtime re-configuration and re-deployment and the main issues to enable them.

First we compared the common goals of our projects and thus identified the topics of interest. This helped us to distinguish re-configuration from re-engineering approaches. Re-configuration is the process of (a) changing, (b) re-building and (c) re-deploying a system. It is thus a subset of re-engineering processes, including techniques to perform the necessary changes to the system while maintaining its consistency. The process of reconfiguration can include the identification of change requests (e.g. context dependencies), but it mainly focuses on processing those change requests. Hence, it only ensures the technical consistency of the system. Questions concerning the sense of the reconfiguration are to be answered by re-engineering approaches. Runtime re-configuration takes place at system runtime. The affected sub-system should be changed, re-build and re-deployed at system runtime. Runtime re-deployment should also include consistency checks of the system. It can be seen as an extension of hot deployment and dynamic reloading including consistency checks. We consider "rich" components (with information on services, connections, composition properties, hence more than mere executable code). The main problem here is the application at runtime. To successfully perform a runtime re-configuration we need much more information about the running system than its structural dependencies. We need an appropriate architecture description including (1) structural view (describing the hierarchical system structure and enabling its composition), (2) dynamic view (description of the system's runtime behaviour, particularly concerning the use dependencies among instances of components) and (3) resource mapping view. A well defined mapping between the views including relationships between description and implementation (reflection) is essential to identify and isolate the affected sub-system. Furthermore, we need techniques for checking consistency like monitoring, simulation, model checking etc. For identifying necessary changes we need an additional definition of environmental (contextual) dependencies. It becomes clear that enabling runtime re-configuration while maintaining the consistency of the system is a very complex problem. So, a legitimate question about the sense of those approaches arises. There are a variety of systems which are characterised through high availability (e.g. mission critical systems) and have to be changed at runtime. Performance aspects, like load balancing (horizontal and vertical) or resource driven adaptation (foraging) could also trigger a runtime reconfiguration. Finally, our group briefly proposed our concrete approaches concerning runtime re-configuration of component-based systems: (1) architecture-based (PIRMA) considering runtime dependency graphs, (2) resource-contract driven (CoDAMoS) as a context driven adaptation of mobile services and (3) Programming driven (XParts) treating dynamic distributable Java components. The following participants participated in the break-out group: Anders Gravdal, Jasminka Matevska-Meyer, Peter Rigole, Ulf Schreier and Bart de Win.

The Big Picture by Jean-Guy Schneider

In the following, we will summarize the results of the discussion of the breakout group on "the Big Picture". The main motivation of this breakout group was to step back from current trends and set component-based Software Engineering

(CBSE) into a bigger picture. In this context, we discussed various issues related to CBSE from a broader perspective and tried to come up with some directions towards which the discipline might be heading. As a starting point, we discussed the question where we could apply a component-based approach for software development. It was argued by one group member that CBSE is very time consuming, requires dual application development, and considerable knowledge in the respective application domain is essential. Furthermore, a group of experts is needed to apply CBSE successfully (”For most normal people CBSE is way too hard!”). In this context it was also mentioned that the existence of reusable components is not only a precondition for CBSE, but also a sign of a maturing domain. Hence, we concluded that CBSE can only be applied in mature domains where reusable components already exist. But what are the problems of finding reusable components and composing them in such a domain? It was stated that component repositories are not the issue, but skills in social dialogs are: talking to people to find out whether a component you need is available seems to be a much more promising approach than searching in a repository. Dedicated component repositories are not easy to set up and use as it is difficult to come up with suitable ontologies to classify components. Furthermore, ad-hoc search strategies are often more successful (”Why worry about UDDI, one can use Google”). What about composing components? It was argued that the real complexity is not in interfacing with existing components, but the real problem lies in the semantics of the data. Hence, we concluded that component models should focus (more) on standardizing component connections (”Standards are the key to setting up a component market at all”). So what is the situation with web services? One group member pointed out that web services are basically a trick by commercial vendors to make both developers and end users select their respective platforms, but, as such, are nothing else than components offering their services on the web. Hence, we came to the agreement that ”web service” is probably not such a good term and should be replaced by something like ”e-Service”. There was also some consensus that the people in the discipline are working too much on the solution domain (components, component models, composition environments etc.), but not enough in the problem domain (what kind of problems are suitable for CBSE). However, we acknowledged that CBSE has solved many technical issues - other areas have been addressed, but not yet solved. Does CBSE cover all aspects of Software Engineering? This turned out to be quite a difficult question to answer. After some discussion it was suggested that CBSE covers some of the traditional SE life-cycle activities, but it does a rather poor job when it comes down to requirements engineering. However, we failed to clearly identify which aspects are covered well and which ones poorly (or not at all). Hence, this is a topic where further investigations are needed. Finally, we addressed the question whether CBSE is just hype and will disappear in five years or is it here to stay. Not surprisingly (and probably due to a certain bias within the group), we agreed that CBSE is here to say and will not disappear in the near future. Unfortunately, we did not have the time to talk about many more issues that were raised during the lively discussion. Summing up what we were able to talk about, we came to the conclusions that

– it is the market, legal issues, and standardization, but not technology which will govern the success of CBSE,
– CBSE works well for product lines and in-house development, but not on the open market, and
– specialized people are needed to cope with the technology/complexity to develop applications using component-based Software Engineering approaches.

The following people participated in this breakout group (in alphabetical order): Ivica Crnkovic, Bernhard Groene, Stefano De Panfilis, Jean-Guy Schneider, David Thomas, and Hoang Truong.

4 Final Words

As organizers, we look back on yet another highly successful workshop on component-oriented programming. We are especially pleased with the constantly evolving range of topics addressed in the workshops, the enthusiasm of the attendees, the quality of the contributions and the continuing large attendance of more than 25 and often as many as 40 persons. We would like to thank all participants of and contributors to the ninth international workshop on component-oriented programming. In particular, we would like to thank the scribes of the break-out groups.

5 Accepted Papers

The full papers and additional information and material can be found on the workshop's Web site (`http://research.microsoft.com/cszypers/events/WCOP2004/`). This site also has the details for the Microsoft Research technical report that gathers the papers and this report. The following list of accepted papers is sorted by the name of the presenting author.

1. Ivica Crnkovic (Mlardalen University, Sweden). "Component-based approach for embedded systems"
2. Thomas Cottenier, Tzilla Elrad (Illinois IT, USA). "Validation of context-dependent aspect-oriented adaptations to components"
3. Jasminka Matevska-Meyer, Wilhelm Hasselbring and Ralf H. Reussner (University of Oldenburg, Germany). "Software architecture description supporting component deployment and system runtime reconfiguration"
4. Stefano De Panfilis (Engineering Ingegneria Informatica S.p.A., Italy) and Arne J. Berre (SINTEF, Norway). "Open issues and concerns on component-based software engineering"
5. Trevor Parsons (Dublin City U, Ireland) and John Murphy (University College Dublin, Ireland). "A framework for automatically detecting and assessing performance antipatterns in component based systems using run-time analysis"

6. Ralf H. Reussner, Viktoria Firus and Steffen Becker (University of Oldenburg, Germany). ”Parametric performance contracts for software components and their compositionality”
7. Jean-Guy Schneider and Jun Han (Swinburne University of Technology, Australia). ”Components - the past, the present, and the future”
8. Judith A. Stafford (Tufts University, USA) and John D. McGregor Clemson University, USA). ”Top-down analysis for bottom-up development”
9. Amir Zeid, Michael Messiha and Sami Youssef (American University Cairo, Egypt). ”Applicability of component-based development in high-performance systems”