Available online at www.sciencedirect.com

SCIENCE DIRECT®

The Journal of Systems and Software 74 (2005) 1–3

The Journal of Systems and Software

www.elsevier.com/locate/jss

Guest editorial

Automated Component-Based Software Engineering

Following the successful *5th ICSE Workshop on Component-Based Software Engineering: Automated Component-Based Software Engineering* held in Orlando, Florida, in May 2002, this special issue of *The Journal of Systems and Software* is devoted to automated component-based software engineering. We depend on software components for everyday activities at work, at home, in traffic and transport, banking, health, telecommunications, defence and other areas. Many times the software components are part of mission-critical functions and services provided in these domains. Primary reasons for component production and deployment are: separability of components from their context; independent component development, testing, configuration and later reuse; upgrade and replacement in running systems. However, compositionality of component-based systems is often taken for granted. Moreover, component technologies are not entirely independent of particular hardware or operating platforms, programming languages or the specific middleware technology in which they are based. Ideally, the development, quality control, and deployment of software components should be automated similarly to other engineering domains, which deal with the construction of large systems from well-understood components with predictable properties and under acceptable budget and timing constraints.

Automated component-based software engineering is emerging as a field of study in software engineering. There are many open issues that need to be resolved before a component-based approach can make a significant impact on mission-critical software automation. Methods must be developed that allow measurement and prediction of functional and extra-functional characteristics such as availability, adaptability, security, and performance. Analysis and design methodologies whose goals are automation or partial automation are still lacking in the required formalisation and automation support. Middleware technologies such as Microsoft's .NET and Sun Microsystem's EJB are slowly moving into the domain of automated component-based software engineering. Their emphasis is typically on generation, such as glue code generation or user-interface generation; support for automated testing and quality control is only nascent.

Our ability to formally model and reason about component-based systems is vital to any endeavour to automate the process of component development, adaptation, integration and deployment. System integration and changes to architecture may have significant impact on critical system properties and overall system quality; frequently, small structural changes have a discontinuously large impact on such properties.

CBSE workshop history

Beginning in 1998, there have been six successive workshops on component-based software engineering held in conjunction with the International Conference on Software Engineering. CBSE workshops have brought together researchers and practitioners from many points around the globe—and from several fields closely connected to CBSE, notably software architecture, component technology and trustworthy, dependable systems.

Papers in this special issue

This JSS special issue is dedicated to *Automated Component-Based Software Engineering*.

The call for papers was widely disseminated through mailing lists and news groups of the software engineering community. Authors of the CBSE6 workshop were encouraged to develop submissions from their workshop contributions. The call was open and all papers were subject to the same rigorous review process with three to four independent peer reviews per paper. We received 28 papers of which seven were selected.

The papers in this issue represent several areas of automated software engineering research. The first group of papers takes an analytic approach starting from component-based models and aiming at quality assurance, improvement and prediction. The quality of components is a key factor in the overall service quality and performance of any component-based system. Analysis of software artifacts for system quality assurance connects early architecture and component design with runtime execution of reusable deployed components to achieve model-driven quality assurance, testing and verification, and prediction and certification.

0164-1212/$ - see front matter © 2003 Published by Elsevier Inc.
doi:10.1016/j.jss.2003.11.016

John Grundy, Guoliang Ding and John Hosking present an approach to testing deployed components 'on site' using an aspect-oriented approach to dynamic validation agents. Such agents test components at runtime against functional and extra-functional requirements. The agents are generated from design-time descriptions given as concerns cutting across components.

Mohammad Zulkernine and Rudolph Seviora harness the executability of deployed reusable components for automatic testing of software architectures and their integration. Their methods employ communicating state machines and aim at improving the overall service quality of component-based software systems by using a specification-based approach to component interfaces and component-based architectures.

Roel Wuyts, Stéphane Ducasse and Oscar Nierstrasz model small embedded real-time components (software and devices) and develop scheduling and analysis methods aimed at meeting hard and soft real-time constraints in systems of such components. To this end the semi-automatic approach taken in their Pecos system combines constraint satisfaction techniques with rate-monotonic analysis for periodic, aperiodic and sporadic tasks. Their data-centric approach models these kinds of tasks by distinguishing active, passive and event-driven components respectively.

Shiping Chen, Yan Liu, Ian Gorton and Anna Liu report on feasibility studies towards a practical parameterized performance prediction method for component-based application systems. Based on their previous work, which demonstrates that the performance of a given application depends critically on the underlying component technology and platform, the studied approach is technology-specific: it uses predictor model parameters that may be actualised differently for different platforms. The paper shows that accurate predictions are feasible in *n*-tier distributed enterprise application systems for current industry-strength platforms.

The second group of papers takes a more generative approach aiming at partially automated software construction from component-based software models. Adaptation, synthesis, and generative approaches more generally use component-based specifications and designs for generating code or code skeletons automatically.

The success of architecture and product-line based approaches to component-based software systems necessitates the distinction between component adaptation, which involves interface and contextual changes, and component implementation reuse 'as is'. In the formal approach to component adaptation described by Andrea Bracciali, Antonio Brogi and Carlos Canal, declarative component and adaptor specifications are utilised to overcome mismatches between reused components and their deployment environment by automatic adapter generation.

Marcelo R. Campo, J. Andrés Díaz Pace and Federico U. Trilnik focus their work on supporting novice users of component-based technologies, notably Java, UML and aspect-oriented programming. They present Hint, an environment for assisting the instantiation of Java applications. The underlying method uses agents capable of proposing programming activities aiming at building new applications satisfying well-defined requirements. The most relevant contribution of this work is the use of novel planning techniques to guide the execution of configuration and instantiation activities.

Gerald C. Gannod, Sudhakiran V. Mudiam and Timothy E. Linquist apply their expertise to a publish-subscribe development paradigm integrating data and behavior. Applications in this paradigm integrate services semi-autonomously, based on target runtime properties and policies, by generating wrappers and synthesizing code according to high-level architectural specifications.

As a group, this set of papers provides a broad coverage of model-driven, analytic and generative approaches to automating parts of the component-based development process.

Acknowledgements

We would like to acknowledge the support of the Software Engineering Institute at Carnegie Mellon University, Pittsburgh, USA as well as Monash University, Melbourne, Australia and Mälardalen University, Västerås, Sweden.

We are especially grateful to the community effort contributing to this special issue: the many authors who made the effort to submit papers, and the editorial board members who produced their reviews at short notice and within a tight time window.

Ivica Crnkovic
Mälardalen University
Västerås, Sweden

Heinz W. Schmidt
Computer Science and Software Engineering
Monash University
Wellington Road, 3800
Clayton, Vic., Australia
E-mail address: hws@csse.monash.edu.au

Judith Stafford
Tufts Unversity
Boston, USA

Kurt Wallnau
Software Engineering Institute
Pittsburgh, USA

Available online 21 January 2004