Dedal-CDL: Modeling First-class Architectural Changes in Dedal

Huaxi Yulin Zhang, Christelle Urtado, Sylvain Vauttier, Lei Zhang, Marianne Huchard, Bernard Coulette

Dedal-CDL: Modeling First-class Architectural Changes in Dedal

Huaxi (Yulin) Zhang*, Christelle Urtado†, Sylvain Vauttier†, Lei Zhang‡, Marianne Huchard§ and Bernard Coulette*
*Dept. Math/Info, Université Toulouse 2, Toulouse, France
Email: Zhang,Coulette@irit.fr
†LGI2P / Ecole des Mines d'Alès, Nîmes, France
Email: Christelle.Urtado, Sylvain.Vauttier@mines-ales.fr
‡Research Center of Automation, Northeastern University, Shenyang, China
Email: zl.org.cn@gmail.com
§LIRMM, UMR 5506, CNRS and Univ. Montpellier 2, Montpellier, France
Email: huchard@lirmm.fr

***Abstract*—In component-based software engineering, software architectures govern not only software development but also software evolution. Indeed, to efficiently and accurately manage software evolution and guarantee its quality, architecture models should be at the core of the evolution process, be accurately synchronized with the runtime systems and have their changes and version information be completely tracked. As architecture models are often captured by ADLs (Architecture Description Languages), an ADL supporting architecture-centric evolution is required. In this paper, we study how architecture-centric evolution can be supported by the Dedal ADL. We thus propose a dedicated CDL (Change Description Language) which models architectural changes as first-class entities and describes them from a semantic viewpoint.**

I. Introduction

Component-based software development (CBSD) has been thoroughly studied lately, but there has been less effort to study component-based software evolution. Architecture are "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [1]. Architecture-centric evolution can both ease software evolution and make it more controllable than code-based evolution by shifting abstraction levels. For example, updating a software component to use a newly available library can be easily achieved by changing its architectural connections so that it is linked to the new library. Focusing on high abstraction levels prevents developers from having to delve into low-level component source code, which is harder to understand and modify [2].

To efficiently and accurately manage software evolution and its quality, architecture models must be the core of the evolution process, be accurately synchronized with the runtime system, and have their changes and version information be completely tracked [3]. Software evolution can be triggered from any architectural abstraction level of component-based software as modeled in our Dedal ADL (architecture specification, architecture configuration or component assembly). A change initiated on a given level might impact others in a process called architectural co-evolution [3]. To track and monitor changes during architecture model co-evolution, these architectural changes firstly need to be explicitly expressed in the ADL, as first-class entities.

In this paper, we introduce a CDL (Change Description Language) dedicated to Dedal which models architectural changes as first-class entities and describes them from a semantic viewpoint. A version model is also proposed for software architectures in Dedal that enables to track version history at all levels.

The remaining of this paper is organized as follows. Section II presents the context of software architectures in component-based software development and evolution, and describes the example used in the paper. Section III presents how evolution is expressed in Dedal: its CDL and version model. Section IV discusses existing ADLs how they support architecture evolution expressions. Section V concludes with future work directions.

II. Software Architectures in CBSD

In this section, we present the context of software architectures from both the development and evolution viewpoints. At last, an illustrative example to be used throughout the paper is introduced.

A. Software Architecture Description in Dedal

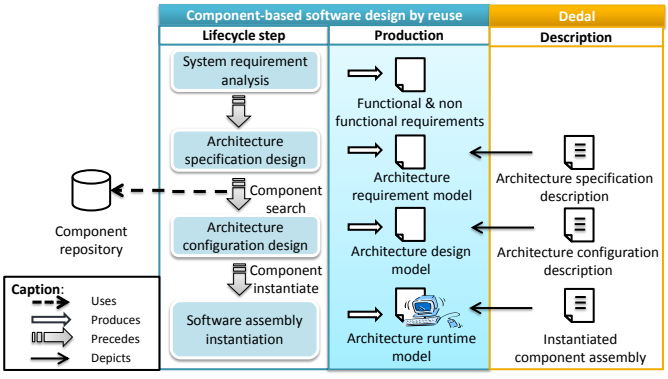

Fig. 1. Component-based software development process

CBSD is characterized by its implementation of the "reuse in the large" principle. Reusing existing (off-the-shelf) software components therefore becomes the central concern during development. In the context of CBSD, traditional software development life-cycles have to be adapted [4], [5]. Fig. 1 illustrates our vision of such a development life-cycle, with the supporting Dedal ADL [6].

Our CBSD life-cycle deliberately focuses on the produced artifacts: *architecture models* for each development step. Three kinds of architecture models are produced:

1) *Architecture specification.* After a classical requirement analysis step, architects establish the *abstract* architecture requirement model – architecture specification, which are composed by abstract component types (as-wished components) called *component roles*.
2) *Architecture configuration.* In a second step, architects design the concrete architecture model, called architecture configuration. To do so, he searches the component repository and selects *component classes* that implement the component roles that come from the specification, and connect them with *connector classes*. These component classes now are concrete component types (as-found components).
3) *Instantiated component assembly.* In a third step, configurations are instantiated into runtime models, called instantiated component assemblies, and deployed to executable software applications. Instantiated component assemblies are composed by *component and connector instances* that are instantiated from the component and connector classes.

B. Software Evolution in CBSD

Software system maintenance is a major phase in their life-cycle as it is both technically challenging and costly relatively to development. Indeed, software systems are always required to evolve after their release, in order to correct bugs, improve their performance, enhance their functionality, etc. Software evolution is also tightly linked to software architectures. The evolution that concerns a collection of software architectural activities to change a software from its old version to a new version and is activated by architecture changes is said to be *architecture-centric* [7]. During architecture-centric evolution, the architecture of a system can be modified at each of its levels. Thus, as a prerequisite of architecture- centric evolution, the applied architecture changes should be described together with software architectures themselves, in ADLs, which gather the important information on software architectures.

C. Example of a Bicycle Rental System

Fig. 2 shows the example used throughout the paper: the architecture specification of a bicycle rental system (BRS). A *BikerGUI* component manages a user interface. It cooperates with a *Session* component which handles user commands. The *Session* component in turn cooperates with the *Account* and *Bike&Course* components to identify the user, check the balance of its account, assign him an available bike and then calculate the price of the trip when the rent bike is returned. In the following sections, we will use a part of this system (*BikeCourse* and *BikeCourseDB*) to illustrate our concepts.

The left part of Fig. 3 presents an architecture configuration implementing the above architecture specification. In this example, architecture changes are initiated at the architecture configuration level, with the objective to transform the original configuration from left to right by adding the *StationData* component class.

III. EVOLUTION DESCRIPTION EXTENDED DEDAL

This section presents how Dedal is extended to integrate evolution description. This extension has two facets: architecture change descriptions with our CDL and architecture version tracking.

A. Modeling Changes with a Change Description Language

Based on Dedal, we propose a dedicated CDL (Change Description Language) to describe architecture changes. Change descriptions should capture the different facets of software changes that affect architecture evolution decisions. CDLs resemble ADLs, but from the viewpoint of architecture changes. Our Dedal-CDL is designed to specifically model software architecture changes by:

1) describing changes using the same syntax as that of the ADL,
2) treating changes as first-class entities in architecture descriptions,
3) modeling changes from a semantic (declarative) view rather than from an operational (procedural) one.

Using Dedal-CDL, software evolution can be tracked through all software architecture models. Architecture changes become explicit. Quality of software architectures can be evaluated.

In order to model architecture changes from the semantic viewpoint, the characteristics of changes should be part of the description language. Various information can be used to characterize architecture changes. To identify meaningful characteristic among those from the state-of-the art, we used a simple heuristic proposed by Buckley *et al.* [8] [1]. According to this strategy, we identified seven orthogonal characteristics of architecture changes: *time of change*, *anticipation*, *change type*, *change operation*, *change purpose*, *subject of change* and *nature of change*. These and their possible values are collected in Table. I.

1) Time of change: Based on *when* changes happen, three categories of changes can be deduced: static, load-time and dynamic [8]. As far as architecture evolution is concerned, changes are often either static or dynamic. Static changes can produce architecture pendency [7]. Unmanaged dynamic changes often result in architecture drift or erosion [9][2].

[1]This heuristic recommends to put the characteristic in a simple sentence of the form: "The change is <characteristic>".

[2]Architecture pendency, erosion and drift denote possible mismatches between the different architecture levels. This occurs when either of them evolves without its changes being propagated to other levels.

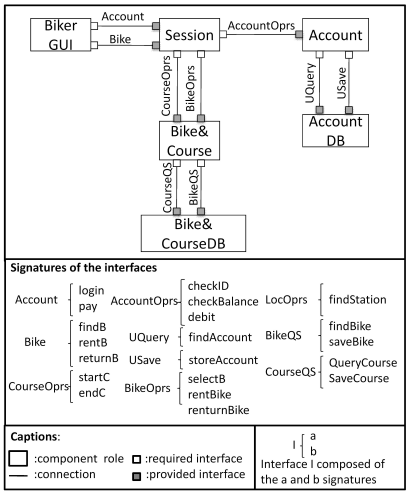

Fig. 2. BRS abstract architecture specification

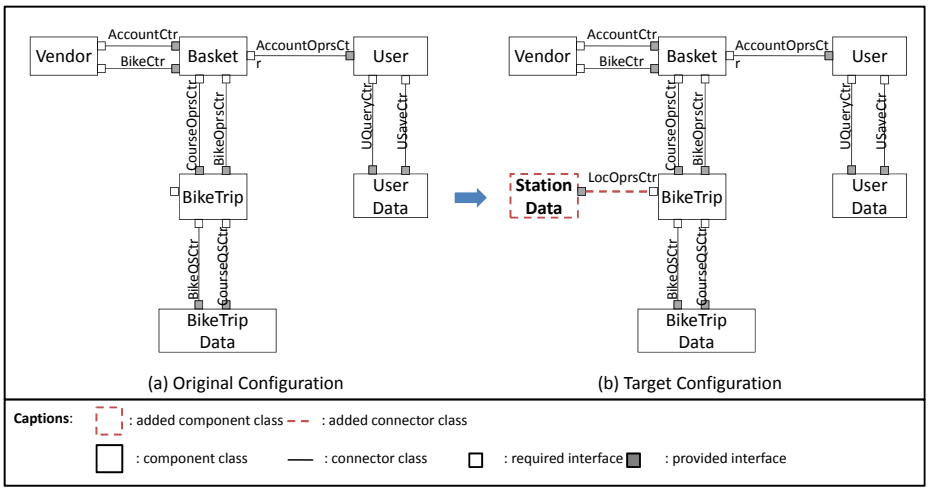

Fig. 3. BRS architecture configuration

TABLE I
CHANGE CHARACTERISTIC NAMES AND VALUES IN CDL

Change Characteristics	Values
time of change	static, dynamic
anticipation	anticipated, unanticipated
affected level	specification, configuration, assembly
change operation	addition, removal, substitution, modification
subject of change	elements of architectures in three levels
change purpose	corrective, perfective, adaptive
nature of change	given, generated, propagated[a]
a	If generated or propagated, using **from** indicates the original change

2) Anticipation: This characteristics refers to the time when the change model requires software changes to be foreseen. Two types of changes from this viewpoint: anticipated and unanticipated changes [10]. When they concern architectures, *anticipated changes* are changes that can be foreseen during the initial development of the system: optional components are predefined that are to be deployed into the system as needed. On the contrary, *unanticipated changes* cannot be foreseen during development: they arise later and are to be reacted to as unpredictable components are added or deleted.

3) Level of Change: With Dedal, software architectures are described at three levels: specification, configuration and assembly. Changes can be initiated on any of these description levels. Changes initiated at a given level can affect any of the two others. Knowing where some change comes from thus impacts how the change is going to be propagated among architecture levels.

4) Change Type and Operation: Change type refers to how the change affects a software. Architecture changes might affect a software's *semantics* or its *structure*. Semantic changes are confined to the interior of a component or a connector. They sometimes correspond to replacing one or more components or connectors by their newer versions, or modifying the architecture behavior description. Structural changes alter the structure of the configuration (the graph formed by components and connectors) by adding or removing a component or a connector. Change types can further be refined into different architecture change operations *i.e., addition*, *removal*, *substitution*, or *modification* of architecture elements.

5) Subject of change: Artifacts subject to change (*what changes?*) in a software system's architecture are its building blocks, such as components, connectors or connections (bindings) between them. At different architecture levels, these elements take specific forms. Sujects of change are *component roles*, *connections* and *architecture behavior* in specifications; *component classes* and *connector classes* in configurations; and *component instances*, *connector instances* and *assembly constraints*[3] in assemblies.

6) Change purpose: According to the ISO/IEC 14764 Standard for Software Maintenance [11], change purposes can be categorized into four types, depending on what motivates the changes: corrective, perfective, adaptive and preventive. For component substitution, change purpose can be used as a test criterion to evaluate whether a new version meets the change request. We adapt their definitions to architecture changes:

- *Corrective changes* mean that the changed architecture corrects errors / bugs identified in the old one.
- *Perfective changes* mean that the changed software alters the code of a component or connector to improve its non-functional qualities (such as performance).
- *Adaptive changes* adapt components or connectors to fit changes in either their environment or requirements.
- *Preventive changes* are undertaken to prevent side-effects of other changes. For example, adding a new component often requires that its connected component be adapted.

7) Nature of change: The nature of change represents the situation indicating how the change is triggered by being *given*, *generated* or *propagated*.

- *Given changes*. When a change is prescribed by architects or maintenance engineers, it is a *given* change.
- *Generated changes*. When a change is generated from and triggered by a given change to complete its action in

[3]These three levels of architecture descriptions and the elements each level can be composed of are defined in Dedal [6].

the same architecture level, it is defined as a *generated* change. For example, the additions of a new component role will cause one or more addition of connections between the new added component role and the existing component roles.

- *Propagated changes*. When a change is created because of and propagated from a given or generated change in a different level, it is said to be a *propagated* change. For example, the addition of a component class in a configuration can imply the addition of a component role, propagating bottom-up to the specification, or the addition of a component instance, propagating top-down to the assembly.

For generated and propagated changes, the source of the change should be indicated, using the *from* key word (*see* Fig. 4).

Among these characteristics of changes, some constraints apply. Firstly, change levels and subject of change should be coherent: the subject of change must be included in the level where the change initiated. For example, in the specification level, change artifacts are either component roles, connections or architecture behaviors. Secondly, the different artifacts subject to change have their own permitted operations. For example architecture behaviors can only be modified.

Based on the above seven characteristics of architecture changes, Dedal-CDL defines its architecture change model as shown in Fig. 4. Its syntax is detailed in [7]. Fig. 4 also gives an example of change description: the *additionStationData* change adds the *StationData* component class in the BRS configuration.

```
change::=
 change change_name
 time change_time
 anticipation
    change_anticipation
 level initial_level
 operation change_operation
 subject element
    is element_name
 purpose change_purpose
 nature change_nature
     ( from change )?
```

```
change additionStationData
 time dynamic
 anticipation unanticipated
 level configuration
 operation addition
 subject component_class
    is StationData
 purpose perfective
 nature given
```

Fig. 4. Syntax of the CDL (left) and a change description (right)

B. Version Model

Dedal integrate versioning capabilities to its three architecture levels. Only components can be versioned (versioning connectors is a perspective). Versioning architectures as wholes can help track their evolution over time. The reason why we chose to version only component classes is twofold. First, our work targets to component-based software where connectors are often automatically generated and do not contain specific information. Second, versioning components can aid and ensure the evolution test [12] by explicitly tracking all differences between two successive versions. For example, version 2.0 is modified to perform better than version 1.0.

Versioning architectural elements requires to record both the *versionID* and the identity of the previous version (*pre_version*). Pre_version's value is composed by the predecessor's version name and versionID. If a version's name is identical to its predecessor's, the name can be omitted as shown in the example on the right of Fig 5.

```
configuration BRSConfig
 implements BRSSpec (1.0)
 component_classes
  BikeTrip (1.0)
    as BikeCourse;
  BikeCourseDBClass (1.0)
    as BikeCourseDB
 versionID 1.0
```

```
configuration BRSConfig
 implements BRSSpec (2.0)
 component_classes
  BikeTrip (1.0)
    as BikeCourse;
  BikeCourseDBClass (1.0)
    as BikeCourseDB;
  StationData (1.0) as GIS
 versionID 2.0;
 pre_version 1.0;
 by additionStationDataList;
```

Fig. 5. Example of a configuration description before (left) and after (right) evolution and versioning.

As we know changes as first class information, it becomes easy to draw a change-based version tree of software systems according to their architecture descriptions. Fig. 6 shows an example of such a version tree for the BRS.

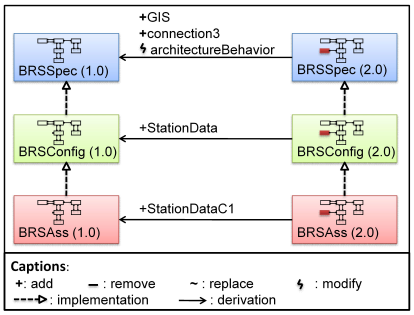

Fig. 6. Version tree of the BRS

IV. EVOLUTION IN MAJOR ADLS

In this section, we give an overview on how existing ADLs express architecture evolution from the viewpoints of architecture changes and versioning.

1) Change Description: Most existing ADLs do not support change description. Darwin [13], [14] and C2SADEL [15] are exceptions. Darwin uses change specifications to describe changes (create, remove, link, unlink). C2SADEL uses A ML (an architecture modification language) which focuses on describing changes to architecture descriptions [16], [17]. It has more actions than Darwin (createComponent, createConnector, weld, unweld, etc.) [18]. However, AMLs are imperative languages that aim in modifying architectures, but no ADL exists that provide a full and formal description language for evolution covering all its characteristics. Furthermore, these existing ADLs miss another important point: they do not treat changes as first-class entities.

2) Version Model: Versioning problems including variants, evolving artifacts, forking and branching have been studied by software engineering for years, mainly targeted to source code versioning but also, sometimes, to model elements [19]. The main application of these work is *configuration management*. However, versioning software architectures is less studied.

TABLE II
COMPARISON OF CHANGE CHARACTERISTICS IN EXISTING ADLS

Characteristics of change	C2	Darwin	Dynamic Wright	SOFA2.0	xADL2.0	MAE	Dedal
Time of change	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic
Anticipation	Unanticipated	Anticipated, Unanticipated	Anticipated	Anticipated	Unanticipated	Unanticipated	Anticipated, Unanticipated
Change type	Structural	Structural	Structural	Structural	Structural	Semantical	Structural, Semantical
Change purpose	—	—	—	—	—	Perfective	Perfective, corrective
Level of change	Configuration	Configuration	Configuration	Configuration	Configuration	Configuration	Specification, configuration, assembly
Change operation	Addition, removal	Addition, removal	Addition, removal	Addition, removal	Addition, removal	Substitution	Addition, removal, substitution
Subject of change	Components, connectors, connections	Components, connections	Components, connections	Components, connections, interfaces of composite components	Components, connectors, connections	Components	Components, connectors in three levels
Version model	—	—	—	State-based	—	Change-based	Change-based

Ménage [20] is the first work to integrate configuration management into ADLs to control the versioning of architectural artifacts. MAE [21] and xADL2.0 [22] both use the version model proposed by Ménage. However, they focus on the version control of architectural elements, such as component types and connector types. The versioning whole architectures is not studied. SOFA 2.0 [23] enables to version software architectures which are treated as composite components, but its version model is too simple and cannot capture enough information on evolution. A simple solution proposed by Taylor *et al.* [9] is to use CVS [24] or Subversion (SVN) [25] on the texts that describe architectures. This approach does not work well for creating or maintaining component-based software architectures as it is designed to capture the versioning of source codes. Only MAE and xADL can model version histories with branching version trees. These version trees unfortunately apply only on architecture element types.

V. CONCLUSION

In this paper, we discuss architecture-centric evolution in Dedal. Dedal-CDL, a language extension dedicated to describing changes, and a version model for software architectures are proposed to model the evolution information explicitly. With the explicit evolution description capability of Dedal, architecture-centric evolution can be better performed and its quality better guaranteed.

Perspectives for this work are to formalize Dedal and Dedal-CDL using model-driven engineering techniques and migrate our implementation of Dedal and its companion tools for them to become an eclipse plugin.

REFERENCES

[1] ISO/IEC 42010:2011, *Systems and software engineering – Architecture description*, IEEE Standards Association Std.

[2] J. C. Georgas, E. M. Dashofy, and R. N. Taylor, "Architecture-centric development: a different approach to software engineering," *ACM Crossroads*, vol. 12, no. 4, pp. 6–6, 2006.

[3] T. Mens, J. Magee, and B. Rumpe, "Evolving software architecture descriptions of critical systems," *IEEE Computer*, vol. 43, no. 5, pp. 42–48, 2010.

[4] I. Crnkovic, M. Chaudron, and S. Larsson, "Component-based development process and component lifecycle," in *ICSEA '06*, Papeete, French Polynesia, october 2006, p. 44.

[5] M. R. V. Chaudron and I. Crnkovic, *Software Engineering: Principles and Practice*. John Wiley & Sons, 2008, ch. Component-based Software Engineering, pp. 605–628.

[6] H. Y. Zhang, C. Urtado, and S. Vauttier, "Architecture-centric component-based development needs a three-level ADL," in *ECSA'10*, Copenhagen, Denmark, 2010, pp. 295–310.

[7] H. Y. Zhang, "A multi-dimensional architecture description language for forward and reverse evolution of component-based software," Ph.D. dissertation, Montpellier Uiniversity II, 2010.

[8] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a taxonomy of software change: Research articles," *J. Softw. Maint. Evol.*, vol. 17, no. 5, pp. 309–332, September 2005.

[9] R. N. Taylor, N. Medvidovic, and E. M. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons, January 2009.

[10] G. E. Kniesel and R. E. Filman, "Unanticipated software evolution: Issue overviews," *J. Softw. Maint. Evol.*, vol. 17, no. 5, pp. 307–308, 2005.

[11] ISO, *Standard 14764 on Software Engineering - Software Maintenance*. ISO/IEC, 1999.

[12] H. Y. Zhang, C. Urtado, and S. Vauttier, "Connector-driven process for the gradual evolution of component-based software," in *ASWEC'09*, Gold Coast, Australia, April 2009.

[13] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik, "Abstractions for software architecture and tools to support them," *IEEE Trans Software Eng*, vol. 21, no. 4, pp. 314–335, 1995.

[14] M. Shaw, R. DeLine, and G. Zelesnik, "Abstractions and implementations for architectural connections," in *ICCDS '96*, Annapolis, Maryland, 1996, pp. 2–10.

[15] N. Medvidovic, D. S. Rosenblum, and R. N. Taylor, "A language and environment for architecture-based software development and evolution," in *ICSE'99*, Los Angeles, CA, May 1999, pp. 44–53.

[16] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *ICSE'98*, Kyoto, Japan, April 1998, pp. 177–186.

[17] B. Agnew, C. Hofmeister, and J. Purtilo, "Planning for change: a reconfiguration language for distributed systems," in *Proc. 2nd International Workshop on Configurable Distributed Systems*, 1994, pp. 15–22.

[18] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Runtime software adaptation: framework, approaches, and styles," in *ICSE Companion*, 2008, pp. 899–910.

[19] C. Urtado and C. Oussalah, "Complex entity versioning at two granularity levels," *Information Systems*, vol. 23, no. 2/3, pp. 197–216, 1998.

[20] A. van der Hoek, D. Heimbigner, and A. L. Wolf, "Software architecture, configuration management, and configurable distributed systems: A menage a trois," University of Colorado, Tech. Rep. Computer Science Technical Report CU-CS-849-98, 1998.

[21] R. Roshandel, A. V. D. Hoek, M. Mikic-Rakic, and N. Medvidovic, "Mae–a system model and environment for managing architectural evolution," *TOSEM*, vol. 13, no. 2, pp. 240–276, 2004.

[22] E. M. Dashofy, A. van der Hoek, and R. N. Taylor, "An infrastructure for the rapid development of xml-based architecture description languages," in *ICSE '02*. Orlando, Florida: ACM Press, 2002, pp. 266–276.

[23] T. Bures, P. Hnetynka, and F. Plasil, "Sofa 2.0: Balancing advanced features in a hierarchical component model," in *SERA '06*, Seattle, USA, 2006, pp. 40–48.

[24] K. Fogel, *Open Source Development with CVS*. Scottsdale, Arizona: CoriolisOpen Press, 1999.

[25] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick, *Version Control with Subversion, 2nd Edition*. O'Reilly Media, Inc., 2008.