04511 Abstracts Collection
Architecting Systems with Trustworthy Components
— Dagstuhl Seminar —

Ralf Reussner[1], Judith Stafford[2] and Clemens Szyperski[3]

[1] Univ. Oldenburg, DE
reussner@acm.org
[2] Tufts Univ., Medford MA, US
jas@cs.tufts.edu
[3] Microsoft Research, Redmond, US

Abstract. From 12.12.04 to 17.12.04, the Dagstuhl Seminar 04511 "Architecting Systems with Trustworthy Components" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Component frameworks, quality prediction, performance prediction, limits and assumptions of predicition methods, adaptation, archtiectural mismatches, patterns, components errors blame analysis assignment, protocols, interoperability, component specification

04511 Breakout Group – Component Frameworks

Terminology The breakout group attempted to frame the term "component framework" by agreeing on definition fragments of the terms component and framework, for the purposes of this discussion. Components: units characterized by relatively late binding (i.e., not by a developer), retention of identity post deployment, and explicit context dependencies.

Frameworks: starting from the observation that random components, even if 100% perfect, are not going to work with each other and that an adapter is a symptom of something being wrong, there is a need to produce components that fit certain standards, interfaces, abstractions, and ontology. These are, what a component framework provides.

Dagstuhl Seminar Proceedings 04511
Architecting Systems with Trustworthy Components
http://drops.dagstuhl.de/opus/volltexte/2006/473

Examples:

In the vague context of the terminology frame, a wide range of examples helps make the component framework idea more concrete. Such examples include:

- Extensible Editors like Emacs
- Application Servers like JBoss with different component categories, like beans and interceptors
- Extensible IDEs, like Eclipse or Visual Studio, with plug-ins (or add-ins) as components
- Operating systems with applications, device drivers, kernel modules, etc. as components
- OS kernel architecture, with micro and nano kernels as the result of creating minimal component frameworks
- Layered stacks, such as networking stacks
- Interception component frameworks, as found in both application servers and networking stacks, to guard against propagation of errors Why Component Frameworks
- As a specific case of applying a separation of concerns discipline, component frameworks can be used to factor services and aspects out of individual components.
- Assuming a component framework that itself is significantly more reliable than any of the components, the component framework can help to assign blame to failing components.
- Component frameworks can impose common design decisions and thus enforce constraints. "We have to make some things theoretically impossible, to make (other!) things possible in practice".
- The presence of a component framework helps establishing a higher basis of guarantees under composition and varied components.
- By factoring out the hardest / most guarding substrate of all solutions build using a component framework, higher levels of robustness can be achieved.

Trade offs

- The traditional tradeoff of mechanisms versus policies applies and the end-to-end argument suggests that component frameworks should not embody policies that go beyond their scope.
- The trade-off between the degree of framework completeness and constraint enforcement on the one hand and flexibility and potential on the other characterizes the nature of component frameworks.
- Given the difficult tradeoffs, the expectations of a high quality bar, and the likelihood to embody a critical point of failure - when comparing component frameworks and components - highly specialized people are needed.
- As usual with broad and deep abstractions, it is necessary to conduct at least three different case studies to validate a framework before it should be released to a broader audience. (Instead of three some other larger prime number may be chosen. The patterns community finds seven an appealing number.)

- A component framework helps composing components designed with that framework in mind. However, how are frameworks composed with each other? One appealing approach is to think of component frameworks together with the components they carry as large components at the next larger level of granularity. Then, a hierarchy of component frameworks can be used, applying the component framework concept recursively.

Concerns

- While the scalability of fundamental component-framework concepts is desirable, it is unlikely that any specific component framework can meaningfully scale across a very wide range.
- Assemblies build as compositions based on component frameworks may themselves experience scalability or performance problems caused by the component frameworks presence.
- Given the requirement to build a specific solution, a component framework should not get in the way. However, this is in contrast to the need to restrict composition on top of a component framework - just to enable the main advantages of using a component framework in the first place.

Semantics

What? How? Articulation of domain Component framework Quality attributes Design patterns, architectural styles

Future research

Domain-specific languages can be used both to capture and to exploit the semantics of a particular domain. One perspective is that component frameworks are domain-specific languages without surface syntax. It should be fruitful to research the potential of cross-pollination of the two disciplines of domain-specific languages and component frameworks.

Another area of future work would be investigations into programming-language support for the use and, possibly, the construction of component frameworks. (The latter task is presumably far less common, making language support less compelling.)

Participants:Shriram Krishnamurthi, Stig Larsson, Judith Stafford, Alexander Stuckenholz, Clemens Szyperski, Rob van Ommering, Wolfgang Weck

Keywords: Component Frameworks

04511 Breakout Group – Performance Prediction of Component-Based Systems

Predicting the performance of component-based systems has been in the focus of research for quite some time now. Consequently, a variety of methods that use different mathematical models evolved until now.

Most of methods, however, make specific assumptions that increases or decreases their applicability in certain domains of systems. To classify, which of the methods is appropriate in a specific context (i.e. for a certain class of systems), we provide a comparison of methods for performance prediction with respect to their (implicit) assumptions.

Therefore, we identified a variety of influences which have eventually to be taken into consideration during the predicition of the performance of a particular system. In addition, we classified the existing methods according to their underlying mathematical model. Building upon these two abstractions, we compared the applicability of exisiting methods according to the identified influences and deduced a variety of pratical recommendations.

Keywords: Quality Prediction, Performance Prediction, Limits and Assumptions of Predicition Methods

Joint work of: Becker, Steffen; Firus, Viktoria; Gorton, Ian; Grunske, Lars; Mirandola, Raffaela; Overhage, Sven

04511 Breakout Group – Adaptation: Towards an Engineering Approach for Component Adaptation

The adaptation of components is an inherent task of component-based software engineering (cf. BG Adaptation I). Therefore, the formation of an engineering approach to component adaptation is necessary to support the development of trustworthy systems that reliably meet their requirements. The main goals of such an engineering approach are to provide a theoretically substantiated solution to adapt components and simultaneously support the reasoning about any changes in the quality attributes that arise from the adaptation. Moreover, the envisaged solution should allow the creation of adapters by hand as well as the development of adapter generators that automate the adaptation task.

As a first step towards an engineering approach, a classification of interoperability problems that may cause a need for adaptation has been developed and different classes of interoperability problems have been identified. For each of the identified classes, a variety of patterns has been listed. These patterns can be used to overcome a particular interoperability problem. In so doing, we distinguished between basic building blocks of patterns, generic patterns, and specific patterns (which often build upon generic pattern to provide solutions for a particular problem domain).

In order to create an adapter, a certain interoperability problem first has to be detected and then to be eliminated. Both of these tasks require specific metadata which has to be provided by the respective component manufacturers (e.g. in form of a component specification). In order to provide a recommendation on the metadata that has to be provided, we plan to deliver a classification that describes which kind of metadata is required to detect and eliminate a certain class of interoperability problems. In addition, we plan to investigate the effects that a certain kind of adaptation has on the quality attributes of the system.

Keywords: Adaptation, Archtiectural Mismatches, Patterns

Joint work of: Becker, Steffen; Brogi, Antonio; Firus, Viktoria; Goos, Gerhard; Gorton, Ian; Mirandola, Raffaela; Overhage, Sven; Romanovsky, Alexander; Tivoli, Massimo

04511 Breakout Group – Blame Assignment

Blame assignment is difficult enough in theory: sometimes blame cannot be assigned to any proper subset of components, yet it isn't necessarily the fault of the integrator either. It may sometimes be better to blame the wrong component – just to initiate the investigation – than to cautiously blame none at all! Indeed, the economics of the situation potentially even moves the problem out of the technical arena: for instance, user perceptions may force major corporations to shoulder blame, even when a fault clearly lies with a supplier.

We must move towards architect components to enable blame assignment. This seems like a good role for a component framework, as the enforcer of such architectures. As blame problems grow in significance, integrators and users may come to expect components certified by independent bodies. In turn, developers may come to view the possibility of being blamed as a risk against which they can purchase insurance.

Keywords: Components errors blame analysis assignment

Joint work of: Brederecke, Jan; Larsson, Stig; Krishnamurthi, Shriram; Stuckenholz, Alexander; Sulzmann, Christian; van Ommering, Rob; Szyperski, Clemens; Weck, Wolfgang

04511 Brakout Group – Classification of System Engineering Approaches

In this breakout group, we discussed of different methods that can be used to develop a system and to define its architecture. In order to be able to classify them, we proposed framework to first introduce some classification criteria then show on a diagram the studied method. We applied this framework introducing five criteria classifying four methods.

Keywords: Developement methods, comparison criteria

Joint work of: Bunse, Christian; Freiling, Felix; Levy, Nicole

04511 Breakout Group – Adaptation - Coming to Terms with Adaptation

The breakout group discussed the answers to the questions

- why adaptation is needed
- when adaptation is performed
- what kind of adaptations can be distinguished
- and finally how adaptation can be done on a systematical basis.

The first question was answered by pointing to the WebService and Service oriented architectures where heterogenious services are quite common. Additionally, we came up with the integration of legacy systems, where it is often impossible to change the interface of the old system so it has to be adapted to fit into new contexts. Lastly we realized that even with standardized interfaces and frameworks there is evolution in the system due to changing requirements so that adapters have to be provided to not get into a maintainance problem.

To answer the question when adaptation is done we classified into the categories of design-time adaptation and run-time or behavioural adaptation. Design-time adaptation is being done when a requires interface of a component is mismatching with the provides interface of a component which are suposed to work together. Note that it is depending on the interface model what the actual meaning is. Interface models capturing protocol or QoS information are able to detect more mismatches than ones which only model signatures.

Run-time adaptation takes place when the system is actually running and is often done by reconfiguring the component instances of the system. This might be done on a contextual basis (i.e., moving a mobile device into a different context) or on the basis of policies telling the system based on objectives, how to reconfigure under certain conditions.

A classification of the different types of adaptations was discussed by the participants. We came up with two basic classes: one for functional adaptation problems and another one for non-functional interoperability problems. Somewhat orthogonal to that kind of classification is the second dimension we discussed. Along that dimension we distinguished syntactial problems, protocol and semantical problems. Syntactical problems include technical issues of bridging different plattforms and signature matching. Protocol problems deal with adding or deleting messages in order to ensure a certain protocol flow and is most often done with statefull adapters. Semantical problems arise from a different understanding of the underlying domain and are often hard to detect and hard to bridge.

To come up with sugestions on the how a closer look on desing patterns and architectural styles used in adapation was proposed for a second breakout session. Additionally we planed to think about the actual metadata needed for a respective generator tool to construct the adapter.

Keywords: Component Adaptation, Adaptation Times, Adaptation Techniques

04511 Breakout Group – Unified Prediction Model

Under a unified prediction model we understand prediction models for two or several quality attributes of software which base on the same abstraction of the software. An example is a queuing model used to predict the timing and the reliability of the software. The interest in such models is raised (a) by practical concerns: as one has to analyse more than one quality attribute for software, it is beneficial the build the prediction on one single model than to spend effort in defining several models. (b) as several quality attributes depend on each other, one often cannot defined a metric for a quality attribute without considering other quality attributes (e.g., relaibility prediction models may need assumptions on the timing behaviour of the software). A different form of interdependencies arise due to antagonistic relations between quality attributes. For many software designs, the optimisation of one quality attributes will result in a loss of another quality attribute. Due to these dependencies, the consideration of combined quality attributes is often inevitable for an evaluation of a software architecture.

Keywords: Quality prediction, compositional reasoning, abstract models for software

Joint work of: Becker, Steffen; Crnkovic, Ivica; Jezequel, Jean-Marc; Firus, Viktoria; Gorton, Ian; Küster-Filipe, Juliana; Mirandola, Raffaela; Overhage, Sven; Reussner, Ralf; Romanovsky, Alexander; Salzmann, Chris

04511 – Semantics of Specification Languages for Component-Based Systems

This session discussed the requirements for specification languages that are specifically targeted to component-based systems. The summary is presented as five key points. Also included are several pages of raw notes; these include reference to some topics that were important and interesting, but for some reason did not surface in the "top five."

Joint work of: Brogi, Antonio; Fisler, Kathy; Goos, Gerhard; Jahnke, Jens; Levy, Nicole; Schmidt, Heinz; Wallnau, Kurt

04511 – Final Report on Limits of Predictability

We summarize the results of our discussion as a collection of not-yet written papers and their abstracts.

Joint work of: Becker, Steffen; Fisler, Kathi; Hofmeister, Christine; Jahnke, Jens-Holger; Kniesel, Günter; Krishnamurthi, Shriram; Levy, Nicole; Reussner, Ralf; Schneider, Jürgen; Wallnau, Kurt

04511 – First day notes and report of breakout group on limits of predictability

These are the notes taken on the first meeting of the breakout group on limits of predictability on Wednesday together with the slides which presented the notes on Thursday morning.

Joint work of: Freiling, Felix

04511 Breakout Group Summary – Interaction Protocols

Interaction protocols represent the abstract behaviour of a software component by abstractions of traces. In our community, the term "protocol" means a totoally or partially ordered set of events/calls. A component's behavour can be described by several different of these sets: (a) the provides protocol (i.e., set of valid sequences of calls to provided services), (b) the requires-protocol (i.e., the set of possibly emmited sequences of calls to external services) and (c) a protocol specifying the relationship between provided- and requires-protocol, i.e., for each provided service one specifies the set of possibly emmitted sequences of calls to external services.

Applications of interaction protocols include (a) eliciting abstract observational semantics of components, (b) exposing interaction inconsistencies / errors early (interopability checks), and (c) exposing abstract observable 8216;safe 8217; states for atomic dynamic updates.

Keywords: Protocols, interoperability, component specification

Joint work of: Sunbül, Asuman; Hofmeister, Christine; Küster-Filipe, Juliana; Glesner, Sabine; Krämer, Bernd; Goos, Gerhard; Plasil, Frantisek; Crnkovic, Ivica; Reussner, Ralf; Schmidt, Heinz W.

Staged Architectures

Uwe Assmann (TU Dresden, D)

We present the concept of a staged architecture for software systems and active documents. Such an architecture consists of several computation stages that generate each other. Every stage employs a specific component model as well as a software architecture. With a staged architecture, very complex systems can be described very concisely. Also variant configuration is very simple. We give an overview to the connection to model-driven architecture and web engineering.

Keywords: Architecture, staged programming

Enhancing the Trustworthiness of Component-Based Systems through Built in Contract Testing

Colin Atkinson (Universität Mannheim, D)

Assembling new software systems from prefabricated components is an attractive alternative to traditional software engineering practices which promises to increase reuse and reduce development costs. However, these benefits will only occur if separately developed components can be made to work effectively together with reasonable effort. Lengthy and costly in-situ verification and acceptance testing directly undermines the benefits of independent component fabrication and late system integration. Building self testing capabilities into components is techniques for reducing manual system verification effort by equipping components with the ability to automatically check their execution environments at runtime. When deployed in new systems, built-in test (BIT) components check the contract-compliance of their server components, including the run-time system, and thus automatically verify their ability to fulfill their own obligations. Enhancing traditional component-based development methods with built-in contract testing in this way reduces the costs associated with component assembly, and thus makes the "plug-and-play" vision of component-based development closer to practical reality.

Keywords: Built in Test Components

The Impact of Software Component Adaptation on Quality of Service

Steffen Becker (Universität Oldenburg, D)

Software component adaptation is important when it comes to bridging mismatching components that need to interoperate. The possible scenarios include legacy integration and system evolvment where interfaces finally are declared depricated for reasons of maintainability.

When bridging functional mismatches the Quality of Service characteristics of the adapted component are changed by the adapter. This is likly to be unwanted if the component was selected to build a trustworthy system. Additionally changing the Quality of Service by the adapter might be on purpose when extra-functional adaptation is considered. Examples include replication or failure-retry mechanisms. Especially for the class of adapters that can be constructed automatically or semi-automatically by the support of an appropriate suite of adapter generators it is worth investigating the impact on QoS in advance so that building predictable compositions with adapters becomes feasable.

Keywords: Adapter generation, component adaptation, component composition

Modular Requirements Against Feature Interaction Problems

Jan Bredereke (Universität Bremen, D)

Structuring requirements into information-hiding modules helps against feature interaction problems. Feature-oriented descriptions are popular, for example, in telephone switching. But composing many features often leads to undesired behaviour. Our requirements modules group those properties together that are likely to change together. This reduces dependencies among requirements modules.

Dependencies should be documented explicitly. This helps to detect remaining interaction problems. For the formalism Z, we show how we can structure requirements into modules and document dependencies.

We propose a small extension for Z that allows hierarchical grouping, and interfaces. Interfaces restrict the access to changing parts of the requirements.

Keywords: Feature orientation, feature interaction problems, maintenance, information-hiding modules, formal requirements, Z

Adapting Components with Mismatching Behaviour

Antonio Brogi (Università di Pisa, I)

I shall present a formal methodology that we have developed to adapt components presenting mismatching interaction behaviour. The approach consists of three main ingredients: (1) extend component interfaces with a description of the component behaviour, (2) express (partial) adaptor specifications as simple correspondences among actions of two components, (3) automatically generate an adaptor component, given its partial specification and the interfaces of two components. I shall also briefly mention other developments of the methodology - such as forms of "soft" adaptation, adaptation trading, and the use of behavioural types for software adaptation - and our recent activity oriented towards (Web) service discovery and aggregation.

UML-based Development of Embedded Systems - Improving Component Quality

Christian Bunse (FhG IESE - Kaiserslautern, D)

Model-driven development, using the UML, has become the most dominant development paradigm, particularly in business and web application engineering, due to their many advantages over traditional procedural approaches. However, Model-driven and UML-based development methods are still inferior to conventional software development approaches when it comes to embedded and real-time system development. Most such methods provide only weak systematic and methodological support for system development in the embedded domain. The most fundamental problems in this domain stem from the fact that individual techniques for embedded system development only acknowledge and address the particularities of object and component technologies insufficiently, and more importantly, that individual technologies are mostly treated in isolation. Important aspects are the heterogeneity of embedded systems, containing both, hardware and software components, and missing methodological support concerning the modeling and verification/validation of non-functional properties. This is the goal of the MARMOT approach (www.marmot-project.de) currently under development.

Keywords: UML, Embedded Systems, CBSD, Verification & Validation

Component-based Approach for Embedded Systems

Ivica Crnkovic (Mälardalen University - Vasterås, S)

Although attractive for many reasons, such as reusability, time-to market, component-based approach has not been widely adopted in domains of embedded systems. The experience has shown that existing technologies cannot be directly used for development of embedded systems. The reason is inability of these technologies to address the main concerns of embedded systems: real-time properties, resource consumption (power, memory, CPU, etc.), and dependability. On the other hand an increasing understanding of principles of component-based approach makes it possible to utilize these principles in implementation of different component-based models, more appropriate for embedded systems. There are some proprietary component models that have been successfully used in development of embedded systems. The aim of this presentation is to discuss the question:

Which concerns related to the embedded systems should be "embedded" into component models? Further, how can we achieve the basic principles of component-based approach, such as substitutability, reusability, expandability and composability for these concerns?

Verifying Compilers

Gerhard Goos (Universität Karlsruhe, D)

This is a short report about the finished DFG project Verifix of the universities Karlsruhe, Kiel and Ulm. We quote the meaning of "correctness" of a compiler and hint to program checking as a technique for verifying the result of a compilation rather than the compiler itself.

Performance Prediction for EJB Applications

Ian Gorton (National ICT Australia - Eveleigh, AU)

A challenging software engineering problem is the design and implementation of component-based (CB) applications that can meet specified performance requirements. Our PPCB approach has been developed to facilitate performance prediction of CB applications built using black-box component infrastructures such as J2EE. Such deployment scenarios are problematic for traditional performance modeling approaches, which typically focus on modeling application component performance and neglect the complex influence of the specific component technology that hosts the application. In this paper, an overview of the PPCB modeling approach is given. Example results from predicting the performance of a J2EE application are presented. These results are then statistically analyzed to quantify the uncertainty in the predicted results. The contribution of the paper is the presentation of concrete measures of the confidence an architect can have in the performance predictions produced by the PPCB.

Joint work of: Gorton, Ian; Liu, Jenny

Automatic Application of Behaviour-Preserving Transformations to Improve Non-Functional Properties of an Architecture Specification

Lars Grunske (The University of Queensland, AU)

For safety-critical systems it is necessary to make sure that the non-functional properties imposed by a system architecture meet safety requirements as early as possible in the system development lifecycle. The idea is to use quality-improving architectural transformations in case the non-functional properties do not fulfil their requirements. Selection and application of appropriate architectural transformations is a time-consuming and error prone task, but many aspects of it can be automated. In this talk, an approach is presented that uses hypergraph grammars to formally specify such transformations.

Safety, Liveness, and Information Flow: A Framework for Designing Dependable Distributed Systems

Felix C. Freiling (RWTH Aachen, D)

The notion of dependability was introduced to cover a range of critical system attributes, namely: availability, reliability, safety, and security. We present a formal framework to precisely talk and reason about dependable systems. The framework is based on three distinct classes of (system specification) properties

we call "safety", "liveness" and "information flow". We discuss several examples of dependable systems within this framework and argue that these classes are sufficient to model the functional requirements of dependable systems satisfying to high degrees both safety and security attributes.

Keywords: Fault-tolerance, security, non-interference, possibilistic information flow, framework, dependability, abstraction, model, verification, design

Avoiding Interface Failure Between Stubborn Components (a.k.a. Researchers)

Shriram Krishnamurthi (Brown Univ. - Providence, USA)

Components, composition, interfaces: these are the stuff of motherhood and apple pie. Yet each of these terms is fraught with ambiguous interpretations; we can immediately imagine at least a dozen different meanings of the phrase "Interfaces for Component Composition". Failure to keep these meanings separate leads to cross-talk, confusion and, eventually, wasted effort.

Joint work of: Krishnamurthi, Shriram; Fisler, Kathi

Imposing Synchronization Constraints on Interaction Protocols of CORBA Objects

Bernd Krämer (FernUniversität Hagen, D)

In earlier works we developed a declarative approach to specify synchronization constraints for interaction protocol of distributed objects that may exhibit concurrent behavior [Krämer 98]. The approach is based on partially ordered sets of events, called processes. These processes model the abstract behavior of an object in terms of events denoting the beginning and ending of individual operation executions. A counting function # allows us to compute how often some operation has been executed in a given process. This function provides the basis to define standard synchronization constraints, such as mutual exclusion, self-exclusion (an operation can be executed by at most one thread at a time), precedence, or synchronic distance (an operation must not occur more than n times more often than another operation in a given process), in term of logic predicates.

This specification mechanism has then been applied to CORBA applications by including synchronization predicates as annotations invisible to standard IDL compilers [Jacobsen and Krämer 00a]. The synchronization constraints were separately compiled into code implementing corresponding sanity checks. In a further step we searched for design alternatives that exploit different features of the CORBA standard to seamlessly integrate synthesized synchronization code with manual implementations of the object's functionality. These solutions were

developed into a suite of design patterns for implementing IDL extensions that co-exist with standard IDL compilers. Prototype implementations of the proposed design patterns served to empirically investigate their advantages and disadvantages.

References

[Jacobsen and Krämer 00a] H.-A. Jacobsen and B.J. Krämer: Modeling Interface Definition Language Extensions. In: 37th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS-37), 20-23 November 2000, Sydney, Australia.

[Jacobsen and Krämer 00b] H.-A. Jacobsen and B.J. Krämer: Design Patterns for Synchronization Adapters of CORBA Objects, L'Objet, 6(1), pp. 57-82, 2000

[Krämer 98] B.J. Krämer: Synchronization Constraints in Object Interfaces, in B.J. Krämer and M.P. Papazoglou and H.-W. Schmidt (Eds.) Information Systems Interoperability, chapter 5. Research Studies Press (Wiley & Sons), 1998.

Keywords: Obejct synchronization, concurrency, interaction protocol

A modelling and reasoning framework for dependability

Juliana Küster Filipe (University of Birmingham, GB)

For complex distributed critical systems properties like safety, reliability and security aspects are clearly of great concern. Typical examples of such systems include systems in healthcare environments, air traffic control, nuclear power plants, and industrial automation. When developing critical systems it is important to have techniques to analyse and verify dependability requirements as well as reveal potential trade-offs. Formal methods can underpin these techniques, but currently there is no combined powerful formal framework that allows this.

In practical terms, besides having theoretical foundations available to reason about dependability, it is necessary to offer modelling notations to software engineers which integrate with standard methodologies and tools.

I am particularly interested in:

1. Developing a uniform formal logic-based framework for reasoning about dependability in distributed critical systems.
2. Extending existing modelling notation for capturing dependability requirements at different levels of detail.
3. Developing prototype tools which build an environment for dependability evaluation and formal verification.

Product integration with software components

Stig Larsson (ABB - Västerås, S)

So far research focus for component-based engineering has been on technical issues. There is a clear need to further explore the processes for architecting system based on components. Both top-down and bottom-up methods are needed. The technologies that can support these methods must also take quality attributes into account. It is also important to build the family of methods based on the knowledge that there are different resons to architect with components; e.g. re-use or simplifying the engineering of systems. Especially the process of integrating products will benefit from a well-designed family of concepts for components and methods for the engineering of systems.

Architecture development with patterns

Nicole Levy (University of Versailles, F)

The design of the software architecture of a system is a challenging task. Architectural patterns are recognized as useful to provide architectural solutions. But their selection among a pattern library and their application remains difficult. This is due on to the informal description of both the problem to be solved and its nonfunctional requirements.

We propose a quality-based approach to architectural design focusing on the problem to be solved. The problem is described in terms of its functional and nonfunctional requirements. The architecture is refined by application of architectural patterns. An architectural pattern is defined as a problem-solution pair. Both the problem and the solution parts contain a functional and a nonfunctional description.

Keywords: Software architecture, architectural patterns, functional and nonfunctional requirements

Predicting the Quality of Service of Component-based Systems

Raffaela Mirandola (Università di Roma II, I)

Component Based Software Engineering (CBSE) is today the emerging paradigm for the development of large complex systems. By maximizing the re-use of separately developed generic components, it promises to yield cheaper and higher quality assembled systems. The basic understood principle (or actually aspiration) is that the individual components are released once and for all with documented properties and that the properties then resulting for an assembled system can be obtained from these in compositional way. While this principle/aspiration has been actively pursued for the system functional properties since the advent of CBSE, it is only recently that equal emphasis is being devoted to the as important non-functional aspects or Quality of Service (QoS), such as reliability, security and performance. To facilitate QoS analysis since the design stage, automatic prediction tools should be devised, that predict some overall quality attribute of the application, without requiring extensive knowledge of analysis methodologies to the application designer In this talk we focus on the evaluation of performance properties (like response time, throughput, etc.) and we present a journey through different methodologies (and related tools) for the specification and analysis of performance related properties of components and assemblies of components.

Joint work of: Mirandola, Raffaela; Bertolino, Antonia; Grassi, Vincenzo

Unified Specification of Components: Towards a Standardized Framework to Support Component Development, Discovery, and Composition

Sven Overhage (Universität Augsburg, D)

Many methods and tools that support component-based development require information about components, e.g. to reason about system properties, perform compatibility checks, or to achieve component adaptation. In order to support all these methods, the UnSCom (Unified Specification of Components) framework has been introduced as a single source of information. It focuses on providing the information necessary to facilitate component development, discovery, and composition. To be applicable in all these ?elds, the UnSCom framework ties together a mix of different speci?cation aspects and unifies the speci?cation of components using a single, coherent approach. This approach is based on the notion of design by contract which has been extended to component-based software engineering by introducing service and composition contracts. The UnSCom framework supports the specification of composition contracts, which describe the required and provided interfaces of components on various contract levels. They are thematically grouped into colored pages: blue pages describe the required and provided functionality, green pages comprise the architectural design of the required and provided interfaces, and grey pages describe the required and provided quality of components.

Keywords: Component specification, design by contract

Session Notes: Semantics of Specification Languages for Component-Based Systems

Ralf Reussner (Universität Oldenburg, D)

Session notes covering top five results and also several pages of raw notes.

Joint work of: Brogi, Antonio; Fisler, Kathy; Goos, Gerhard; Jahnke, Jens; Levy, Nicole; Schmidt, Heinz; Wallnau, Kurt

Dependability-explicit Computing: applications in e-Science and Virtual Organisations

Alexander Romanovsky (University of Newcastle, GB)

Providing a predictable level of dependability is a challenge for applications which choreograph services from many different providers. Applications routinely fail because a component service fails, yet the designers of applications have, at best, limited information about component service dependability. This limits their ability to make informed decisions about when it is cost-effective to use a service or to employ potentially expensive fault containment or tolerance techniques such as redundancy. We consider ways to improve support for the publication and exploitation of dependability metadata for services by developing publication methods and ontologies to support shared metadata definitions. Two diverse examples of metadata are considered: service availability information and descriptions of service failure modes. The availability work is particularly relevant to bioinformatics, while work on failure modes is explored in the context of virtual organisations with long-term interactions.

Joint work of: Romanovsky, Alexander; Fitzgerald, John

Reasoning about component architectures and their extra-functional properties in real-time systems

Heinz W. Schmidt (Monash University, AU)

This talk will briefly summarise our recent work which falls squarely in the intersection of component-based software architecture, component protocol specification and extra-functional property modeling. In our formalism for modeling architecture definitions we capture dependency networks between components and connectors and associate formal protocol types with components and connectors. Our formalism, dependent finite state machines or DFSMs, has been studied in connection with prediction of worst-case time in distributed control systems and reliablity in distributed transaction systems - for instance web services. DFSMs associate finite state automata with connectors and finite state translators with components to enable reasoning *through components* from provided to required interfaces and following connectors. Subtyping in DFSMs leads to a flexible notion of conformance not only for component connection, but also component replacement and architecture reconfiguration.

Reasoning across architectural dependency networks connecting components is key to modeling adaptation, replacement and extra-functional properties of component-based systems. For example, if we want to understand the reliability of a service provided by a component C we have to factor in the reliability of the external component services called by C via its required interfaces. The reliability of the service provided may depend significantly on the reliability of an called component in the deployment environment and, more importantly, it may differ significantly depending on the expected call frequencies. For example, if C calls a low-reliability component L frequently in its inner loops while only calling another high-reliability external component H once during initialisation, C's reliability will tend towards the lower reliability of L.

As the basic formalism has been published (and in fact I gave a presentation at another Dagstuhl seminar in 2002), I will mainly summarise more recent results. In particular, I plan (a) to provide an update on component-based time prediction models in an industrial cooperation with ABB, (b) point to the results of a recent research student project on connecting our model with resource cost predictions using Markov chains, and (c) mention work in progress, which extends DFSMs to so-called Depedent Context-Free Translators (DCFTs), which lift several of the limiting architectural restrictions of DFSMs, without giving up efficient prediction or the spirit of underpinning extra-functional property models by architectural dependency networks.

Keywords: Component-based software engineering, software architecture, extra-functional properties, real-time systems, reliability, soft real-time, worst-case execution time

Manage High Availability with Concepts of Automated Operations' AO

Jürgen Schneider (IBM - Böblingen, D)

This reflects some conceptual work within IBM to support business resilient IT services. We are introducing into some automation concepts and take this model further to support HA clusters and distributed heterogeneous distributed applications.

Keywords: Availability Automation Recovery Disaster

Compatible component upgrades through smart component swapping

Alexander Stuckenholz (FernUniversität Hagen, D)

Emerging component-based software development architectures promise better re-use of software components, greater flexibility, scalability and higher quality of services. But like any other piece of software too, software components are hardly perfect, when being created. Problems and bugs have to be fixed and new features need to be added.

This paper will give an introduction to the problem of component evolution and the syntactical incompatibilities which result during necessary multi component upgrades. The authors present an approach for the detection of such incompatibilities between multiple generations of component versions and present a solution for automated versioning of relevant component facets. The main concern of the paper will be the automated reconfiguration of component based software systems by intelligent swapping of component versions to find conflict free system states.

Keywords: Component evolution, component upgrades, automated versioning, automated reconfiguration

Joint work of: Stuckenholz, Alexander; Zwintzscher, Olaf

A survey of composition techniques for Wireless sensor Networks

Asuman Sünbül (SAP Research Labs - Palo Alto, USA)

When we look at the current technology, we will observe a strong divergence away from traditional computing. While the emphasis of traditional computing involves a certain kind of more or less “fixed” networks, its more future oriented counterpart is a collection of mostly mobile, tiny devices which may be equipped with sensors, actuators and radio. Typical examples are e.g. swarm type systems, ubiquitous computing, biologically inspired or network embedded systems.

The application design for those environments is considerably different than their traditional counterparts. There is no centralized processor to guide the global strategy towards good solutions. The system has to be capable of accomplishing difficult tasks in dynamic and varied environments without any external guidance or control and with no central coordination. Further, they vary in terms of network connectivity, available power, available sensors and reliability of sensor data.

This talk will give an insight of ongoing work within sensor networks and will focus on constituting a natural model particularly suited for distributed problem solving, which aims at transforming an abstract, problem-oriented model into executable code.

Synthesis of "correct" adaptors for protocol enhancement in component-based systems

Massimo Tivoli (Univ. degli Studi di L'Aquila, I)

Adaptation of software components is an important issue in Component Based Software Engineering (CBSE). Building a system from reusable or Commercial-Of-The-Shelf (COTS) components introduces a set of problems, mainly related to compatibility and communication aspects. On one hand, components may have incompatible interaction behavior.

This might require to restrict the systemŠs behavior to a subset of safe behaviors. On the other hand, it might be necessary to enhance the current communication protocol. This might require to augment the systemŠs behavior to introduce more sophisticated interactions among components. We address these problems by enhancing our architectural approach which allows for detection and recovery of incompatible interactions by synthesizing a suitable coordinator. Taking into account the specification of the system to be assembled and the specification of the protocol enhancements, our tool (called SYNTHESIS) automatically derives, in a compositional way, the glue code for the set of components.

The synthesized glue code implements a software coordinator which avoids incompatible interactions and provides a protocol-enhanced version of the composed system. By using an assume-guarantee technique, we are able to check, in a compositional way, if the protocol enhancement is consistent with respect to the restrictions applied to assure the specified safe behaviors.

Keywords: Component Based Software Engineering, Component Adaptation, Adaptors Synthesis, Component Assembly, Protocol Transformation, Protocol Enhancement

Joint work of: Autili, Marco; Inverardi, Paola; Tivoli, Massimo; Garlan, David

Predictable Performance of Control Systems Under 3rd Party Extension

Kurt Wallnau (CMU - Pittsburgh, USA)

An open control system (OCS) has hard real-time and other safety requirements, but will be extended with new software functions by value-added integrators. In this talk I report on work the SEI has done in an industrial robotics OCS. The problem is to ensure that 3rd-party extensions that have stochastic (aperiodic) execution time do not interfere with hard real-time (periodic) OCS control tasks, and to provide optimal, and predictable, processing resources to these extensions. Our solution is to use a special kind of software (component) container called a sporadic server. A sporadic server implements a priority-based coordination scheme that is guaranteed to have bounded invasiveness on periodic control tasks, and to preserve the rate monotonic analysis predictability of OCS deadlines. In this talk I describe the sporadic server coordination model, and outline a novel application of queuing theory for predicting the average latency of stochastic tasks executing within sporadic server containers. I then situate this work in the broader context of predictable assembly from trusted (certifiable) software components.

10 Minute Abstract

Kurt Wallnau (CMU - Pittsburgh, USA)

What I wish to say in 10 minutes: My belief is that design specifications should have precise semantics that are distinct from, but related to, the mathematically-based semantics of programming language (e.g., structural operational and denotational semantics).

I say "distinct from" because the issues of concern for design specifications, such as performance, are difficult to formalize and to provide automated analysis for using traditional models of computation (e.g., transition systems and lambda calculus). I say "related to" because system-behavior is of course emergent from program-level behavior. As with programming languages, certain syntactic disciplines lead to cleaner semantics and, in some cases, stronger assertions that can be made about well-formed programs; for example, type systems and assertions about maintenance of representation invariants. Analogous syntactic disciplines exist for design-level specifications. Previous work on "styles" pointed in this direction, but was not strongly based in formal semantics; it is not clear whether styles in general yield meaningful assertions about system behavior.

My interest in a particular idiomatic interpretation of component technology which might be called "programmable containers." Containers can be used to develop design-level abstractions that impose useful syntactic discipline. In particular, the idiom of programmable containers provides a mechanism for separating the coordinational aspects of a program from its functional aspects. This, in turn, makes possible the use of non-standard computational models (that reflect the ideal machines encoded in a restricted coordination scheme) to formalize the semantics of designs, i.e., behavioral semantics including timing, fault tolerance, security, and so forth. At the same time, standard approaches can be used to specify the semantics of the functional aspects of a container.

I am prepared to give a brief description of the "sporadic server" container, the priority-based coordination scheme it uses to guarantee bounded invasiveness on periodic deadlines in real-time systems, and its associated queuing-based computational model for analyzing the average execution time of programs executing within the sporadic server container.

Summary Session: Semantics of Specification Languages for Component-Based Systems

Kurt Wallnau (CMU - Pittsburgh, USA)

The top five results from discussion, plus several pages of raw notes.

Joint work of: Brogi, Antonio; Fisler, Kathy; Goos, Gerhard; Jahnke, Jens; Levy, Nicole; Schmidt, Heinz; Wallnau, Kurt

Tales From Component Hell

Wolfgang Weck (Software Architecture Consultant - Zürich, CH)

When integrating third-party components, even trusted components may turn out to be not 100integrated system may fail. While improving trust into components is well worthwhile, we should not expect a zero-error scenario as a result. While raising probabilities is good, we should not expect them to become equal to 1. In such a world it is important for a component integrator to quickly produce evidence about which component of which vendor can be hold responsible for a specific problem. If you as an integrator cannot assign blame to a vendor, you will find yourself in Component Hell. We report examples from an industrial project.

Predicting Properties of Koala Assemblies

Rob van Ommering (Philips Research - Eindhoven, NL)

This presentation explains how we annotate software components with information on threading aspects of interfaces, and how we use this information to assess whether systems are composed out of components correctly with respect to thread synchronization.

Keywords: Software Components, Product Populations, Multi-threading analysis