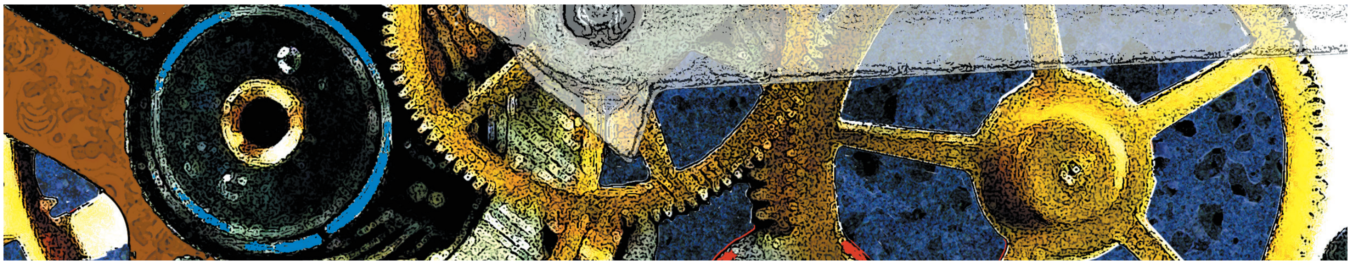
Software Components beyond Programming: From Routines to Services

Ivica Crnkovic, Mälardalen University

Judith Stafford, Tufts University

Clemens Szyperski, Microsoft



AT THE FIRST software engineering (SE) conference in 1968, Doug McIlroy introduced the concept of software components during his keynote speech, "Mass-Produced Software Components."[1] That components hold such an esteemed place in SE history should come as no surprise: componentization is a fundamental engineering principle. Top-down approaches decompose large systems into smaller parts—components—and bottom-up approaches compose smaller parts—components—into larger systems. Since 1968, components have played a role in both SE research and practice. For example, components have been an immanent part of software architecture from its early days.[2] In 1998, the International Conference on Software Engineering introduced component-based software engineering (CBSE) as a specific area within SE at the first workshop on CBSE, an event that eventually evolved into a series of symposia (http://cbse-conferences.org). In parallel, other events have addressed various CBSE-specific topics such as composability, predictability of functional and extrafunctional properties, modeling of component-based systems, reusability, deployment, software architecture and components, dynamic architecture, and middleware (see the sidebar "CBSE Terminology and Basic Concepts"). These topics have also become standard parts of many SE conferences (see the sidebar "CBSE Events"). Researchers and developers have taken notice: by March 2011, the Web of Science reported 1,546 publications containing both *software* and *component* in their titles; IEEE Xplore had 907 publications, and the ACM Digital Library had 1,254 titles. Clearly, this is a field experiencing a lot of growth.

A Shifting Paradigm

CBSE aims to build software from preexisting components, build components as reusable entities, and evolve applications by replacing components. This requires significant changes in the development paradigm, from both technical and business viewpoints. They require us to think from a top-down approach to a combination of top-down and bottom-up approaches, from analysis to predictability, from open source reuse to black-box reuse, and from selling

0740-7459/11/$26.00 © 2011 IEEE

CBSE EVENTS

Over the last 10 years, topics related to component-based software engineering (CBSE) have been regularly included at software engineering conferences. These topics include requirements management, software modeling and design, software testing, and software life cycle, as well as component-specific topics such as component compositions, interfaces, component models, and deployment. Conferences dedicated to components and CBSE itself also feature such topics.

- *International Symposium on Component Based Software Engineering* (CBSE; http://cbse-conferences.org). This annual conference focuses on various aspects of CBSE. It encompasses research (theoretical and applied) that extends the state of the art in component specification, composition, analysis, testing, and verification. CBSE is affiliated with CompArch, a federated conference event that also includes other events related to CBSE and software architecture.
- *International Symposia on Formal Methods for Components and Objects* (FCMO; http://fmco.liacs.nl). FMCO's objective is to bring together software engineering and formal methods researchers and practitioners to discuss concepts related to component-based and object-oriented software systems.
- *International Conference on Software Composition* (SC; http://swcomp2010.appspot.com). This conference addresses the challenges of how to compose software parts to build and maintain large systems.
- *Model-Based Development, Components, and Services Track* (MoCS; http://seaa2011.oulu.fi/index.php/mocs). Affiliated with the Euromicro Software Engineering and Advanced Application Conference, this track focuses on a combination of model-based, component-based, and service-based software engineering.
- *IFIP/ACM Working Conference, International Working Conference on Component Deployment* (CD). This occasional conference focuses on component deployment and related issues.
- *Specification and Verification of Component-Based Systems Workshop* (SAVCBS; www.eecs.ucf.edu/~leavens/SAVCBS). Affiliated with the European Software Engineering Conference (ESEC) and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE) conferences, this workshop concerns how formal techniques can help establish a suitable foundation for the specification and verification of component-based systems.
- *Workshop on Component-Oriented Programming* (WCOP; http://research.microsoft.com/en-us/um/people/cszypers/events/WCOP2011). Held as part of the Component-Based Software Engineering and Software Architecture (CompArch) conference series, this workshop serves as a doctoral symposium for young researchers on CBSE, software architecture, and software quality.

In addition, software engineering conferences such as the International Conference on Software Engineering (ICSE), ESEC/FSE, and Model Driven Engineering Languages and Systems (MODELS) regularly have topics related to CBSE.

final products to building up a market for components.

SE has witnessed significant advances in component-based development over the past two decades. Developers and researchers have created and deployed new technologies—for example, MS COM/DCOM, J2EE, the Corba Component Model, OSGi, and .NET MEF—that are widely used in many application domains, including distributed systems, office, and desktop applications. Many large software companies have built their own component-based technologies—for example, Koala and BlueAx, developed at Philips and Robert Bosch, respectively.

But in spite of its many successes, CBSE has also experienced certain drawbacks. Several promising concepts, such as component markets and automatic component search and deployment, remain realized only in small scale, and many concepts developed in the research community, such as formal component composition proofs, have only partially been established in practice.

The latest trend is component-based applications in embedded and real-time systems—for example, the Automotive Open System Architecture (Autosar). In recent years, CBSE's focus has seemingly shifted from components to services specified similarly to components through their interfaces—particularly, as just-in-time components used in dynamic application configuration to form the essential elements of service-oriented architecture, Web services, distributed systems, and cloud computing.

In This Issue

This special issue of *IEEE Software* highlights the wide variety of domains that utilize CBSE. It opens with a Point/Counterpoint that discusses the role of components in bridging the gap between software architecture and implementation. In the point piece, "Predictability by Construction: Meeting Programmers' and Architects' Concerns," Kurt Wallnau emphasizes a misconception in separating design from implementation. A component-based approach, with components as well-defined executable units and a predictable

CBSE TERMINOLOGY AND BASIC CONCEPTS

What is a software component? Is it a UML component, or is it a part of a system, like a database or graphical package? What are the specifics of CBSE, and which roles play components in CBSE? The term *software component* is almost as old as software engineering itself, but its definition and related terminology remain subjects of intensive discussions.

TERMINOLOGY

Software Component and Component Model

The most cited definition of *software components* is "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."[1] Another definition emphasizes the role of a component model: "A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard."[2] Here, "a component model defines a set of standards for component implementation, naming, interoperability, customization, composition, evolution and deployment."[2] The second definition gives a different meaning of a component—one defined by individual component models.

Interface

A *component interface* defines a set of component functional properties, that is, a set of actions that's understood by both the interface provider (the component) and user (other components, or other software that interacts with the provider). An interface has a double role—it's a component specification, and it's also a means for interaction between the component and its environment. In CBSE, component models distinguish the actions that components provide to their environment (that is, the *provided interface*) from the actions they require from this environment (that is, the *required interface*). In certain component models, a component interface also specifies extrafunctional properties—for example, timing properties, resource utilization, and dependability properties such as reliability and security—and other component metadata.[3]

Deployment, Binding, and Composition

Component deployment is a process that enables component integration into the system. A deployed component is registered in the system and ready to provide services. *Binding* is the process that establishes connections among components through their interfaces and interaction channels. In CBSE literature, binding is also often called *component composition*, which assumes a composition of the components' functions. In addition to function compositions, CBSE deals with the composition of the components' extrafunctional properties.

BASIC CONCEPTS AND PRINCIPLES

By its very name, CBSE implies building systems from components as reusable units and keeping component development separate from system development. This separation has significant

behavior of component assemblies, can help bridge the gap between architects and programmers. In his counterpoint, "Walking across the Seam," Philippe Kruchten states that this approach is useful, but it isn't enough and isn't essential to achieve productive communication between architects and programmers. Architect and programmer are roles and don't necessarily imply distinct individuals.

In their article "Facilitating Performance Predictions Using Software Components," Jens Happe, Heiko Koziolek, and Ralf Reussner show how CBSE concepts can help model and predict evolving systems' performance. A component-based architecture and suitable component technology first enable annotations of a component's extraimplications for business goals (for example, building a market for components), technologies (for example, on-the-fly deployment of new functionality), and legal and social issues (for example, trust, responsibility, and maintenance). To achieve its primary goals of increased development efficiency and quality and decreased time to market, CBSE is built on the following four principles.

Reusability

The entire CBSE approach is fully utilized only if the components, developed once, have the potential for reuse many times in different applications. Industry has established several reusability types as best practices: COTS (commercial off-the-shelf) components, product-line components, and open source components. CBSE is also useful in building architectural components for a particular system, without intention to reuse the components in other systems.

Substitutability

With substitutability, systems maintain correctness even when one component replaces another. This requirement boils down to the Liskov substitution principle:

Let $q(x)$ be a property provable about objects x of type T. Then $q(y)$ should be true for objects y of type S where S is a subtype of T.

This principle is feasible for functional properties, but it isn't obvious for extrafunctional properties because it depends on other factors, such as a system context—for example, a faster component can cause a deadlock and break timing requirements in a system using a nonpreemptive scheduling mechanism.

Extensibility

In CBSE, extensibility aims to support evolution by adding new components or evolving existing ones to extend the system's functionality. A typical solution to support component evolvability is to provide components multiple interfaces.

Composability

Composability is a fundamental CBSE principle. Every component-based technology supports the composition of functional properties (component binding). More rarely, there's support for composition of extrafunctional properties, for example composition of components' reliability, or execution time, or memory usage. Composition of extrafunctional properties remains one of the major challenges of CBSE research.

References

1. C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Professional, 1997.
2. G.T. Heineman and W.T. Councill, *Component-Based Software Engineering: Putting the Pieces Together*, Addison-Wesley Longman, 2001.
3. I. Crnkovic et al., "A Classification Framework for Software Component Models," *IEEE Trans. Software Eng.*, vol. PP, no. 99; doi: 10.1109/TSE.2010.83.

functional properties and then enable the analysis of system performance on the basis of component properties in different system architectures.

In "Components in the Pipeline: The MeDICi Approach," Ian Gorton, Adam Wynne, Yan Liu, and Jian Yin describe a component-based framework that efficiently manages a very large amount of scientific data with respect to scalability, modifiability, and complexity. The framework, which uses the pipeline paradigm, enables a direct mapping between architectural concepts and the resulting implementation.

In "Rigorous Component-Based System Design Using the BIP Framework," Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis present a component-based framework for the development of embedded and real-time systems. This domain is especially challenging because it has difficult requirements for timing properties, dependability, and resource utilization. The component-based framework the authors present allows a system specification at different levels of abstraction, including software and hardware components, transformations, and model composition, while preserving correct timing properties in the implementation.

Finally, in "Managing Evolving Services," Michael P. Papazoglou, Vasilios Andrikopoulos, and Salima Benbernou discuss an important issue: service evolution. Services and components have many things in common, such as interfaces and interaction. But they also have differences, such as deployment and composition. In addition to describing types of service changes, this article gives a brief overview of component and service evolution management mechanisms.

During several decades of SE development, software components have changed their form and their purpose. Starting as elements of source code, such as routines, procedures, modules, or objects, they transformed to architectural units and ready-to-execute blocks that are dynamically plugged into the running systems. CBSE contributed to advances in software development by giving components formal specifications and identifying a clear role for them as reusable units. In parallel to CBSE, components are essential elements of other SE approaches—for example, software reuse, software product lines and model-driven development. Such widespread adoption throughout various disciplines shows that software components will remain a key aspect of SE for a long time to come.

ABOUT THE AUTHORS

IVICA CRNKOVIC is a professor of industrial software engineering at Mälardalen University. His research interests include component-based software engineering, software architecture, software evolution, embedded systems software, and software development environments and tools. Crnkovic has a PhD in computer science from the University of Zagreb. He coauthored the book *Building Reliable Component-Based Software Systems*. Contact him at ivica.crnkovic@mdh.se.

JUDITH STAFFORD is a member of the faculty of the Department of Computer Science at Tufts University. She is also visiting scientist at the Software Engineering Institute at Carnegie Mellon University. Her research interests center around component-based software engineering and software architecture, more specifically the development of compositional analysis techniques for reasoning about quality attributes of assemblies of software components. Stafford has a PhD in computer science from the University of Colorado at Boulder. She coauthored the book Documenting Software Architectures. Contact her at jas@cs.tufts.edu.

CLEMENS SZYPERSKI is a principal software engineer at Microsoft. His recent work includes the Managed Extensibility Framework, a system enabling self-directed composition that's part of the .NET Framework 4.0. Szyperski has a PhD in computer science from ETH Zurich. He's the author of the books *Component Software* and *Software Ecosystem*. Contact him at Clemens.Szyperski@microsoft.com.

References

1. M.D. McIlroy, "Mass-Produced Software Components," *Software Engineering: A Report on a Conf. Sponsored by the NATO Science Committee*, NATO, 1969, pp. 138–155; http://www.cs.dartmouth.edu/~doug/components.txt.
2. M. Shaw et al., "Abstractions for Software Architecture and Tools to Support Them," *IEEE Trans. Software Eng.*, vol. 21, no. 4, 1995, pp. 314–335.