Evolve: Tool Support for Architecture Evolution

Andrew McVeigh, Jeff Kramer, and Jeff Magee
Imperial College London
{a.mcveigh,j.kramer,j.magee}@imperial.ac.uk

ABSTRACT

Incremental change is intrinsic to both the initial development and subsequent evolution of large complex software systems. Evolve is a graphical design tool that captures this incremental change in the definition of software architecture. It supports a principled and manageable way of dealing with unplanned change and extension. In addition, Evolve supports decentralized evolution in which software is extended and evolved by multiple independent developers. Evolve supports a model-driven approach in that architecture definition is used to directly construct both initial implementations and extensions to these implementations. The tool implements Backbone - an architectural description language (ADL), which has both a textual and a UML2, based graphical representation. The demonstration focuses on the graphical representation.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques

General Terms

Design.

Keywords

Software Architecture Evolution.

1. INTRODUCTION

Architectural Description Languages (ADLs) have been proposed by the research community, for example [1, 2, 3], as a way of capturing the high-level structure of complex software systems in terms of components and connectors. While software architecture is seen as an appropriate level for system redesign and restructuring to permit change, the proposed ADLs do not deal directly with evolution, considering it to be an extrinsic concern dealt with by tools and processes external to those concerned with architecture definition. Mae [4] takes the approach of making a configuration management system aware of architectural constructs, however, it does not have an ADL that can capture change.

The Backbone ADL and its associated design tool Evolve are the result of research [5, 6] to explore the alternative in which evolution is regarded as a concern that is intrinsic to architecture definition such that the structural constructs included in Backbone to capture change and extension can be used during both initial development and subsequent evolution. This intrinsic definition brings with it the requirement to deal with unplanned extension, for it is impossible, whichever development process is adopted, to foresee all possible future requirements for change and evolution of a system.

ICSE'11, May 21-28, 2011, Waikiki, Honolulu, HI, USA.

Dealing with unplanned extension introduces a difficult dilemma in designing constructs to support intrinsic definition. We would prefer to have constructs that constrain change in such a way that their application always results in structurally well formed and type correct systems. However, such an approach inevitably means that only a sub-class of all possible valid change to a system is permitted. This is clearly incompatible with dealing with unplanned extension since we cannot predict the change that will be necessary. As a result, Backbone has constructs that permit changes that result in invalid systems; however, Evolve does comprehensive structural and type checking. In other words, the approach combines the freedom to perform incorrect changes with the ability to detect these errors so that there is sufficient expressiveness to deal with unplanned changes. Change can be destructive – deleting elements of an architecture – in addition to being constructive – adding elements to an architecture.

Evolve supports distributed design and evolution in which development and extension is carried out by different organizations. The demonstration presents an example of the following scenario which current approaches to managing evolution find problematic. A development organization produces a software framework product that is used by other organizations to build applications. In meeting their local development requirements, these organizations may need to modify and extend the framework to support their applications without necessarily having access to its source code. The original framework will evolve over time and the organizations that use it need to apply their local changes to the framework before using the evolved framework for their applications. In addition, a third party may wish to use applications from more than one extenders of the framework and thus need to merge changes from both these organizations and the original framework provider.

If software architecture description is regarded only as design documentation, a major problem arises in keeping this documentation in step with the software implementation as a system evolves. Evolve supports a model driven engineering approach in that the Backbone architecture definition is not just a documentation artifact but is a precise model used to directly construct both initial implementations and extensions to these implementations.

2. DEFINING CHANGE IN BACKBONE

Backbone includes *resemblance* and *replacement* constructs that facilitate the definition of an evolved architecture in terms of a previous version. An architecture here is a hierarchically structured composition of components. A *stratum* is a packaging construct used to manage and distribute changes.

Resemblance defines a new component as the difference in structure from one or more existing components. It is the delta, consisting of the set of additions, deletions and replacements of the elements of these components, that is applied to arrive at the new definition. For a component, these elements are: parts – instances of subcomponents, ports – instances of interfaces, connectors – bindings between ports, and attributes – component parameters. Resemblance may also be applied to interfaces in which case the modified elements are operations. If a resemblance delta consists only of additions then when applied to an interface, it defines a proper subtype. Resemblance is a many to one relation to permit the merging of multiple component definitions that arise in distributed development.

Replacement globally substitutes the definition of one component for another while preserving the identity of the original definition such that any use relations that a larger system has with this definition are preserved. When combined with resemblance, replacement permits the incremental evolution of a component definition without necessarily having to change the composite component definitions that use this component. Components and interfaces in the Backbone ADL are given globally unique identifiers to permit the correct unambiguous application of replacement. Replacement is the key to managing change in composite hierarchical definitions since it permits substitution of component definitions at one level of the composition hierarchy without necessarily affecting higher layers.

Stratum is the container for architecture definitions. Evolve uses the current stratum and the transitive closure of all strata that this stratum depends on to assemble the system that the stratum represents. The stratum is the unit of sharing and ownership. Each stratum is owned by a single party that has modification rights to it. Evolve and the Backbone runtime environment support the import and export of strata for distributing extensions and subsets of an architecture. A stratum is also a unit of deployment in that its compiled version together with associated component code can be sent to an end user to extend their system in a similar fashion to Eclipse Plugins.

Example

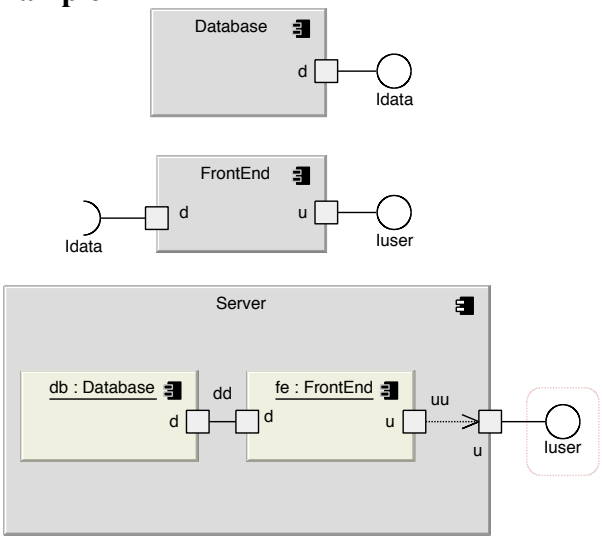

Figure 1: Software Architecture of a simple database server

Figure 1 depicts the software architecture of a simple database server that has two internal component parts – Database and FrontEnd. Figure 2 shows an evolution of the simple server of Figure 1 that has been extended using resemblance to add access management to the data stored in the server. The diagram depicts that ManagedServer resembles Server and the text note is the Backbone definition of the delta that results from editing Server to arrive at ManagedServer. Note that the diagrams have been directly cut and pasted from Evolve and that Backbone code is never directly input but produced as the result of graphical edits.

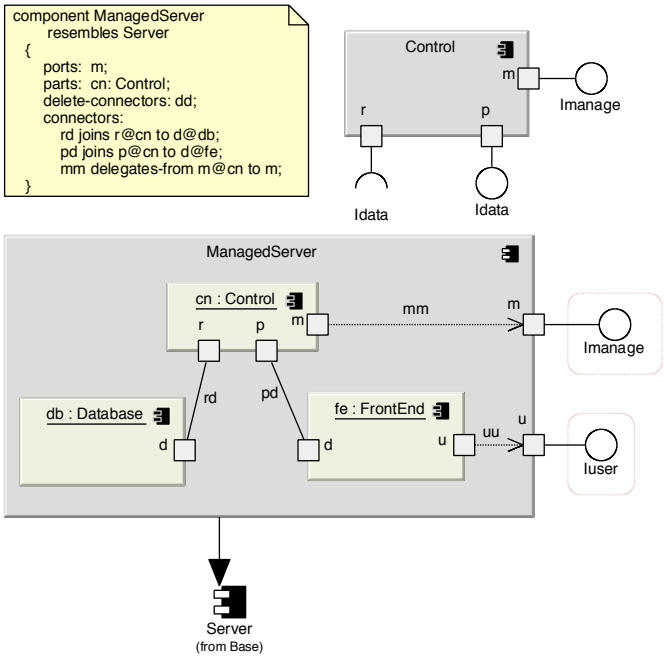

Figure 2: Software Architecture of managed database server

Figure 3 is an example of replacement in which an improved implementation of the Database component is substituted. It can be applied to both the simple base server system and the extended server system.

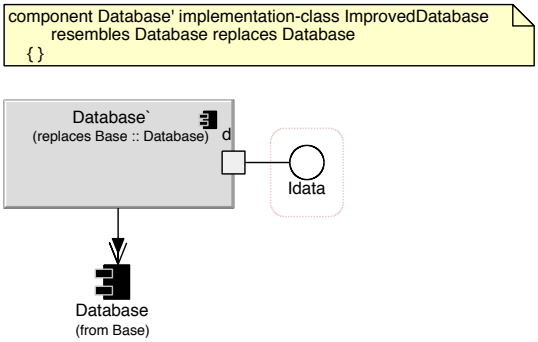

Figure 3: Replacing the database component

The original system of Figure 1 and the two changes represented by Figures 2 & 3 are packaged into the strata shown in Figure 4. Note that the dotted lines between strata represent dependencies. To build a system with access management and the

improved database, we simply need to create a new stratum called Combined with dependency links to the Managed and Improved strata. Evolve can automatically assemble the system represented by Combined by merging the strata it depends on – in addition conflicts are detected.

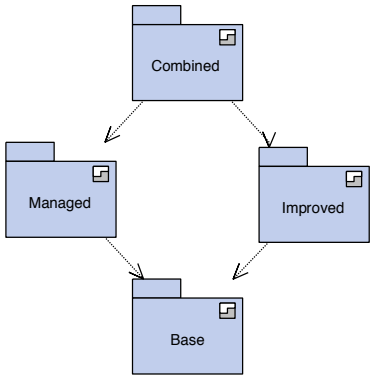

Figure 4: Database Strata diagram

3. EVOLVE

Evolve is the graphical modeling tool that supports definition and evolution of systems described in the Backbone ADL. The Backbone runtime environment instantiates and interconnects components from a Backbone description. Figure 5 is an overview of the elements that make up the modeling tool and runtime environment. The DeltaEngine layer implements the resemblance algorithm required to combine multiple delta definitions contained in multiple strata. The algorithm has been rigorously specified and modeled in Alloy to ensure that structural conflicts can be comprehensively detected when strata are merged [6]. The DeltaEngine is used both in the modeling tool to build graphical representations of composite components from delta definitions and in the runtime environment to instantiate components from these same delta definitions. In addition to this interpreted runtime, the modeling tool can optionally compile a "flat" description of the model as a builder class. In this case, no runtime environment is required. It should be noted that even the interpreted runtime environment incurs no overhead during execution of a system; it is only active at startup time when it directs instantiation and interconnection.

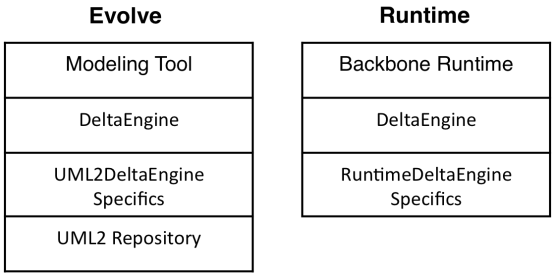

Figure 5: Evolve and runtime implementation

Evolve is currently targeted at systems programmed in Java and as such it requires access to the Java libraries. Consequently, the tool has the facility to import JavaBeans as components. Since JavaBeans typically have a large number of interfaces and attributes, the tool has the facility to manage the visibility of these.

The tool is currently available from http://www.intrinsarc.com/, the website of a small private company set up by Andrew McVeigh to promote the commercial use of his PhD research. Evolve is freely available for academic research and the production of open source software under the GNU Affero General Public License version 3. The demonstration video is available at this site at the following URL:

http://intrinsarc.com/movies/evolve.html

and at:

http://www.youtube.com/watch?v=fRAp4no3hFs&hd=1

The video shows the tool being used to construct and evolve a simple system.

4. CONCLUSION

Evolve and its associated architectural description language Backbone has been developed to investigate the advantages of capturing the description of change in architecture definition. The approach is proposed as a way of dealing with the unplanned change that is a fundamental characteristic of evolution. The approach permits the deployment of system extensions with the same facility as plugin architectures, however, it does not require preplanned extension points. The ability to deal with decentralized development is critical and addresses a key problem that faces the providers and users of software frameworks. The tool has currently been applied to a number of large case studies including merging of a number of diverging variants of the author's LTSA tool – an experiment in addressing legacy systems. We are eager to facilitate its use by the research community and see it as a basis for exploring challenges such as behavioral modeling and component compatibility in an evolutionary context. The tool currently provides structural and static interface compatibility checks. Including behavioral compatibility checks is ingoing research.

5. REFERENCES

[1] D. Garlan, R. Monroe, and D. Wile. Acme: an architecture description interchange language. In CASCON '97: *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 7. IBM Press, 1997.

[2] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying distributed software architectures. In W. Schäfer and P. Botella, editors, *Proc. 5th European Software Engineering Conf.* (ESEC 95), volume 989, pages 137–153, Sitges, Spain, 1995. Springer-Verlag, Berlin.

[3] R. Taylor, N. Medvidovic, M. Anderson, E. Whitehead Jr., and J. Robbins. A component- and message-based architectural style for gui software. *In Proceedings of the 17th international conference on Software engineering*, pages 295–304, Seattle, Washington, United States, 1995. ACM Press.

[4] R. Roshandel, A. Van Der Hoek, M. Mikic-Rakic, and N. Medvidovic. Mae—a system model and environment for managing architectural evolution. *ACM Trans. Softw. Eng. Methodol.*, 13(2):240–276, 2004.

[5] A. McVeigh, J. Kramer, J. Magee. Using resemblance to support component reuse and evolution. In *Proceedings of the 2006 Conference on Specification and Verification of Component-Based Systems* (Portland, Oregon, November 10 - 11, 2006). SAVCBS '06.

[6] A McVeigh, A Rigorous, Architectural Approach to Extensible Applications, PhD Thesis, Department of Computing, Imperial College London, 2009.
(available at:- http://www.intrinsarc.com/backbone/research)