ROA MODULAR LDAP-BASED APPROACH TO INDUSTRIAL SOFTWARE REVISION CONTROL

Cristina De Castro
IEIIT-CNR, Italian National Research Council
V.le Risorgimento 2, Bologna 40136, Italy

Paolo Toppan
CNIT, Italian Inter-University Consortium for telecommunications,
V.le Risorgimento 2, Bologna 40136, Italy

Keywords: Software revision control, industrial context, plant installations, LDAP schemes for revision control.

Abstract: A software revision control system stores and manages successive, revised versions of applications, so that every design stage can be easily backtracked. In an industrial context, revision control concerns the evolution of software installed on complex systems and plants, where the need for revision is likely to arise from many different and correlated factors. In this paper, starting from the wide bibliography available on the subject, some typical schemes are discussed for representing such factors. An LDAP-based architecture is addressed for modelling and storing their evolution.

1 INTRODUCTION

The efficient management of applications' lifecycle is becoming more and more important and complex in industrial environments, where software is installed in large and often distributed plants. The overall process results from several different factors: for instance, a malfunctioning can arise from the incorrect interaction between an old software version running on an updated or modified machine; software results generally from the synergy of many different people, approaches and requirements, etc. The problem of revising software in industrial plants is thus more and more challenging and the critical issue of software documentation is bound to become more and more essential in the backtracking process. Versioning is widely used for the management of software development and revision (Westfechtel et al., 2001, Fischer et al, 2003), even if not always with a precise temporal semantics. This process, named *revision control*, allows to backtrack and compare every design stage. Far from expecting to solve this very difficult task, this paper discusses some simple, typical schemes for representing a software revision process and proposes an LDAP-based temporal architecture for their representation.

2 COMPONENTS OF AN INSTALLATION, EVOLUTION AND THE LDAP SCHEME

In an industrial context, software can not be considered an isolated issue. As a matter of fact, software is installed on a set of machines and has related configuration specifications. An *installation instance* (Fig. 1) can thus be defined as the set of these three groups of components and their interaction. In this scheme, the plant requirements and the documentation of software, hardware and installation features, detailing all the design phases and the interactions, are considered part of the installation instance. As a matter of fact, they play a fundamental role in the process of software development and revision. An installation instance represents the snapshot of the state of a plant after a generic installation or update, thus it represents a version in the process of industrial software revision control. Let us now consider the generic structure of an installation instance, one of its versions is the collection of the following *component versions*: (1) requirements version; (2) software version; (3) hardware version; (4) configuration files version; (5) documentation version.

De Castro C. and Toppan P. (2006).
ROA MODULAR LDAP-BASED APPROACH TO INDUSTRIAL SOFTWARE REVISION CONTROL.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 261-265

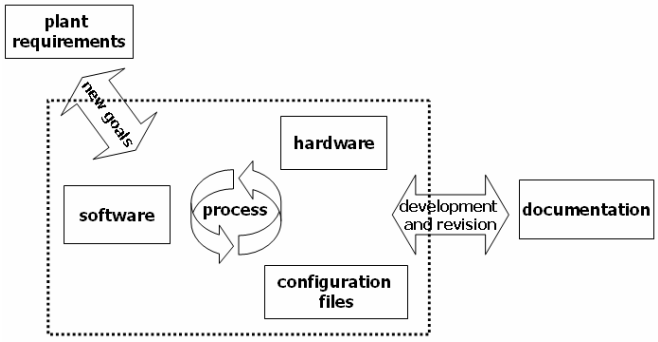

Figure 1: Installation instance.

In order to allow a component version to be shared and referred by many installation instances, each of the components can be labelled independently. As far as the type of temporal labelling is concerned, it is quite common to label the versions by means of progressive numbers, but the necessity can arise of referring the system to a precise temporal phase of its lifecycle. For instance, if a malfunctioning causes trouble to a client, it can be of great importance, even from a juridical point of view (Grandi *et al*, 2003, Harris *et al*, n.d.), to know precisely how the installation was when the problem occurred. The support of at least two kinds of time dimensions is widely emphasized in literature (Böhlen *et al,* 2006, Elmasri *et al*, 1990, Tansel *et al,* 1993): transaction time, which tells when an event is recorded or updated in a knowledge base, and valid time, which represents when an event occurs, occurred or is expected to occur in the real world. Transaction-time is represented by adding the endpoints IN, OUT and valid-time is represented by the endpoints FROM, TO. In the considered environment, transaction time tells when the installation version description discussed above was recorded in the revision control knowledge base and valid time refers to when the installation is operative in the plant. For instance, the installation version h in Fig. 2 was recorded in the knowledge base on May 3rd 2006 and it has not been updated yet, so IN = May 3rd 2006 and OUT = ∞. It has been fully operative on the plant FROM May 15th 2006 and is expected to be used until (TO) December 23rd 2007. Note that there is no need a priori that they should be labelled with the same values of transaction and valid-time. For instance, the software version j can have been recorded before May 3rd 2006 but used within the installation version h. In this way, components can be shared and duplication is partly avoided. For instance, the installation version g in the background in Fig. 2 shares software version j.

A further advantage of using this kind of timestamping is that, if a bug is detected within an installation version, all the other versions that may be touched by the same bug can be found in a very simple way by means of temporal queries.

This structure can be easily developed by means of the Lightweight Directory Access Protocol (LDAP). LDAP (Howes *et al*, 2003, Koutsonikola *et al*, 2004) can be particularly suitable for representing installation versions in wide enterprises: as a matter of fact, LDAP is widely used in enterprise databases and it is optimised for reading operations, being thus suitable for storing and managing temporal data that must be backtracked. Furthermore, LDAP schemes can be very easily changed and extended in order to record new attributes and new classes. This feature can be very useful when designing a revision control. As a matter of fact, new objects and new attributes are very likely to be modified or added due to new needs or improvements made by the people who develop it. LDAP was also built for distributed environments, so it suits the distributed location of industrial applications very well.

Referring to the schemes discussed above, the LDAP object-classes tree can be defined as follows (see Fig. 3, Tab. 1): the 0-level class describes the *installation* in general; the 1-level classes are respectively *plant requirements*, *software*, *hardware*, *configuration files* and *documentation*. It must be observed that the timespan of the different objects can have different meanings. For instance, the temporal attributes of class “software” have the following meaning: 1) IN, OUT: when the software version was recorded/updated in the revision information system; 2) FROM, TO: the period in which the software version is/was/will be operative.

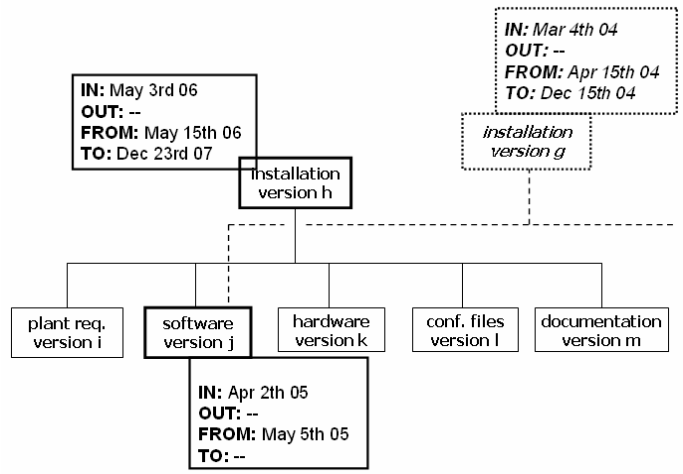

Figure 2: Timestamps of an Installation Version and its components.

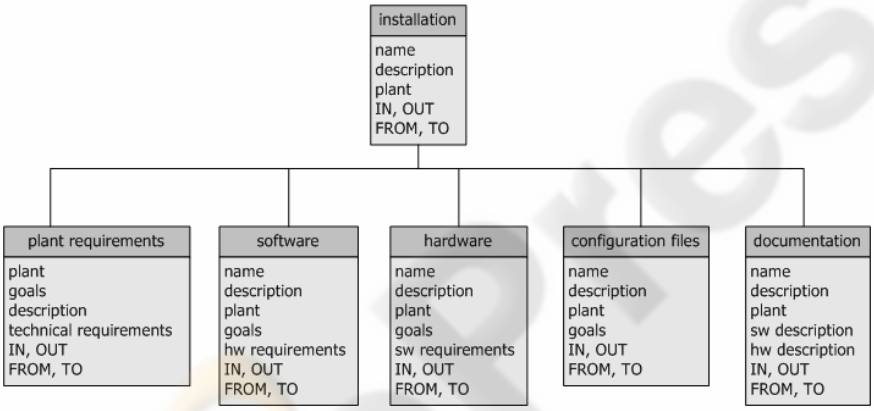

Figure 3: An LDAP basic structure for the representation of installation versions.

3 CONCLUSIONS

In this paper, a basic architecture was discussed for representing the temporal evolution of a software installation on an industrial plant. This model divides an installation in many interacting components and, making use of the LDAP model, represents them in a hierarchy. The evolution of the system is represented by means of
both transaction-time and valid-time. Future work will be devoted to deepen the semantics and the interaction mechanisms of the components of this model, in order to merge it in a revision control system.

ACKNOWLEDGEMENTS

The authors want to thank the InSeBaLa project of Regione Emilia Romagna (Italy) for supporting this work.

Table 1: Semantics of classes and attributes.

Class	Attributes	Description
installation	oid, name, description	
	plant	the specific industrial plant where the installation was made
	IN, OUT	when the installation version was recorded/updated in the revision information system
	FROM, TO	the period in which the installation version is/was/will be fully operative on the plant
plant requirements	oid , plant, goals (multivalued), description, technical requirements (multivalued)	
	….	
	IN, OUT	when the requirements version were recorded/updated in the revision information system
	FROM, TO	the period in which the requirements are/were/will be the actual reference of the plant
software	oid, name, description, plant , goals (multivalued)	
	hardware requirements (multivalued)	
	….	it can refer to the hardware version/s of another/others installation version/s
	IN, OUT	when the software version was recorded/updated in the revision information system
	FROM, TO	the period in which the software version is/was/will be operative on the plant
hardware	oid, name, description, plant, goals (multivalued),	
	software requirements (multivalued)	it can refer to the software version/s of another/other installation version/s
	….	
	IN, OUT	when the hardware version was recorded/updated in the revision information system
	FROM, TO	the period in which the hardware version is/was/will be operative on the plant
configuration files	oid name, description, plant, goals (multivalued) IN, OUT, FROM, TO	similar semantics
documentation	oid , plant, description, goals, …	
	software description, hardware description	it can refer to the software version/s of another/other installation version/s
	IN, OUT, FROM, TO	….

REFERENCES

Böhlen, M., Gamper, J., Jensen, C.S. (2006). "Multi-dimensional aggregation for temporal data". *Proc. of EDBT-2006*, pp. 257-275.

Elmasri, R., El-Assal, I., Kouramajian, V. (1990). "Semantics of Temporal Data in an Extended ER Model". *Proc. Intl. Conf. on E-R Approach,* pp. 52-61

Fischer, M., Pinzger, M., Gall, H. (2003). "Populating a Release History Database from Version Control and Bug Tracking Systems. *Proc. of 19th IEEE International Conference on Software Maintenance (ICSM'03)*, pp. 23-29.

Grandi, F., Mandreoli, F., Tiberio, P., Bergonzini, M. (2003). "A Temporal Data Model and Management System for Normative Texts in XML Format". *Proc. of WIDM'03*.

Harris, C., Allen, R. B., Plaisant, C., Shneiderman, B. (n.d.). "Temporal Visualization for Legal Case Histories". *http://hcil.cs.umd.edu/trs/99-18/99-18.html*

Howes, T. A., Smith, M. C., Good, G. S. (2003). *Understanding and Deploying LDAP Directory Services*, Addison Wesley 2003, 2^ ed.

Koutsonikola, V., Vakali, A. (2004). "LDAP: framework, practices, and trends", *IEEE Internet Computing*, Vol. 8, Issue 5, Pages: 66 - 72

Tansel, A., Snodgrass, R.T., Clifford, J., Gadia, S.K., Segev, A. (eds.) (1993). *Temporal Databases, Theory, Design and Implementation*. Benjamin-Cummings.

Westfechtel, B., Munch, B. P., Conradi, R. (2001). "A Layered Architecture for Uniform Version Management", *IEEE Transactions on Software Engineering*, Vol. 27, N. 12