Integrating Configuration Management and Software Deployment

André van der Hoek
Institute for Software Research
University of California, Irvine
Irvine, CA 92612-3425
andre@ics.uci.edu

Abstract

As the development and use of component-based software becomes increasingly pervasive, the need arises for tools that support the controlled evolution of independently developed components—both at development and at deployment time. Traditionally, such tools have been separate: configuration management tools are used during the development of a software product and deployment tools are used to install, update, and reconfigure a software product once it is out in the field. This position paper first presents an argument why this separation is inadequate for component-based software. Then, it presents a set of requirements for an integrated system that intimately combines configuration management and software deployment functionality to support the evolution of component-based software in a unified manner.

1. Introduction

The increased popularity and staying power of component-based software cannot be denied. Major conferences such as the International Conference on Software Engineering (ICSE) now routinely involve a session devoted to the topic; open any kind of respected journal or magazine and one will find an article on the topic; and, most tellingly, powerhouses such as Microsoft and Sun have put a tremendous amount of resources behind their recently released component frameworks .NET and Enterprise Java Beans. With the exception of design tools and run-time "composition" tools, however, few tools are available to support the rather dramatic shift in focus from monolithic to component-based software. It is particularly concerning that two classes of tools that thus far have been among the few successful cornerstones of traditional software development, namely configuration management and software deployment, have not adapted to the new situation.

Our belief is that for component-based software to succeed, it is of paramount importance that these two classes of tools catch up quickly to provide the kinds of functionalities necessary to manage the evolution of component-based software, both at development and at deployment time. However, the traditional separation that has existed between the two is a concern for component-based software. The non-monolithic nature of component-based software requires installations, updates, and reconfigurations of "foreign" components within a development environment and the recording, management, and enforcement of valid configurations within a deployment environment. We, thus, argue that a two-way integration of configuration management tools with software deployment tools is required, in effect leading to a single, unified approach to adequately and uniformly capture and manage the evolution of component-based software.

In the remainder of this position paper, we first introduce briefly the fields of configuration management and software deployment. We then present a scenario of component-based software development that is the driver for our research and identify how this scenario highlights problems with the traditional separation of configuration management from software deployment. We then present our core requirements for an integrated configuration management and software deployment environment and conclude with a brief outlook at our future work.

2. Configuration Management

Configuration management [3] is a discipline that manages the evolution of a software product during its development phase. The two main functions a configuration management tool provides are the archival of different versions of evolving artifacts and the enforcement of collaboration policies to avoid (and resolve) as many development conflicts as possible. Leveraging this basic functionality, many advanced configuration management tools have been developed that each support their own particular storage formats, access methods, processes, and automated reports. As of late, the functionality in different configuration management tools is converging to almost full coverage of the spectrum of functionality identified by Dart [4]—demonstrating a mature discipline in which growth of functionality focuses on incremental features rather than radical new ideas.

3. Software Deployment

Software deployment is a discipline that manages the evolution of a software product after it has been devel-

oped; that is, it manages the process of installing, updating, reconfiguring, and adapting software at a customer site [6]. While the basic functionality is well-understood and clearly supported on individual platforms such as Windows or Linux (using tools such as InstallShield [7] and RPM [1], respectively), lacking is more advanced functionality that is able to deploy, to a set of distributed machines, complex systems consisting of multiple, interdependent parts. Research is currently underway in addressing these issues, but results still have to transition and more research is needed before the deployment problem in its fullest is addressed.

4. Scenario

The introduction of component-based software development introduces new concerns for the disciplines of configuration management and software deployment. Each is making progress towards addressing these issues (configuration management through an increased focus on the use of explicit system models and the use of vendor code management; software deployment through the introduction of agent-based deployment supporting explicit management of dependencies across development organizations). However, we argue that addressing configuration management or software deployment in isolation will not yield an adequate and satisfying solution. Consider the following scenario in which four independent organizations each develop components whose evolution, eventually, is surprisingly closely interrelated.

> *Organization AA is the market leader in developing document repository components; to keep competitive, its components are available in a number of different variants that each have some unique features. The organization operates in a very private manner: it does not share source code, only sells binaries, and develops the components according to its own release schedule.*
>
> *Organization BB is the anti-thesis of organization AA: in developing its highly advanced indexing and search component, it is very open to other organizations contributing to the functionality of that component. In fact, organization BB makes available the source code of its component and welcomes with open arms any patches that are submitted.*
>
> *Organization CC is a specialist in building components that facilitate the easy creation of Web sites. It offers a large set of components and is evolving those components at a rapid pace. Sometimes, the organization ignores the backwards compatibility of certain components for reasons of simplified technological advancements. Organization CC actually relies on components from a number of other organizations in the development of its components.*
>
> *Organization DD is in the business of providing Web-based indexing and search access to document repositories. The organization has purchased the relevant components from organizations AA, BB, and CC, adapted some of the components, and integrated them into a single software application that it sells. Organization DD regularly sends its patches to organization BB and has a contract with organization CC that allows it direct access to the source code repository of some of CC's components to jointly develop some of the necessary functionality. However, to some other components, it makes proprietary changes that it keeps to itself.*

This relatively small-scale scenario presents some unique challenges to the disciplines of configuration management and software deployment. In particular, it highlights a need for a two-way integration: as new component versions are released, software deployment needs to be present to bring these new versions into the configuration management environments of other organizations. Similarly, as those organizations make changes to the components they have obtained, these changes sometimes need to be fed back into the original development environment. Although existing tools augmented with a manual process can be used, a much more desirable and scalable situation is one in which the configuration management and software deployment tools are tightly integrated in a single system capable of handling the evolution of large-scale, distributed, component-based software at both development and deployment time.

5. Requirements

The development of an integrated configuration management and software deployment solution poses a number of unique requirements. In particular, we believe the use of different trust policies to express sharing and autonomy concerns lies at the heart of this kind of system. Traditional CM systems in essence hardwire a single such policy into their system, but the distributed and decentralized nature of component-based software development requires that privacy and autonomy concerns be raised in importance: trust levels need to be customizable for different situations and configurable for use at different levels among different sets of organizations. A basic catego-

rization of current trust levels for configuration management and software deployment is the following:

- **Copy binary**: only a deployment system is involved in obtaining a component, no trust is established among the developing organization and the deploying organization;
- **Copy source**: the deploying organization is trusted to obtain the source code and make a private copy to which it can make proprietary changes;
- **Patch sharing**: changes made by the deploying organization are trusted by the developing organization and, after inspection, incorporated in the official line of development of the component; and
- **Code sharing**: the deploying and developing organization intimately trust each other and share the same code base for certain components.

However, this categorization by far does not represent all aspects of trust that are present in component-based software development: it only deals with read and write access at development time. Other important aspects that will need to be expressed by trust policies involve such variables as when updates to deployed components are allowed, who is allowed to initiate and approve such updates (in essence: push versus pull), what kind of changes are allowed by the participating organizations, and, rather important, how transitive relations are handled. Moreover, it should be possible to specify different trust policies among the same organizations for different components: some components may be shared intimately whereas others may be regarded as too much of a competitive advantage to allow even the copying of source code.

Besides trust levels, we believe three other technologies are of paramount importance in the development of an integrated configuration management and software deployment system. Briefly:

- **Software architecture**: the availability of an explicit system model is essential in the proper management of the evolution of a component-based software system. Such a system model needs to incorporate facilities for the composition of a system out of other partial systems and components, for managing the versions and variants in which a component may exist, and for capturing behaviors and constraints such that they can be analyzed at composition time. A software architecture description language such as xADL 2.0 [5] provides these facilities and, thus, is an ideal platform to serve as a system model for component-based software.
- **Events**: Because of the decentralized nature of component-based software development and the high likelihood of participating organizations having different configuration management and trust policies, an integrated configuration management and software deployment system needs to be a loosely coupled component-based system in and of itself. As such, the use of events as the primary method of communication is important. Because of the fluxing nature of the federation of participating organizations, an event service such as Siena [2] is needed to keep all participating organizations abreast of what is happening in the federation—allowing the trust policies to react to events rather than be tightly integrated with each other.
- **Agents**: Since trust policies should be customizable and configurable, since different trust policies need to be enforced in different settings, and since different trust policies are authored by the developing organization but often carried out and enforced at the deploying organization, trust policies themselves need to be mobile. This fits well with the event-based infrastructure that is required. As demonstrated by the Software Dock [6] deployment system, an agent-based system combined with event-based communication supports the model of deployment of component-based systems that are authored by decentralized organizations well.

The move to component-based software development, thus, calls for an integrated configuration management and software deployment system that is heavily focused on the use of trust policies and based on the use of software architecture, events, and agents. This represents a rather significant departure from the current state of the art in especially configuration management (which currently does not use software architecture, events, or agents) but also in software deployment (which currently does not use software architecture). Much research is needed into producing the right (set of) integrated capabilities.

6. Conclusions

Imagine a world in which component-based software development has taken over: thousands of independent organizations are developing tens of thousands of interrelated components. In this world, configuration management and software deployment need to be intimately integrated in a two-way fashion: deployment functionality needs to be added to configuration management tools and configuration management functionality needs to be added to software deployment tools. An optimal solution, even, tightly integrates the two kinds of functionalities in a single system. As we have demonstrated in this paper, such integration brings with it significant challenges not addressed by the current state of the art.

We have begun the process of designing and building an integrated configuration management and software deployment system. Preliminary efforts have focused on developing the right architecture description language to be used (resulting in xADL 2.0). We are now shifting our

focus towards building actual configuration management and software deployment functionalities.

7. Acknowledgements

Effort sponsored by the Defense Advanced Research Projects Agency, Rome Laboratory, Air Force Materiel Command, USAF under agreement numbers F30602-00-2-0599 and F30602-00-2-0608. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory or the U.S. Government.

8. References

[1] Bailey, E.C., *Maximum RPM*. 1997, Red Hat Software Inc.

[2] Carzaniga, A., Rosenblum, D.S., and Wolf, A.L., *Design and Evaluation of a Wide-Area Event Notification Service.* ACM Transactions on Computer Systems, 2001.

[3] Conradi, R. and Westfechtel, B., *Version Models for Software Configuration Management.* ACM Computing Surveys, 1998. **30**(2): p. 232-282.

[4] Dart, S., *Spectrum of Functionality in Configuration Management Systems*. 1990, Software Engineering Institute: Pittsburgh, Pennsylvania.

[5] Dashofy, E.M., van der Hoek, A., and Taylor, R.N., *A Highly-Extensible, XML-Based Architecture Description Language*, in *Working IEEE/IFIP Conference on Software Architecture*. 2001 (to appear).

[6] Hall, R.S., Heimbigner, D.M., and Wolf, A.L., *A Cooperative Approach to Support Software Deployment Using the Software Dock*, in *Proceedings of the 1999 International Conference on Software Engineering*. 1999, ACM Press. p. 174-183.

[7] InstallShield Corp., *InstallShield*. 1998.