Towards Compositional Support for a Heterogeneous Repository of Software Components

Agnes F. N. Lumala and Ezra K. Mugisa

Composition of Software Components is a key activity in a component-based software development lifecycle. In this paper we highlight the importance of having development environments for component composition. Currently, software components are developed under different standards, therefore introducing the challenge of heterogeneity at the time of composition. We propose research to define composition within an environment of heterogeneous software artifacts. We then propose a strategy to handle the research. As a first step towards solving the research problem, this paper concentrates on defining a software component and outlines our future work.

Introduction

The ever growing need to produce larger complex systems in the shortest possible time, at the lowest cost, has promoted a shift to paradigms such as component-based software engineering. It is important that all software engineering stakeholders promote such paradigms in a bid to give the customer the best of service. This widely accepted fact serves as a foundation for our research.

Motivations

Component-based software development (CBD) advocates the use of existing artifacts of the software development process to build systems. This does not exclude any artifact of the software development process. In fact, reuse in the early lifecycle is claimed to have higher potential pay off in terms of quality and productivity (Ali and Du, 2003). However, most current efforts are towards the use of existing artifacts of the implementation activity of the software development process. This claim is supported by results of a survey we did of a number of software repositories (Guo and Luqi, 2000; Kung-Kiu and Zheng, 2005; Lee et al., 2003; MacroVista; LogicLibrary; SourceBook; SourceForge; Tarvar) to find out the type of artifacts which they store. We randomly sampled repositories which store one or more software artifacts irrespective of what type of artifacts they store. We studied the types of artifacts each repository stores. The types of artifacts have been generalized based on the generic activities of the software development lifecycle. Table 1, shows the results of our survey.
Table 1: Types of Artifacts Stored in Different Repositories

<table>
<thead>
<tr>
<th>Repository</th>
<th>Stored Software Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Requirements</td>
</tr>
<tr>
<td>+1 House</td>
<td>✓</td>
</tr>
<tr>
<td>SALMS</td>
<td></td>
</tr>
<tr>
<td>ARS</td>
<td></td>
</tr>
<tr>
<td>DSRS</td>
<td></td>
</tr>
<tr>
<td>WSRD</td>
<td></td>
</tr>
<tr>
<td>PAL</td>
<td></td>
</tr>
<tr>
<td>Logix</td>
<td></td>
</tr>
<tr>
<td>AFSDRS</td>
<td></td>
</tr>
<tr>
<td>SourceForge</td>
<td></td>
</tr>
<tr>
<td>CodeVista</td>
<td>✓</td>
</tr>
<tr>
<td>VCL Code Library</td>
<td></td>
</tr>
<tr>
<td>PHP Code Snippet Library</td>
<td></td>
</tr>
<tr>
<td>ToolBox in BDK</td>
<td></td>
</tr>
<tr>
<td>JTEE Server</td>
<td></td>
</tr>
<tr>
<td>COM Server</td>
<td></td>
</tr>
<tr>
<td>CMN Server</td>
<td></td>
</tr>
<tr>
<td>SOFA Template Repository</td>
<td></td>
</tr>
<tr>
<td>Koboa components file system</td>
<td></td>
</tr>
</tbody>
</table>


From table 1, it is clear that only 5 of the 19 repositories surveyed, store artifacts of the whole software development process. Only 2 of the surveyed repositories do not store artifacts of the implementation activities of the software development process.

Early reuse has various challenges as advanced by (Ali and Du, 2003; Cybulski et al., 1998; Li and Van Katwijk, 1992; Rubin, 1990) One key challenge of early reuse is the lack of tools and reuse-friendly environments needed to foster reuse of the early artifacts. This motivates our research to create a component based development environment.

The Software Supermarket (SoS)
The SoS is a bigger research aiming at providing a “one stop shop” for component
Based software development. That is a place to obtain components, compose them and/or obtain systems assembled from various components. It is therefore an appropriate case for validating and verifying the results of our proposed research on composition. We therefore introduce it in this paper before we discuss the proposed research.

As a result of the activities to build the SoS, a heterogeneous repository to house the various components has been developed (Pyne and Mugisa, 2004). We look at heterogeneity in this case, from two perspectives. In the first perspective, the repository houses all artifacts of the software development process. Secondly, the repository houses artifacts which have been built under different standards. Essential elements that a development environment needs to support the said repository, were identified by Pyne and Mugisa, (2004) as:

- A facility for component composition.
- A mechanism for ensuring that each component used is properly self-described.
- A facility for component adaptation.
- Support for heterogeneous software repositories.
- Support for all life cycle activities.
- Mechanism for retrieval of components from the repository based on behavior specification.
- Platform independent target environment.
- Collaboration with third party repositories.
- Cooperation with other development environments.
- Scalability motivated by changing user requirements.
- Support for developers to create new components.
- Support for application testing.
- Support for access to remote repository.
- A formal specification tool.
- A simulation environment.
- A code generation facility.
- Graphical user interfacing (GUI).

It was noted that no current environment satisfies all the identified elements. One of the elements is a facility for composition i.e. a facility to enable the composition of deployable components in assembling a new application or maintaining an existing one. Most of the existing environments have a facility for component composition. However, most of the environments support homogeneous repositories. This research therefore generally aims to define composition within an environment of heterogeneous software artifacts.
Approach to the Research
Composition of software components can occur both at the design and deployment phases of the life cycle of a component (Kung-Kiu and Zheng, 2006). Therefore we need to formally define how components can be composed into composites and deposited back into our repository for future retrieval. On the other hand we need to formally define how component binaries can be composed in order to realize an executable system. In order to address composition challenges, we need to address static and dynamic aspects of composition as well (Kiziltan et al., 2000). In both these aspects we are talking about connection-oriented activities. We therefore intend to use a model that supports connectors for component composition. We find the Abstract Behavior Type (ABT) model (Arbab, 2005; 2006) appropriate to define connectors that will assist in:

- Binding provided and required interfaces of components to be composed,
- Handling non-functional requirements that result from interaction of components.

The ABT model supports exogenous coordination of components which in turn reduces coupling problems, therefore being more suitable for composition of our heterogeneous components. Before we embark on the task of formal definition of software composition, we have to define a software component. Our main reason for this action is that Component-Based Software Engineering lacks universal definitions for some of its core concepts like ‘a software component’.

What is a Software Component?
There are as many definitions of a software component as there are component users. An interesting discussion of various definitions is given in Broy et.al: (1998). A review of relevant literature gives some of the numerous definitions as below:

1. A component 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. This definition is widely adopted (Szyperski et al., 2002).

2. A 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 (Heinman and Councill, 2001).

3. A component is a software element (modular unit) satisfying the following conditions (Meyer, 2003):
   (a) It can be used by other software elements, its 'clients'.
   (b) It possesses an official usage description, which is sufficient for a client author to use it.
   (c) It is not tied to any fixed set of clients.

4. A component is a”static abstraction with plugs”. ”Static” because a software component is a long-lived entity that can be stored in a software base,
independently of the applications in which it has been used. “Abstraction” in the definition means that a component puts more or less an opaque boundary around the software it encapsulates. “With plugs” means that there are well defined ways to interact and communicate with the component (parameters, ports, messages, e.t.c) Nierstranz and Dami, (1995).

5. A component is a deployable, independent unit of software that is completely defined and accessed through a set of interfaces (Sommerville, 2004).

6. A component is a language neutral independently implemented package of software services, delivered in an encapsulated and replaceable container, accessed via one or more published interfaces (Sparling, 2000).

7. A software component is a physical packaging of executable software with a well-defined and published interface (Hopkins, 2000).

8. Brereton and Budgen, (2000) describes software components as units of independent production, acquisition and deployment that interact to form a functional system.

9. D’Souza and Wills, (1999) define a component as a reusable part of software, which is independently developed, and can be brought together with other components to build larger units. It maybe adapted but may not be modified.

The definitions above are clearly different. All the definitions except D’Souza and Wills (1999)’s definition focus on the implementation activity of the software development process. Since reuse should cut across the software development lifecycle, it is important that we define an all encompassing component. We propose a definition for the software component that is closer to definitions by Szyperski et al. (2002), D’souza and Wills (1999). It accommodates all artifacts of the software development process but emphasizes essential characteristics of a component that enable a component to successfully participate in the process of component assembly to produce a system. Before we give the proposed definition, it is important to note that the definitions above agree that a component:

i) Is a software element. That is, a component contains sequences of abstract program statements that describe computations that are performed by a machine (a von Neumann computing device) Heinman and Councill, (2001). In the past, it was not clear whether requirements specifications and documentation qualified to be software elements especially since they were only informally specified. The current trend towards formal specifications has solved the said challenge. Further still documentation is now a breakdown of the system reference, system guide, technical reference and technical guide, which can all be provided for as sequences of abstract program statements that describe computations that are performed by a machine.

ii) Exports and imports services. A component requests for services from its environment and in turn provides services to its environment.
iii) Is able to interact and communicate with other components. That is, a component has interfaces which clearly define how it can interact with other components

iv) Is a unit of composition. That is it is designed with the ability of being used as a part of a whole. It is designed to be used in a compositional way with other components.

**Definition:**

A software component is a reusable artifact of the software development process that provides part of the services that are required to build a software system. The software element should:

(a) Export and/or import services.
(b) Have well defined interfaces
(c) Be a unit of composition.

Examples include: requirements specification, designs, patterns, architectures, test data, test plans, source code and documentation, a component that provides accounting services, a word processing component, a graphical diagram editor, a calculator component e.t.c.

**Conclusions And Future Work**

**Conclusion**

We have stated the need to support composition of components from a heterogeneous repository of software components. We have gone on to make a plan for addressing the composition challenge. Before dealing with composition, we have defined a software component. We have noticed (from literature) that there is more effort on reusing artifacts of the implementation activity of the software development process. Even as definitions for a component continue to emerge, the focus is more on composition and deployment. We have thus proposed a definition that shifts emphasis back to the general aim of CBSE which is to reuse artifacts of the whole of the software development process.

**Future Work**

A component in our context is now clearly defined. However in order to make the SoS a one stop shop, we expect to define composition of our components. Before we define composition, we shall define an abstraction for our components so that they all qualify for exogenous coordination at assembly time. One group of components to pay particular attention to, are the components developed using object-oriented techniques. Object-orientation supports endogenous coordination which we argue is appropriate for intra-component communication but not flexible for inter-component communication (not especially if the components come from different sources). Therefore the next step is to first address abstraction of object-oriented components.
References


