

Panel on “What are the most urgent research problems of component-based software engineering for resource-constraint systems?”

Chair:

- Dieter Hammer, Dptm. of Computer Science, Eindhoven Univ. of Technology, Netherlands; hammer@win.tue.nl

Participants:

- Claudio Becchetti, Dptm. of Software Development, Marconi, Italy; Claudio.becchetti@marconicomms.com
- Priya Narasimhan, Dptm. of Electrical and Computer Engineering, Univ. of California at Santa Barbara, USA; priya@alpha.ece.ucsb.edu
- Andy Schürr, Institute for Software Technology, University of the German Armed Forces, Germany; <mailto:andy.schuerr@unibw-muenchen.de>
- Bran Selic, Rational Software Inc., Canada; bselic@rational.com

1. Introduction

Up to now, the research community did not give much attention to CBSE (Component-Based Software Engineering) in the presence of resource constraints. Nevertheless, the handling of resources is of utmost importance if we want to construct dependable real-time systems in a compositional way. Even if all functional properties of the components (interfaces and behavior) are fully described, the properties of the resulting system can usually not be derived because of resource dependencies. Components that run perfect on a given platform, either in isolation or in a particular configuration, will cause problems on other platforms or in other combinations because of memory-, input/output- or scheduling conflicts. In particular, achieving composability in the timing domain is a hard problem.

Since many systems (like embedded systems and safety-critical systems) have severe dependability constraints (timeliness, performance, reliability, availability, safety and security), this is an unacceptable situation. Although many solutions for specific problems are known, there are no component models and component frameworks that deal with resource constraints in a systematic, predictable and practical way. This is a major challenge for the research community. The goal of this panel was an categorization of the most important research questions in this area.

Prior to the panel, **Dieter Hammer** summarized his view of the problem and the most appropriate solution directions in his position paper “Component-Based Software Engineering for Resource-Constraint Systems: What are the Needs?” that is part of these proceedings. The most important statements were:

- Component models must support loose component coupling in order to achieve flexibility and configurability.
- The component model must support the specification and verification of end-to-end dependability constraints. Dependability and especially time must become first-class design dimensions!
- The component specification must include resource requirements and interaction styles (protocols). Resource consumptions should be specified in a platform-independent way.
- The execution platform should support the implementation of dependability. The challenge is the combination of different dependability aspects!
- Visual configuration and verification environments must support the use of component models.

2. Panel discussion

In order to start the discussion, also the panel members presented their position.

Claudio Becchetti described the BCSCADA architecture that is developed by the Network Management Systems department of Marconi. It has the following features:

- Software component “plug & play” model similar to hardware plug & play model .
- Loose component coupling by using a blackboard approach with event notification (observer pattern) and asynchronous communication.
- Exogenous specification (i.e. outside the components) of the component configuration information.
- Common scheduling policy and start-up resource allocation through a common framework.
- Worst-case simulation to define resource consumption.
- Use of Semi-Visual configuration tools and unattended worst case testing tools.
- Strong guidelines for software development.

This approach, clearly takes resources into consideration and proved to reduce the development cost by fostering component reuse and to increase the system reliability by supporting automatic simulation and testing. Nevertheless, there are a number of open issues like

- non simulated real time constraints verification with component mutual dependency,
- platform dependence and
- critical race conditions among modules.

Priya Narasimhan mentioned the following challenges:

- How do we capture resource constraints in a platform-independent way?
- How do we incorporate and enforce distributed resource constraints into component-based design? Leaving the job to the application developer exposes low-level system details and complicates the application. Should the middleware be given this task? Handling violations of resource constraints?
- How do we design components to share resources effectively in a resource-constrained environment? How do resource constraints affect the “-ilities” (e.g. dependability and security) of a component-based system? What are the tradeoffs? How to handle QoS degradation in a resource-constrained environment?
- How much impact has the introduction of resource constraints on existing component-based system?
- How do we design component-based systems to handle evolving and new resource constraints? Is there a difference between design-time, compile-time, run-time resource constraints?

Andy Schürr presented his view on Component Modeling Languages (CML's):

- Define standard visual CML's on top of middleware/binary standards, probably using concepts from UML extensions, Architectural Description Languages (ADL's) and DataFlow Languages (DFL's).
- Enforce communication via explicitly defined connections, where at least the permission to create a connection is statically defined.
- Support various types of required/offered connections between components, like continuous data propagation, asynchr. discrete event propagation, etc.
- Extend CML's with visual sublanguages that have a static type system for describing runtime reconfigurations instead of using runtime libraries calls (e.g. in C/C++ code fragments).
- Support the verification of semantic compatibility of interconnected interfaces/ports, e.g. based on refinement/equivalence of protocol definition automata.
- Support light-weight components and connections for logical modeling purposes: passive, no reflection, embedded in binary component, no dynamic binding, etc.

Finally, **Bran Selic** elaborated on the migration from object- to component-technology and stated a number of requirements that will be covered in the upcoming UML 2.0 standard that is expected in february 2002. The problem with objects are that they provide only one-way encapsulation (at their provides interfaces) and that their scope and granularity is limited (single interface, single address space, static interfaces, etc.). The requirements for CBSE are:

- 2-way encapsulation of components: the implementation of a component must access its environment only via well defined interfaces (required interfaces).
- Dynamic interfaces: Component interfaces should not only specify a list of operation signatures, but also the valid dynamic interaction sequences (protocols). Furthermore, protocols should also be components.
- Extended object model: It should be possible for components to have multiple (dynamic) interfaces, and to span address spaces/sites.
- QoS specification: Components should specify both the QoS they require from the environment and the QoS they provide at their interfaces, e.g. in terms of timing, processor and memory characteristics, etc.
- Component meta-interfaces: Components should provide reflective interfaces through which they can be dynamically customized to suite the idiosyncratic needs of a particular client.

3. Conclusions

Both the industry and the research community have started to realize that dealing with resource-constraints is essential for utilizing the advantages of component-based technology for real-time dependable systems. Some initial standards and implementations are appearing, but the bulk of the work still lies ahead. The audience agreed with the panel that the posed questions cover a big part of the problems to be solved.

A first step towards solving these problems could be the definition of appropriate component models for resource-constraint systems. These models must not only cover the functional aspects but also the non-functional aspects like dependability and the relevant “-ilities”.

However, the implementation of such models has to tackle a number of yet unsolved problems: Trading of contradictory non-functional requirements (e.g. timeliness and reliability); realistic resource estimations for modern execution platforms with their complex caching strategies and operating systems; design and implementation of appropriate distributed scheduling algorithms in order to guarantee end-to-end deadlines; resource-aware middleware; and much more.