Component-Based Software Engineering: Building systems from Components

at 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems

Ivica Crnkovic[1], Stig Larsson[2], Judith Stafford[3]

[1]Mälardalen University, Department of Computer Engineering, Sweden, ivica.crnkovic@mdh.se
[2]ABB Automation Technology Products, Sweden, stig.bm.larsson@se.abb.com
[3]Software Engineering Institute, Carnegie Mellon University, USA, jas@sei.cmu.edu

Abstract

This paper gives a short overview of the Workshop on Component-based Software Engineering – Building Systems from Components held at 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems in Lund, Sweden, April, 2002. The aim of the workshop was to bring together researches and practitioners from system engineering, software architecture and from component-based software engineering communities in order to exchange experiences and research results from these domains.

1 Introduction

Component-based Software Engineering (CBSE) is concerned with the development of software systems from reusable parts (components), the development of components, and system maintenance and improvement by means of component replacement or customization.

Building systems from components and building components for different systems requires established methodologies and processes not only in relation to development/maintenance phases, but also to the entire component and system lifecycle including organizational, marketing, legal, and other aspects. In addition to objectives such as component specification, composition, and component technology development that are specific to CBSE, there are a number of software engineering disciplines and processes that require methodologies be specialized for application in component-based development. Many of these methodologies are not yet established in practice, some have not yet been developed.

The progress of software development in the near future will depend very much on the successful establishment of CBSE; this is recognized by both industry and academia. The growing interest in CBSE is reflected in the number of workshops and conferences with CBSE tracks [2-5]. The Workshop on Component-based Software Engineering held at 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems in Lund, Sweden [1] followed that trend by setting focus on the system level – Building Systems from Components.

2 The aim of the workshop

Component-based approach is not new. Components are explicitly addressed in software architecture and commonly used in system engineering. Yet, there is different understanding of components in these areas. While the primary goal of software architecture and system engineering is to understand the system by dividing it in components and identifying components as composable units that express certain functions and properties, CBSE starts from the given properties of the components and then defines a system by utilizing these properties. The main idea for the workshop was to analyze and compare these approaches, to find common understandings and possibly cross-fertilize the best ideas and practices from these areas. Systems attributes in relation to component attributes and the composition process were the primary subjects of the workshop.

The following areas of interest were listed in the Call for paper:

- Software architecture as related to CBSE;
- Analysis/design methods for building component-based systems;
- Selection/evolution criteria for components and assemblies of components;
- Predictability of component compositions;
- Configuration management of components and component compositions;
- Verification of systems based on component attributes.

3 The Workshop Results

In total 14 papers were selected and presented at the workshop; in total, 22 researchers and practitioners participated. The workshop was divided, according to the topics of position papers, into six sessions: a keynote speech and five working sessions. Each working session started with a 10 minutes presentation of each paper and then it continued with discussions related to the session's topic. In this way the discussions were focused on more general and more important aspects, rather then on particular papers.

The keynote speech was an excellent introduction to the workshop since it gave an overview of development of a complex system using a platform- and component-based approach, while at the same time considering the system aspect, emphasizing the importance of considering the system as a whole. The speech sparked a lively discussion about differences between system engineering, a typical top-down approach where a whole picture of the system is the most important, and a component-based software engineering which emphasis the role of software, software architecture and software components.

The working sessions covered both theoretical/research and practical aspects of CBSE and covered the following topics:

- CB Development Process
- Software Architecture and CBSE
- Predictable composition
- Dynamic Component-based Systems
- CBSE and Formal Methods

The presentations and the discussions suggested that the workshop focus on addressing two types of questions. One type was: how to design a system and identify the properties of components that are needed for the system? The next stage in this approach was considered to be either find such a component or to generate it, or to develop it. The second type of question was, How to specify a component so that its specification can be properly understood in the context of system requirements and how to predict/derive the system properties from the measurable and specified components? Although the workshop did not identify a unique, common approach, it clearly points out that there is a need for combining the experiences and results from both approaches. In addition to discussion of these principle questions, several case studies (for example Nokia, ABB, Ericsson, Philips) were presented, illustrating the state of the practice as well as main problems and requirements in industry related to component-based development.

The sections that follow include a summary of the keynote speech and abstracts of the papers. The full papers can be found in [6] and the presentation material in [7].

4 Keynote Speech

Peter Ericsson, *Industrial experience of using a component-based approach to industrial robot control system development.*

The keynote speech addressed the experience that ABB Automation Technology Products, Robotics has gained during ten years of development of today's robot controller software, supporting simulation systems and communication software. ABB produces and delivers industrial robot systems to a variety of application fields such as those for car manufacturing, foundry, painting and food packaging. The controller software represents a huge and complex system with several million lines of code and several hundred man-years of development. Many different software engineering fields such as real-time, motion control, databases, application programming language, communication, and human-machine interaction are combined in these products and increase the demands on the development process as well as the system architecture.

The talk addressed the following subjects:

- Organization issues;
- Development processes;
- System architecture;
- Test strategy; and
- Legal and commercial issues.

Since the development and maintenance of this system has been lasted more than ten years, many strategic decisions are related to modifiability and maintainability aspects. In addition the decisions related to the system functionality the following primary goals for the system development has been identified: Understandability, reusability, software architecture, development processes, quality, adaptability, and openness. The key factor for the success was the coherence between organization and system architecture. The basic architecture is a component based, and each organizational unit is responsible for their components. The development process is integration-intensive. The software architecture design plays a crucial role in the development. The platform is designed to support the components' development based on selected design patterns. Components of different size and complexity coexist and represent the basic elements of the system architecture. Most of the components were internally developed but many components (both application and platform) are external. A holistic and recursive use of key design patterns in the platform itself was emphasized.

The following design patterns have been used: Object oriented design, event driven real time kernel, message bus (support of asynchronous communication), subscription–distribution servers, framework engines, multi-layered architecture, and client–server.

A standing challenge in the development is find the balance between platform and application. To achieve this deep domain knowledge is required in order to identify which parts should belong to the platform and which to the application components. The choice depends on analysis on many questions, such as: What will be subject for frequent changes? What are the possible configurations? What are the most probable key scenarios on future system requirements? A too "thin" platform gives insufficient support for application development; a too "fat" platform introduces problems such as to large complexity, inflexibility, and scalability.

Challenges related to component-based development are many:

- Lack of established processes and tools for component-based embedded system development makes it difficult to maintain and transfer know-how to new development teams
- Few common component models exist outside the server and desktop domain.
- Hard to find suppliers of software components other than the basic operating system and related components
- Cope with changes where the operating system suppliers includes both infrastructure and domain specific components.

An interesting observation was that system engineering, which includes both hardware and software parts, is mainly concentrated on the software parts. Hardware is updated more slowly than software and, when done, is more easily accomplish. On the other hand, the software system is often used on several different hardware platforms without requiring additional resources.

5 Presentation and Discussion Sessions

This section summarizes the papers by inclusion of the papers' abstracts.

5.1 Session I - CB Development Process

Antonia Bertolino, Andrea Polini, Re-thinking the Development Process of Component-based Software

This paper contribution to the ECBS workshop is a position statement that a wide gap exists between the technologies for Component-based Software Engineering and the scientific foundations on which this technology relies. What is mostly lacking is a revised model for the development process. We very quickly outline a skeleton for re-thinking the models that have shaped the software production in the last decades, and we start to make some speculations, in particular for what concerns the testing stages. As a working example, we take in consideration the Enterprise Java Beans framework. However, our research goal is to draw generally valid conclusions and insights.

Jonas Hörnstein, Håkan Edler, Test Reuse in CBSE Using Built-in Tests

Component-based software engineering (CBSE) is expected to drastically reduce the time spent on developing software through the use of prefabricated components. However, some of the time gained on reusing components instead has to be spent on testing that components work as specified the new environment. The Component+ project aims at solving this by using built-in tests. This paper presents architecture for the integration of built-in tests in software components that makes it possible to reuse tests and hence minimize the time spent on testing.

5.2 Session II - Software Architecture and CBSE

Iain Bate, Neil Audsley, Architecture Trade-off Analysis and the Influence on Component Design

The production and assurance of systems that are safety-critical and/or real-time is recognized as being costly, time-consuming, hard to manage, and difficult to maintain. This has lead to research into new methods whose objectives include:

- Modular approaches to development, assurance and maintenance to enable: Increased reuse; Increased robustness to change and reduced impact of change.
- Integration strategies that allow systems to be procured and produced by multiple partners, and then efficiently integrated;
- Ways of determining the approach likely to be the "best" (the best can only be found with hindsight);
- Techniques for identifying and managing risks.

Many of the component-based engineering techniques are considered relatively mature for developing dependable components and ensuring correctness across their interfaces when combined with other components, e.g. approaches based on rely-guarantees. This paper addresses the following key remaining issues:

- How the system's objectives should be decomposed and designed into components (i.e. the location and nature of interfaces); and
- What functionality the components should provide to achieve the system's objectives.

The paper develops a method for:

- Derivation of choices – identifies where different design solutions are available for satisfying a goal.
- Manage sensitivities – identifies dependencies between components such that consideration of whether and how to relax them can be made. A benefit of relaxing dependencies could be a reduced impact to change.
- Evaluation of options – allows questions to be derived whose answers can be used for identifying solutions that do/do not meet the system properties, judging how well the properties are met and indicating where refinements of the design might add benefit.
- Influence on the design – identifies constraints on how components should be designed to support the meeting of the system's overall objectives.

Hans de Bruin, Hans van Vliet, The Future of Component-Based Development is Generation, not Retrieval

Component-Based Development (CBD) has not redeemed its promises of reuse and flexibility. Reuse is inhibited due to problems such as component retrieval, architectural mismatch, and application specificness. Component-based systems are flexible in the sense that components can be replaced and fine-tuned, but only under the assumption that the software architecture remains stable during the system's lifetime. In this paper, we argue that systems composed of components should be generated from functional and nonfunctional requirements rather than being composed out of existing or newly developed components. We propose a generation technique that is based on two pillars: Feature-Solution (FS) graphs and top-down component composition. A FS-graph captures architectural knowledge in which requirements are connected to solution fragments. This knowledge is used to compose component-based systems. The starting point is a reference architecture that addresses functionality concerns. This reference architecture is then stepwise required to cater for non-functional requirements using the knowledge captured in a FS-graph. These requirements are the architecture-level counterpart of aspect weaving as found in Aspect-Oriented Programming (AOP).

Ioana Sora, Pierre Verbaeten, Yolande Berbers, Using Component Composition for Self-customizable Systems

Self-customizable systems are equipped with mechanisms to automatically adapt themselves to a set of user requirements or to their environment. We address this customization problem through component composition. Our approach is based on hierarchically decomposed component systems, deploying composed components as a means of abstracting details. Composition is performed in a stepwise refinement manner, which allows to handle the complexity of the system and to realize very fine-tuned compositions even when composition decision is made automatically. The composition strategy is driven by anonymous dependencies established between components by their requirements. Our goal is to perform unanticipated customizations with as few user interventions as possible. We evaluate and prove our composition approach by building customized network protocols.

Frank Lüders, Andreas Sjögren, Case Study: A Component-Based Software Architecture for Industrial Control

When different business units of an international company are responsible for the development of different parts of a large system, component-based software architecture may be a good alternative to more traditional, monolithic architectures. The new common control system, developed by ABB to replace several existing control systems, must incorporate support for a large number of I/O systems, communication interfaces, and communication protocols. An activity has therefore been started to redesign the system's architecture, so that I/O and communication components can be implemented by different development centers around the world. This paper reports on experiences from this effort, describing the system, its current software architecture, the new component- based architecture, and the lessons learned so far.

5.3 Session III - Predictable composition

Ralf H. Reussner, Heinz W. Schmidt, Using Parameterized Contracts to Predict Properties of Component Based Software Architectures

This position paper presents an approach for predicting functional and extra-functional properties of layered software component architectures. Our approach is based on parameterized contracts a generalization of design-by-contract. The main contributions of the paper are twofold. Firstly, it attempts to clarify the meaning of "contractual use of components" a term sometimes used.

E.M. Eskenazi, A.V. Fioukov, D.K. Hammer, M.R.V. Chaudron, Estimation of Static Memory Consumption for Systems Built from Source Code Components

The quantitative evaluation of certain quality attributes – memory consumption, timeliness, and performance – is important for component-based embedded systems. We propose an approach for the estimation of static memory consumption of software components. The approach deploys the Koala component model, used for embedded software in TV sets. There are two main parts in the method: specification of the memory demand of components and estimation of memory demand for systems built of these components. The proposed method allows flexible trade-off between estimation effort and achievable precision, yet requiring no changes in the tools supporting the Koala component model. The method may be extensible to include other resource attributes as well.

Yu Jia, Yuqing Gu, The Representation of Component Semantics: A Feature-Oriented Approach

In this paper a semantic model for component is proposed which is structured in three parts called Domain Space, Definition Space and Context Space. We also argue that the feature-oriented method is an effective and practical approach to fulfill the semantic model.

5.4 Session IV- Dynamic Component-based Systems

Ahmed Saleh, Component-based Environment For Distributed Configurable Applications

One of the basic requirements for distributed applications to run under different working environments is to be flexible, configurable, portable and extensible. Using the current development techniques independently falls short in supporting most of these requirements due to complexity of their integration and the conflict of their objectives. In this context this paper describes an integrated environment based on an interface description language called NCSL, an architecture description language called NADL, and a supporting management system composed of a component-based framework and an event management system that facilitate the process of developing and managing distributed configurable applications based on their non-functional requirements (NFRs).

Ian Oliver, Quality of Service Specification in Dynamically Replaceable Component Based Systems

When working with embedded environments that can automatically download components on as as-needed basis it is necessary to ensure that we do not place too much stress (CPU overload, Memory overload etc) on the system in order to achieve optimal performance for the user. In order to facilitate this one must incorporate quality of service information into the components and perform suitable tests upon this information in order to decide whether to download the component or not. One issue here is how this information is presented, stored and what information should be carried by the component. There are also issues with what the information means and from where it is collected. In this position paper we describe our initial efforts in specifying the quality of service information and also explore some of the implementation issues we have found.

Ronan Mac Laverty, Aapo Rautiainen, Francis Tam, Software Component Deployment in Consumer Device Product-lines

Effective deployment of components is imperative for consumer device manufacturers; these must utilize the resources available optimally. For single systems this is a standard software engineering problem, but for product-lines new techniques must be devised. These are needed to allow component reuse while minimizing the overhead from cross product components. To achieve this prototype for a tool to automatically generate and evaluate a deployment for a consumer device has been developed. This system and the motivation behind its development are described below, including directions for its future development.

5.5 Session V - CBSE and Formal Methods

Rebeca P. Díaz Redondo, José J. Pazos Arias, Ana Fernández Vilas, Reusing Verification Information of Incomplete Specifications

The possibility of verifying systems during any phase of the software development process is one of the most significant advantages of using formal methods. Model checking is considered to be the broadest used formal verification technique, even though a great quantity of computing resources are needed to verify medium-large and large systems. As verification is present over the whole software process, this amount of resources is more critical in incremental and iterative life cycles. Our proposal focuses on reusing incomplete models and their verification results — which are obtained from a model checking algorithm— to reduce formal verification costs in this kind of life cycles.

6 References

[1] 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems, http://www.cigital.com/conferences/ecbs02/

[2] 4th and 5th ICSE Workshops on CBSE: Component Certification and System Prediction, Benchmarks for Predictable Assembly, http://www.sei.cmu.edu/pacc

[3] 27th and 28th Euromicro Conferences: CBSE track, http://www.idt.mdh.se/ecbse

[4] First International Working Conference on Component, http://swt.cs.tu-berlin.de/cd02/

[5] ICSR7 2002 Workshop on CBD Processes, http://www.idt.mdh.se/CBprocesses

[6] Workshop Proceedings Component-based Software engineering: building systems from components, Technical report 2002-05, Mälardalen University, Västerås, Sweden, 2002

[7] Workshop on Component-based Software engineering: building systems from components, http://www.idt.mdh.se/~icc/cbse-ecbs2002/