Managing Dependencies Between Software Products

Mark Northcott[1] and Mark Vigder[2]

[1] School of Computer Science, Carleton University,
1125 Colonel By Drive, Ottawa, Ontario, K1S 5B6, Canada
mnorthcott@rogers.com
[2] National Research Council of Canada,
1200 Montreal Rd., Ottawa, Ontario, K1A 0R6, Canada
Mark.Vigder@nrc-cnrc.gc.ca

Abstract. Systems constructed from diverse software products are often difficult to assemble and deploy correctly, particularly as the products evolve and the underlying platform changes over time. Many of these problems arise because of the many assumptions and dependencies, often implicit, that software products make about the context in which they are deployed. This paper describes an approach to managing the dependencies between the software elements of a system during assembly and deployment. A formal model of dependencies is developed, and it is shown how the model can be applied during the deployment process to verify the correct assembly of a system. The approach is designed to allow system developers, assemblers, and deployers to be part of the user group that collectively manages the dependencies that exist within an assembly.

1 Introduction

Assembling an application from software products is a non-trivial process. Software products may originate from different sources and evolve independently. The dependencies between products involve many types of constraints, and deploying the application to a particular platform introduces additional constraints. As the products and platform evolve, so too do the constraints, and the maintenance of constraints becomes increasingly difficult [6,7,8].

In current practice, developers of software products are directly responsible for defining the dependencies and constraints of the product. However, developers may not know a priori all of the contexts in which the software product will be used. Therefore, it is impossible to know all possible constraints of the product. Users of the software product, whether they be developers integrating the product into an application or end users deploying the product to a particular platform, are likely to encounter undocumented dependencies and constraints [7].

Currently, these newly discovered constraints may be documented through user groups, mailing lists or bug reports. However, it is often a tedious process investigating solutions to problems using any of these methods, especially for new users that may feel intimidated by the expertise of more experienced users. An approach for managing constraints that is accessible by both developers and users

X. Franch and D. Port (Eds.): ICCBSS 2005, LNCS 3412, pp. 201 – 211, 2005.

would aid in alleviating this problem. This will allow for newly discovered constraints to be properly documented by either developers or users, and it will provide a method for automatically verifying a particular deployment of an application or system.

This paper introduces such an approach for managing dependencies between software products that are assembled into a system. It does so by first presenting a formal model to describe system assemblies and the dependencies and constraints that exist between the elements of the assembly. Using this formal model, the activities associated with developing and deploying systems can be augmented by allowing constraints to be specified by any actor involved in the development and deployment process. Moreover, it is possible to specify the constraints in a fashion that can be automatically verified during the deployment process.

Section 2 describes a model for defining dependencies between software products. Section 3 explains how the model is used in practice in order to model a software system and specify constraints that may be automatically verified. An example application of the model is given in Section 4, which demonstrates how the open source xPetstore application is deployed to two different environments. Finally, Section 5 provides a discussion as to how the proposed model compares with current systems for managing dependencies between software products.

2 Modeling Dependencies Within Software Systems

In order to provide a tool for managing constraints, a suitable method for modeling dependencies between software products must be developed. This method should provide the following capabilities:

- Systems are developed in a hierarchical manner.
- Systems can be specified with *virtual components* that allow integrators to provide specific implementations of the component at deployment time.

In this context, a software product is viewed as a component that is a replaceable piece of a system that provides a clear function within the context of the system. Components could be COTS products, open source software, internally developed software, or any other large scale reusable piece of software.

Our model defines three types of components: concrete, virtual and composite.

- A ***concrete component*** implements services without being divisible into subcomponents. It is the most primitive type of component.
- A ***virtual component*** defines the services offered, but does not provide an implementation. It serves as an abstraction layer that allows for any component that provides the required services to be substituted into the system. This is similar to the concept of a virtual component as found in package managers such as in reference [1].
- A ***composite component*** is an assembly of subcomponents. The subcomponents of a composite component are not visible outside the composite component within which they are contained.