CREATIVE MATH. & INF.
16 (2007), 159 - 165

Dedicated to Professor Ioan A. RUS on the occasion of his 70^{th} anniversary

Overview and architecture of a component modeling tool

ANDREEA VESCAN and SIMONA MOTOGNA

ABSTRACT. Component-based development (CBD) advocates the acquisition, adaptation, and integration of reusable software components to rapidly develop and deploy complex software systems with minimum engineering effort and resource cost.

The paper first presents previous results regarding component-based development and describes the used architecture.

Further, the paper provides the steps (scenarios) when using a tool for developing a component-based system by assembling components: finding and selecting components or create new components; adapting components; syntactic system assembling, providing data and control flow; analyzing composition behavior, and deploying the system.

The previous work done by the authors is also described and an analysis about how to integrate them into the tool development is done.

1. INTRODUCTION

Component-based software development (CBSD) or component-based software engineering (CBSE) is concerned with the assembly of pre-existing software components into larger pieces of software. Underlying this process is the notion that software components are written in such a way that they provide functions common to many different systems. Borrowing ideas from hardware components, the goal of CBSD is to allow parts (components) of a software system to be replaced by newer, functionally equivalent, components.

The idea is not new. Componentizing software had been suggested by McIlorys [10] as a way of tackling the software crisis, yet only in the last decade or so has the idea of component-based software development taken off. Nowadays there is an increasing market place for Commercial Off-The-Shelf (COTS) components, embodying a buy, don't build [1] approach to software development. The promise of CBSD is a reduction in development costs: component systems are flexible and easy to maintain due to the intended plug-and-play nature of components.

Related projects. "Relatively little has been done to scale up analysis techniques for the purpose of providing automated analysis tools for component framework." [Cadena Project]

Received: 20.09.2006. In revised form: 11.01.2007
2000 *Mathematics Subject Classification.* 68N19, 68Q60.
Key words and phrases. *Component-based systems, formal models, component models.*

Cadena [2] is a research project (An Integrated Development, Analysis, and Verification Environment for Component-based Systems) that is currently being developed by faculty, staff, and students in the SAnToS Lab at Kansas State University. Cadena is an Eclipse-based extensible integrated modeling and development framework for component-based systems.

Rapide (The Stanford Rapide Project) [13] is designed to support component-based development of large, multi-language systems by utilizing architecture definitions as the development framework. Rapide allows gradual refinement of architectures into products, and supports testing and maintenance based on automated comparison with formal standard architectures.

Component Composition Software Kit (CCSK, 2004 master thesis [6]) is implemented as a simple integrated development environment. With CCSK we are able to check if given components can be compose in order to develop a new system and to provide all possibilities of composing components.

2. Tool overview and architecture

The tool that we want to construct will have, in principle, two important goals: to build the descriptive model in order to predict how the system could work, and to automatically perform certain checkings on this model, such that several situations may be avoided at an early stage in application development.

2.1. **Features & Functionalities.**

Our aim is to develop a model for component based systems such that consistency and efficiency can be verified before execution. There are two issues which need to be addressed where a software system is to be constructed from a collection of components:

- *Component integration* - the mechanical process of wiring components together. There has to be a way to connect the components together.
- *(Behavior) Component composition* - we have to get the components to do what we want. We need to ensure that the assembled system does what is required. The constituent components must not only plug together, they must perform well together.

In CBSD composition is a central issue, since components are supposed to be used as building blocks from a repository and assembled or plugged together into larger blocks or systems. The composition language should have suitable semantics and syntax that are compatible with those of components in the component model.

There are two points of views of composition: *syntactic (structural)* - to describe dependencies between components and *semantic (behavioral)* - to simulates system execution and observe behavior.

2.2. **Use cases.**

The main idea of the component-based approach is building systems from pre-existing components. This assumption has several consequences [5] for the system lifecycle. First, the development processes of component-based systems are separated from development processes of the components; the components should already been developed and possibly used in other products when

the system development process starts. Second, a new separate process will appear: finding and evaluating the components. Third, the activities in the processes will be different from the activities in a noncomponent- based approach; in system development the emphasis will be on finding the proper components and verifying them, and for the component development, design for reuse will be the main concern.

The waterfall model adapted to component-based development. The waterfall model was modified to emphasize component-centric activities [4].

Figure 1 shows the waterfall model and the meaning of the phases. Identifying requirements and a design in the waterfall process is combined with finding and selecting components. The design includes the system architecture design and component identification/selection.

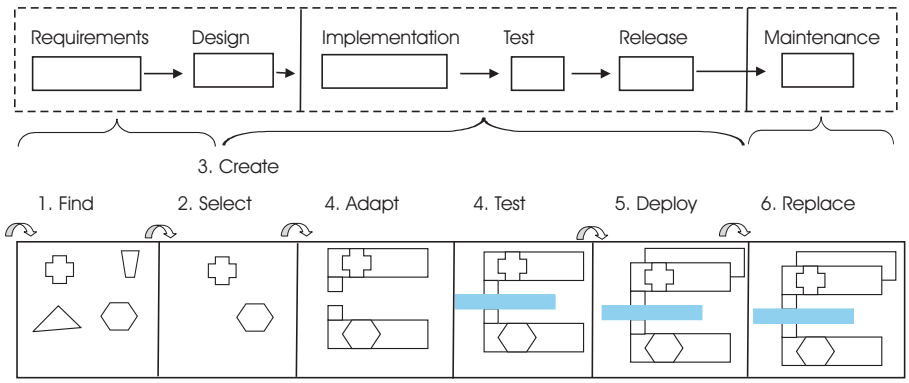

FIGURE 1. The development cycle compared with the waterfall model.

The use cases are developed having different centric points: components, flow, assembly and deploy.

The use cases included into the steps of the development process of a component-based system are:

- Components
  - *Find & select or create new.* Find components which may be used in the system. All possible components are listed here for further investigation. Select the components which meet the requirements of the system. Alternatively, create a proprietary component to be used in the system. In a component based development process this procedure is less attractive as it requires more efforts and lead-time.
  - *Adapt.* Adapt the selected components so that they suit the existing component model or requirement specification. Some components would be possible to directly integrated in to the system, some would be modified through parametrization process, some would need wrapping code for adaptation, etc.
- Flow - *Data flow* (by transitions) and *Control flow* (tasks to be executed, and in what order).

- Assembly - Syntactic integration of components into system based on data/control flow. By syntactically correct model we mean no semantic involvement, but just the way to connect the components together, the mechanical process of "wiring" components together (component integration).
- *Deploy & maintenance.* Compose and deploy the components using a framework for components. Replace earlier with later versions of components. This corresponds to system maintenance. Bugs may have been eliminated or new functionality added.

2.3. **Architecture.** The origins of software architecture as a concept was first identified in the research work of Edsger Dijkstra in 1968 and David Parnas in the early 1970ś. The scientists emphasized that the structure of a software system matters and getting the structure right is critical. The study of the field increased in popularity since the early 1990s with research work concentrating on architectural styles (patterns), architecture description languages, architecture documentation, and formal methods.

The software architecture [14] of a system comprises its software components, their external properties, and their relationships with one another. The term also refers to documentation of a system's software architecture. Documenting software architecture facilitates communication between stakeholders, documents early decisions about high-level design, and allows reuse of design components and patterns between projects.

An architecture, in our view, is a module containing a set of components (*Collection of interfaces*), a set of connection rules that defines the communication between the components (*Collection of interface connections*), and a set of constraints that define conditions for valid behavior (*A set of constraints*).

2.4. **Inside algorithms.** Algorithms were already developed to support component-based development. These algorithms address the component integration issues and also model construction and execution.

Table 1 contains an overview of already developed algorithms grouped by component integration issues and by model construction and model execution.

The scope of the algorithms from [7], [8] was to construct all the component-based systems from a set of given components, as a need for the first step of the assembly (*Component integration*). See Section 2.4.1. The models are constructed based on the architecture (component specification and communication between components) from [11], [12].

The second step of the assembly (*Behavior Component composition*) was accomplished by the algorithm from [9]. The execution of the obtained assembly consists of sequences $(Operations, ComponentsSet)$. See Section 2.4.2 for details.

2.4.1. *Model construction properties.* The resulting final models are checked from distinctive perspective - the properties: lost data (a data is lost if no other component from the system is using it) and just one provider/port (the data for each inport must be received by only one provider). The "reverse" propagation of data (the output data of a component is propagated to different inports) is allowed:

TABLE 1. Developed algorithms, addressing Component Integration (MI), Model Construction (MC) and Model Execution(ME)

Issue	**Paper**	**Algorithm**
MC	ICAM3 (2002)	A formal model for components
MC	ICCC (2004)	Component system checking using compositional analysis
CI	ICAM4 (2004)	Automata-based compositional analysis of component systems
CI	ZAC (2004)	A formal model for component composition
ME	ZAC (2005)	Specification, construction and execution of a component-based model
CI	Studia (2006)	Automata-based component composition analysis
MC	MaCS (2006)	Syntactic analysis of component composition
MC	EUROMICRO 32^{th} (2006)	Syntactic automata-based component composition
MC	SYNASC (2006)	Restraint order component model execution

component C_3 distributes the data d_3 to C_2 and C_6 component. See Figure 2 for details.

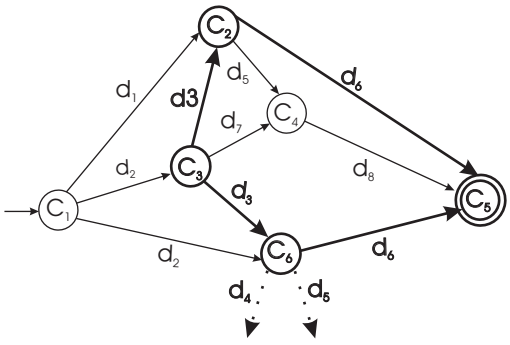

FIGURE 2. A model with lost data (d_4, d_5) and with more than one data provider/inport (d_6 for C_5 component).

2.4.2. *Model execution.* The execution of the obtained component-based system is composed of sequences of the form

$$(\{Op_1, C_1\}, \{Op_2, C_2\}, ..., \{operations, components\}),$$

where for each $i \geq 1$, Op_i is a subset of possible operations and C_i is a subset of components ready for execution.

The possible operations are:

- *propagation* - this rule moves values that have been generated by a component along connections from the component's outport to other components;
- *evaluation* - the component function is evaluated and the result is passed to the output of the component.

If at a given time, both types of operation can be performed, the propagation operation is chosen. We have developed two types of execution when many evaluation operations are possible: randomly chosen a component and restraint order execution.

3. FUTURE WORK

We have presented in this paper an analysis of a new component modeling tool that we are going to develop. A very systematic and detailed analysis of the functionalities, architecture and algorithms was performed in this article.

We have also provided a short analysis of our previous developed algorithms that will help us to develop the tool. These algorithms address the component integration issues and also model construction and execution.

Further work can be done in the following directions:

- Building a tool - to incorporate such functionalities.
- Checking if the build model supports a given sequence of tasks.
- Building a component-based system that contains a given sequence of tasks.
- Improving the assembly analysis (by model execution) with one more inport type.

REFERENCES

[1] F. Brooks, *No Silver Bullet: Essence and Accidents of Software Engineering*, Computer, Volume 20, Issue 4, 1987, pp. 10 - 19

[2] *Cadena Project*, http://cadena.projects.cis.ksu.edu/

[3] Ivica Crnkovic and Magnus Larsson, *Building Reliable Component-Based Software Systems*, Artech House publisher, 2002

[4] Ivica Crnkovic, *Component-based Software Engineering - New Challenges in Software Development*, Software Focus, Ed. John Wiley & Sons, 2001

[5] Ivica Crnkovic, Stig Larsson and Michel Chaudron, *Component-based Development Process and Component Lifecycle*, 27th International Conference Information Technology Interfaces (ITI), 2005

[6] Andreea Fanea, *Software Specification Methods*, Master Thesis, Computer Science Department, Babeş-Bolyai Univeristy, 2004

[7] Andreea Fanea and Simona Motogna, A Formal Model For Component Composition, *Proceeding of the Symposium "Zilele Academice Clujene"*, 2004, pp. 160-167

[8] Andreea Fanea, Simona Motogna and Laura Dioşan, *Automata-based component composition analysis*, Studia Universitas Babeş-Bolyai, Seria Informatica, Volume 51, Number 1, 2006, pp. 13-20

[9] Andreea Fanea, Specification, construction and execution of a component-based model, *In Proceeding of the Symposium "Zilele Academice Clujene"*, 2005, pp. 87-92

[10] McIlroys M.D., *Mass Produced Software Components*, In P. Naur and B. Randell, editors, Software Engineering, Scientific Affairs Division, NATO, 1969, pp. 138-155

[11] Simona Motogna, Petraşcu D. and Pârv B., *Automata-Based Compositional Analysis of Component Systems. Design and Implementation Issues*, In Fourth International Conference on Applied Mathematics (ICAM4), 2004, pp. 197-203
[12] Pârv B., Simona Motogna and Petraşcu D., *Component system checking using compositional analysis*, In Proceedings of the International Conference on Computers and Communications, 2004, pp. 325-329
[13] *Rapide Project*, http://pavg.stanford.edu/rapide/
[14] *Wikipedia*, http://en.wikipedia.org/

BABEŞ-BOLYAI UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
M. KOGĂLNICEANU 1
400084 CLUJ-NAPOCA, ROMÂNIA
E-mail address: camelia@ubbcluj.ro
E-mail address: avescan@cs.ubbcluj.ro