Available online at www.sciencedirect.com

SCIENCE DIRECT®

Science of Computer Programming 56 (2005) 1–4

Science of Computer Programming

www.elsevier.com/locate/scico

Preface

New software composition concepts[☆]

Component-based software engineering has not fulfilled the earlier promise of plug-and-play system production. The reasons may be found in the production details which are much more sophisticated than this high-level glance buoyed up with false hopes. However, although not being able to reach such an easy plug-and-play utopia, new work led to promising results.

An important step is taken by the EU funded project EASYCOMP (IST-1999-14191) which has promoted the development of new software composition concepts. The consortium has addressed the composition problems on the detailed technical and process level well-rounded with industrial case studies.

The goal of EASYCOMP was to provide composition technology to support the end-user on all levels easing the task of constructing systems from components. The results range from methodology and composition techniques to composition infrastructure and case studies (http://www.easycomp.org). Some of these results are presented in separate paper contributions within this special issue.

Besides the project work to promote composition concepts, the EASYCOMP consortium has launched this special issue aiming at publishing the most promising approaches not only of the EASYCOMP consortium itself but of international rank. From 46 submissions 13 high-quality papers have been accepted for this special issue.

With respect to the accepted contributions the production of component systems and the composition issues may be classified into the following problem domains:

- Component Development Processes
  System development means organization of time and resources. Developing component systems has specific requirements.
  Kantorowitz and Lyakas present a framework which supports a use case-driven (based on natural language) development.

[☆] This special issue is partially supported by the European Community under the IST programme—Future and Emerging Technologies, contract IST-1999-14191-EASYCOMP. The authors are solely responsible for the content of this communication. It does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of data appearing therein.

0167-6423/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2004.11.001

One issue with component system development is the heterogeneity of the involved components. Arató, Mann and Orbán address this issue by combining hardware and software components in the development process.

- Component Development Techniques, Component-specific Languages
  Each paradigm has usually specific languages which support its specific abstractions. As for object-oriented programming this is also a requirement for component-oriented programming.
  Component-specific language support is the content of the work of Fröhlich, Gal and Franz and their language Lagoona which supports component-specific concepts like distributed extensibility.
  Lumpe and Schneider express a variety of inheritance mechanisms by a uniform language-neutral concept / model to bridge between different component models.
  Assmann emphasizes the importance of architectural styles for specific component types, the active document components. General and reoccurring concepts like invasiveness, transconsistency, staged architecture are identified.
- "Componentization" and Component Modeling / Specification
  A question is how we build the individual components which are targeted to be combined to a component system. Approaches may be divided into those which derive components from existing (legacy and non-component) software and those which build the components from scratch. The latter are close to component-specific languages.
  Washizaki and Fukazawa follow the former by extracting components from existing object-oriented software. Components are identified based on reusability metrics; components and environment are refactored to preserve the original semantics.
  Scheben introduces a component specification and modeling approach. Based on a type system for (hierarchical) components a formal definition of exchangeability and interoperability is provided.
- Component Mining
  Reuse requires first the detection of suitable existing components. Mining deals with this problem of identification and selection based on suitability criteria.
  Gross, Melideo and Sillitti developed the CLARiFi component-broker platform with certification mechanism to support the collection, management, location, evaluation and selection of components.
- Component Adaptation
  Having selected a component for its reuse in the targeted component system the system developer will have to adapt the component before or while its composition in most cases.
  Liu, Wang and Kerridge developed the scenario-based dynamic component adaptation and generation (SAGA) approach to overcome the mismatch between component and reuse context and to reduce the adaptation overhead.
- Composition Verification, Interference Detection and Testing
  As in all software development errors come together with functionality. For that reason, component-specific verification, interference detection and testing approaches are necessary.
  Dias da Silva and Perkusich use component-based Petri nets to visualize the structure and model the behavior of software architectures and components. Model checking in