Component Configuration in Component-Based Systems

Parminder Kaur[1], Hardeep Singh[2]

ABSTRACT

The world of software development has rapidly changed in the last few years due to the adoption of component-based technologies. The classical software configuration management, which deals with source code versioning, becomes insufficient in the world where most components are distributed in a binary form. The task of configuration management is not well supported by conventional files, directories and ad-hoc persistence mechanisms. Typed, immutable objects combined with ubiquitous versioning provide a much more sound base. In this paper, an effort has been made to analyze the application of configuration management in component-based systems. This paper also summarizes that some level of configuration control can be achieved if components are identified with their version and dependencies to other components.

Key Words : Component, Component Configuration Management, Software Configuration Management, Configuration Model, Component-Based Systems

1. INTRODUCTION

A practical software system consists of hundreds or thousands of separate but related components. A software development environment (SDE) helps in managing this collection by providing configuration management services. Version management is associated with tracking the evolution of individual components. However, configurations naturally exist as versions, and effective management of such configurations distinguishes a truly useful SDE for large, multi-user projects.

An SDE typically relies on the persistent storage facility in the form of files and directories. The weak functionality of this storage facility is that it does not permit adequate modeling of configurations. The development in object-oriented database (OODB) technology provides persistent programming languages, which prove helpful in adequate modeling of configurations [1]. Object-Oriented Analysis and Design (OOAD) can be applied in the construction of Software Configuration Management (SCM) software. Example of such system can be found in [2].

In recent years, the focus of SCM research is on software process support in SCM systems, distributed configuration management systems and unified version models [3,4]. Component-based software development (CBSD) is an emerging paradigm in software development, which has changed the way we develop software greatly [5]. The aim of CBSD is to develop new software by widely reusing pre- fabricated software components. Component-based software development can be seen as an extension of object-orientation, but takes one step further. Object-Oriented Programming (OOP) binds the implementation to a particular class library and language. Components, on the other hand, are generally not bound to a particular language and they communicate through independent interfaces. This paper explores the

[1] Lecturer, Dept. of Computer Science and Engineering, Guru Nanak Dev University, Amritsar email: parminderkaur@yahoo.com

[2] Professor, Dept. of Computer Science and Engineering, Guru Nanak Dev University, Amritsar email: hardeep_gndu@rediffmail.com

applications of software configuration management (SCM) in object-oriented (OO) systems and component-based systems.

2. Software Configuration Management

Software Configuration Management (SCM) is the control of the evolution of complex systems. More pragmatically, it is the discipline that enables us to keep evolving software products under control, and thus contributes to satisfying quality and delay constraints.

SCM emerged as a discipline soon after the so called software crisis was identified, i.e. when it was understood that programming does not cover everything in Software Engineering (SE), and that other issues were hampering SE development, like architecture, building and evolution.

The importance of SCM has been widely recognized, as reflected in particular in the Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI) [6,7]. CMM defines levels of maturity in order to assess software development processes in organizations. SCM serves different needs [8] like:

- As a management support discipline, it is concerned with controlling changes to software products [9-11]. It covers functionalities such as identification of product components and their versions, change control (by establishing strict procedures to be followed when performing a change), status accounting (recording and reporting the status of components and change requests), and audit and review (quality assurance functions to preserve product consistency).
- As a development support discipline, it provides functions that assist developers in performing coordinated changes to software products [12]. To support developers, SCM is in charge of accurately recording the composition of versioned software products evolving into many revisions and variants, maintaining consistency between interdependent components, reconstructing previously recorded software configurations, building derived objects (compiled code and executables) from their sources (program text), and constructing new configurations based on descriptions of their properties.

SCM refers to configuration management processes and systems for managing the software development cycle. These processes further deal with dependencies between project components that are created and those that are generated between source files and executables. The features provided by SCM include:

- Planning, controlling and generating different product configurations.
- Communicating and controlling engineering changes and tracking their implementations.
- Synchronizing product development.
- Tracking problems backward to discover their origins and forward to locate which components have been corrupted.
- The most common tools use a check out / check in procedure to solve the change management, synchronization, and error tracking aspects of CM. Under this scheme, a project is divided up into units of logical functionality called modules. Most SCMs, commercial or free, use the host

file system for storing and comparing versions and define a module as a collection of files in a project subdirectory [3]. The goal of the configuration management is to support the precise specification of a software system that may occur in many versions and variants.

3. Component-Based Systems

Component - Based Software Development (CBSD) is one recent trend in the domain of Software Engineering (SE). One major reason why this paradigm has emerged is the need to build software by assembling reusable units, or components, as opposed to building whole application from scratch. The style and architecture of the applications being developed has significantly changed over the years. Software systems are also becoming increasingly complex and are providing ever increasing functionality. A set of infrastructure standards and supporting technologies like COM, COM+, .NET, CORBA, JavaBeans [13-15] has emerged during the last few years. In order to produce such systems cost-effectively, suppliers often use component-based technologies instead of developing all the parts of the system from scratch. The motivation behind the use of components was initially to reduce the cost of development, but it later became more important to reduce the time to market. By using components, it has become possible to produce more functionality with the same investment of time and money [16].

3.1. Component Configuration Management

In component-based systems, it is difficult to manage components during the lifetime of a system. A system of components is usually configured only during the build-time when known and tested versions of components are used. Later, when the system evolves with new versions of components, the system itself has no mechanism to detect if new components have been installed. There might be a check that the version of replacement component is at least the same as or newer than the original version. This approach prevents the system from using old components, but it does not guarantee its functionality when new components are installed. Using component configuration management, it can be possible to answer questions, such as:

- Which components have been added/removed after are configuration?
- Which dependencies have been added, removed or affected by a reconfiguration?
- If a component is updated, which other components in the system are affected?
- What is the effect on a system if a new system of component is installed?
- What is the difference between two configurations?

These are some of the difficult questions which, when answered correctly can give a better understanding of a system and permit a certain level of predictability when upgrading systems [17-19].

3.1.1. Identification Of Components In Use

In a component-based system, generally, there is a lack of information for identification of components used, their versions and history. There is no standard interface that can be used to gather sufficient information about the components to permit the creation of a dependency graph. Such a dependency graph is necessary to predict the effects of updating the system with new components.

While using configuration management, a component is identified by various parameters like name, creation time,

size, a version identifier and a unique number set by the compiler. The identification data is used to calculate a unique key to be used to compare components. The key is divided into two parts, one for identification, and the other for version. If no version information can be obtained from the component itself, it can be added manually using the embedding pattern [18][20].

3.1.2. Configuration Model for Components

A configuration model defines how components are treated and put under version control. A system of components can be treated as a baseline and be placed under version control. The components themselves are not put under version management but the unique key, which identifies them, is used as a representative. This means that if S is a set of keys, then for improved performance S is to be sorted to permit the easy location of components. This is to be considered as a prerequisite for comparing configurations by means of deductions from the dependency graph.

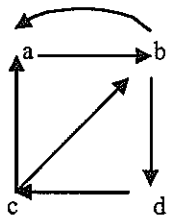
Grouping many different components with same functions can extend configuration model. In that case, it is possible to change components, if one of the components is non functional or is unavailable for any reason [21].

3.1.3. Change Management

The reasons for changes in components are multiple and complex. Changes can originate from many different sources. Change management handles all changes in a system. The reason for a change can be an error, improvement of the component or added functionality. Change management includes tools and processes which support the organization and track the changes from the origin to the actual source code. In configuration management, change management can be applied to both internal and external components. In the case of internal components, it is possible to use the same tools for change management as for the development of the component itself. External components can be placed under change management to permit the monitoring of changes and bugs, which occur. Instead of attaching source code files to change requests, the name of the component can be used to track changes [22, 23].

3.1.4. Managing Dependencies

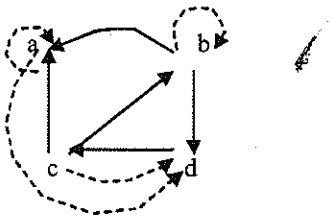
Dependencies between components can be computed and stored for further management. The benefits of this are multiple. It is possible to analyze what has been affected in the system and to create determinism when updating the system with new components. Dependencies between components can be represented with a directed graph.



(A graph with nodes a, b, c, d)

Figure shows an example of graph G = (V, E), in which V = {a, b, c, d} and E = {(a, b), (b, a), (b, d), (c, a), (c, b), (d, c)}

Paths are introduced to be able to define dependencies between components. An example of a path from a to d in above figure is <a, b, d> since each pair (a, b) and (b, d) is a part of the set of edges E. Knowing that there is a path from a to d indicates that a is dependent on d, since a is affected if d changes. A directed graph can be represented through matrices, lists and nodes with pointers to their children and parents. The transitive closure can be calculated to gather all the possible edges of a graph as shown in following figure. It is possible to gather all the dependencies for each of the different representations.



The graph in figure with the transitive closure calculated. The dashed lines are edges, which have been inserted in the graph.

Different algorithms can be used to calculate the transitive closure to obtain all paths in a graph. Warshall's is a simple well-known algorithm, which works on matrices [23].

3.1.5. Dependencies Between Components

As we know that components are the nodes in a graph and the dependencies are the edges. If dependencies are defined and described, it is possible to define the set of all dependencies as a set of dependency pairs.

Once dependencies have been calculated, it is possible to create a system structure, with different levels of components [24]. On the lowest level of components are components without dependencies to other component. This type of system structure is used as a model to calculate quality properties such as complexity and localization factors. The complexity is proportional to the number of dependencies between the components. The localization factor denotes the number of levels between components.

A configuration is a set of components and their dependencies to other components. The configuration is a baseline since it represents a version of a system at a particular time. Configurations are stored under version control for later retrieval. New installed components can be compared with a configuration to permit recognition of the affected components in the system. When new components are installed, new nodes in the dependency graph can be added. In the same way, nodes can be removed if components are removed.

Broken dependencies can be detected when the old configuration is compared with the new. New versions of an existing component can be identified by the version part of the unique key which identifies all components. New versions simply replace the older in the graph. When comparing graphs, new versions can be detected with the help different keys. For proper dependency analysis, it is required that a component and its version should be identified.

3.1.6. Differences Between Configurations

Differences between configurations can be computed and used to obtain a deeper understanding of what has been changed in the system. The dependency graph, which is represented by a matrix or by lists, can easily be used to determine what has been affected when a new version of a component has been installed. It is possible to obtain version history from different configurations by comparing different versions of the dependency graph [21][27].

4. Conclusion

In this paper, it is summarized that the objective of the configuration management is to support the precise specification of a software system that may occur in many versions and variants. It is possible to construct arbitrarily complex collections of components, connected in an OODB, much more easily as compared to file system. This paper also analyzes the requirements of configuration management in CBSD. By applying configuration management techniques to component based systems together with dependency analysis, it is possible to predict the effects of a component update.

Future directions of research include the need to find more effective techniques for integrity maintenance, constituent merging at syntax level and for logical versioning of constituents in component-based systems.

References

[1] Malcom P. Atkinson and O. Peter Buneman, *"Types and Persistence in Database Programming Languages"*, ACM Computing Surveys 19,2 105-190, June 1987.

[2] Mick Jordan and Michael .L. Van De Vanter, *"Software Configuration Management in an Object Oriented Database"*, In proceedings of the USENIX Conf. on Object-Oriented Technologies (COOTS), Monterey, CA, PP. 26-29, June 1995.

[3] R. Conradi and B. Westfechtel, *"Version Models for Software Configuration Management"*, Software Configuration Management Symposium, SCM-7, 1977, Springer, ISBN 3-540-63014-7, ACM Computing Surveys, Vol. 30, No. 2 .

[4] A. Zeller and G. Snelting, *"Unified Versioning through Feature Logic"*, ACM Transactions on Software Engineering and Methodology, 6(4):398-441, October 1997.

[5] M. Aoyama, *"Component-Based Software Engineering: Can it Change the Way of Software Development?"*, In Proceedings Vol. 2 of the 1998 International Conference on Software Engineering, April 1998.

[6] Paulk .M.C, Weber C.V, Curtis .B, Chrissis .M.B, *"The Capability Maturity Model—Guidelines for Improving the Software Process"*. Addison-Wesley, Reading, MA, 1997.

[7] Humphrey .W.S, *"Managing the Software Process"*, SEI Series in Software Engineering, Addison-Wesley, Reading, MA, 1989.

[8] Feiler .P.H, ED. 1991b. Proceedings of the Third International Workshop on Software Configuration Management (Trondheim, Norway, June), ACM Press, New York.

[9] Bersoff .E.H, Henderson .V.D and Siegel S.G, *"Software Configuration Management: An Investment in Product Integrity"*, Prentice-Hall, Englewood Cliffs, NJ, 1980.

[10] IEEE 1983. IEEE Standard for Software Configuration Management Plans: ANSI/IEEE Std 828-1983. IEEE, New York.

[11] IEEE 1988. IEEE Guide to Software Configuration Management: ANSI/IEEE Std 1042- 1987. IEEE, New York.

[12] Babich .W.A, *"Software Configuration Management"*, Addison-Wesley, Reading, MA.

[13] OMG, CORBA, http://www.omg.org/corba

[14] Microsoft COM, http://www.microsoft.com/com , http://www.microsoft.com/net/ http://msdn.microsoft.com/library/

[15] Sun Microsystems, JAVA, http://java.sun.com, http://java.sun.com/beans

[16] Workshop Proceedings Component – based Software Engineering: building systems from components, Technical report 2002-2005, Malardalen University, Sweden, 2002

[17] Cook .J.E and Dage .J.A, *"Highly Reliable Upgrading of Components"*, In Proceedings of 21st International Conference on Software Engineering, ACM Press, 1999.

[18] Hoek .A.V.D, *"Capturing Product Line Architectures"*, In Proceedings of 4th International Software Architecture Workshop, ACM Press, 2000.

[19] R. Bialek, *"The architecture of a dynamically updatable component-based system"*, COMPSAC, 2002.

[20] Larsson .M and Crnkovic .I, *"New Challenges for Configuration Management"*, In Proceedings of 9th Symposium on System Configuration Management, Lecture Notes in computer Science, nr 1675, Springer Verlag, 1999.

[21] Larsson .M and Crnkovic .I, *"Component Configuration Management"*, In Proceedings of 5th Workshop on Component Oriented Programming, 2000.

[22] Garlan .D, Allen .R and Ockerbloom .J, *"Architectural Mismatch: Why Reuse is so Hard"*, IEEE Software, Vol. 12, Issue .6, 1995.

[23] Eve .J and Kurki-Suonio .R, *"On computing the transitive closure of a relation"*, Acta-Informatica, Vol. 8, No. 4, 1977.

[24] Crnkovic .I, *"Large Scale Software System Management"*, Ph.D. Thesis, Department of Electrical Engineering, University of Zagreb, 1991.

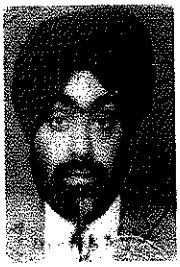
[25] Alexis Leon, *"Software Configuration Management Handbook"*, Second Edition, Artech House, London, 2005.

[26] Mario .E. Moreira, *"Software Configuration Management Implementation Roadmap"*, John Wiley & Sons Ltd. 2004.

[27] Parminder Kaur, Kuljit Kaur, Hardeep Singh, *"Configuration Management in Object-oriented Systems & Component-Based Systems"*, accepted in 16th Int'l conf. on Software Engineering & Data Engineering SEDE-07, Las Vegas, U.S.A. July 2007.

Author's Biography

Parminder Kaur, is working as lecturer, Deptt. of Computer Science & Engg., Guru Nanak Dev University, Amritsar. Her field of interest includes Software Engineering and Component–Based Development Systems. She has published twelve national/international papers in this area.



Dr. Hardeep Singh, is working as Professor, Deptt. of Computer Science & Engg, Guru Nanak Dev University, Amritsar. His areas of specialization are Software Engineering, Information Systems and Open Source Systems. He has published more than fifty papers in National/ International journals and conferences.