Dynamic Architecture: Structuring for Change

W. Morven Gentleman
GINI University Services Inc.
6050 University Avenue
Halifax, NS, B3H 1W5
CANADA

Morven.Gentleman@dal.ca

ABSTRACT

Architecture is the structure by which a system is organized. Dynamic architectures facilitate change to occur in the architecture itself while the system is in use. For long-lived mission-critical systems, disrupting service in order to apply upgrades entails unacceptable delays and risks. Many other systems involve structures that intrinsically change, and this change must be described and modeled. The study of dynamic architectures has thus become increasingly important over the past decade.

WHAT IS AN ARCHITECTURE?

The term "architecture" is used in many disciplines to refer to the structure or framework by which a system is organized. It is common for an architecture to have two aspects. First, there are the levels of abstraction at which the system is thought about. Second, at any level of abstraction, there are the components that make up the system as viewed at that level, and there are the relationships between those components. The components have roles in the organization of the system, and to fulfill those roles have responsibilities and resources. The components are described by attributes. In many systems the roles are active, not merely passive, and components exercise control over other components in order to accomplish the functioning of the system. A common kind of relationship between components is that something is transferred between those components, be that physical, information or just control. Such a relationship is often called a connector.

The structure or framework in which a system is organized is often crucial to the performance of the resulting organization and to how the organization can evolve

The most familiar use of the term "architecture", of course, is in the context of buildings. However, as mentioned above, the term has been used widely in diverse other contexts. For example, in the study of living things, one talks of the architecture of the cellular structure of plants. The anatomy of the human body has been described as having an architecture consisting of organs with major subsystems such as the nervous system and the circulatory system. Manufacturing systems refer to the architecture of product lines. The

Paper presented at the RTO IST Symposium on "Coalition C4ISR Architectures and Information Exchange Capabilities", held in The Hague, The Netherlands, 27-28 September 2004, and published in RTO-MP-IST-042.

corporate structure of an enterprise is described as an architecture. Business processes and workflows form an architecture. In the military context, the order of battle of a force has been termed an architecture, and a plan for logistics support has an architecture.

In information technology, there are four contexts in which we use the term architecture.

1. The hardware architecture of particular computing equipment.
2. The network architecture of communications system infrastructure
3. The information architecture for information acquisition, retention, retrieval, and dissemination.
4. The software architecture for software applications and systems.

HOW IS AN ARCHITECTURE USED?

In most of the contexts where the structure of systems is described as an architecture, the architecture is used in a number of ways:

1. *To communicate about the system*
   The stakeholders with respect to a system need to be able to communicate with each other about that system in particular, and more generally about that class of systems. The architecture gives them a shared vocabulary to do so, and can establish a basis of common assumptions.
2. *To educate with respect to the system*
   As a special case of communication, newcomers learning about a system need a vocabulary in which to be told about the system. An architecture offers more, however, in that it can provide a roadmap, a logical structure of how to learn about the system, a sequence in which to establish conventions and properties that help to make subsequent aspects of the system easier to appreciate.
3. *To analyze the system*
   A powerful motivation for specifying an architecture for a system is that it facilitates reasoning about the system and analyzing system properties. It not only provides a theoretical perspective of the system, but can also suggest empirical measurements that can be made to gain increased understanding of the operation of the system in practice.
4. *To initially construct the system*
   For systems that are designed and built by humans, the architecture not only serves as a tool for expressing the design, but it can also provide guidance for the process of initial construction. This includes recognizing possibilities for concurrent development because of lack of dependencies, and suggestions for scaffolding to substitute for components not yet available. It also can suggest possibilities for early fielding through incremental development, where worthwhile functionality is available once subsets of the components are complete.
5. *To maintain the system*
   Systems that suffer from wear and tear require regular maintenance. The architecture may indicate what kind of maintenance is called for, and even how the maintenance could be performed. Other systems also may require attention from time to time, perhaps to make minor adjustments to accommodate the environment, and this too can be characterized as maintenance and can be guided by architecture.
6. *To repair the system*
   Many systems in operation may be subject to damage, either damage to the system itself or damage to whatever environment in which the system operates. Although in some cases the damage might be

incurred because of deficiencies in the system itself, damage can also be incurred from wholly external factors. In either case, restoring the original system to working order may be a good compromise response to the damage, or modifying the system to deal with new realities may be more appropriate. The architecture can help assess this choice, as well as guiding the process of carrying out the repair itself.

7. *To enhance and evolve the system*
   Over time, requirements for many systems change, and the resources available for those systems also vary. This can lead to a need to change the system itself, either by an enhancement that is a simple extension of the existing system, or by a more profound revision that makes more fundamental changes to the implementation or even the architecture of the existing system. The existing architecture is often a good basis on which to consider, evaluate, and plan such changes.

ARCHITECTURE AND CHANGE

A number of the foregoing uses for architecture refer to change. Because architecture suppresses a level of detail present in an actual system, an architecture can sometimes form a static framework for change. Such an architecture can claim to describe systems that are "built to change, and thereby built to last". The architecture actually describes a set of possible actual systems. The architecture (hopefully) indicates where changes can be made without negative consequences, and what must remain unchanged in order to preserve important properties, behaviours, and analyses that have already been established. Adaptive architecture is a term used to describe architectures defined at a level of abstraction whereby it can attempt to respond to changed environment through components that adapt automatically. There is a considerable literature on adaptive architectures, and many examples are cited. The challenge of adaptive architectures, of course, is that the designers must anticipate the kind of changes to which they can respond, and will not be able to cope with different changes.

Although an architecture can accommodate certain kinds of change, sometimes changes are necessary that require the architecture itself to be modified. Consider a typical system whose architecture is made up of components and connectors. What categories of system change might necessitate changes to the architecture itself?

1. A change to what component is connected to which other components may involve a change in the architecture itself.
2. A change in the number of components of a given type or in the number of connections of a given type present in the system may involve a change in the architecture itself.
3. A change in the type of components present in the system, or in the type of connectors (relationships) present in the system may involve a change in the architecture itself.
4. A change in the level of abstraction at which the components and relationships are viewed may involve a change in the architecture itself. (A concrete example of this is when the level of abstraction must be extended to incorporate addressing a new concern, such as safety or security.)

A concrete example illustrating these kinds of changes is a logistics system. Changes in shipping routes might result in supplies being moved between different depots, a change of category 1. Consolidation of supply depots might result in fewer depots in use, a change of category 2. Opening additional routes to reduce traffic on particular roads would be an increase in the number of connectors, a different change of category 2. If only

road transport was initially considered, introducing transport by air would represent a change of category 3. Supplementing military transport by making use of civilian contractors would represent a change of category 4.

Changes in the architecture itself are particularly critical when the system must continue to be operational and provide service during the change process. This leads to the distinction between an online versus an offline upgrade to the system. An architecture capable of supporting an online upgrade to a system is called a dynamic architecture. Another term sometimes used instead is to say that the architecture is dynamically reconfigurable.

Dynamic software architectures have been a topic of active research for over a decade, especially in the context of applications that are systems distributed across networks. To address dynamic architectures, it is necessary to adopt a different style of architectural description, where the initial and final architectures, as well as the various intermediate states and the actual transitions, can be described within the notation. A particular challenge is that although it may be possible (as we will describe) for the executable code for existing components to support changes in the values of attributes or even changes in the set of attributes to be recorded, changes in the behaviour of those components entails replacing the existing code. This can only be done directly if the execution engine is an interpreter that merely interprets some readily manipulated data structure such as a string. For the more conventional compiled machine code run time, a dynamic architecture must include provision for compiling and linking new versions of code, and more importantly, provisions for downloading the code to the execution environment, establishing necessary bindings and initial state, and initiating execution. This involves machinery that is elaborate and not generally available.

Note that an online update may involve a challenge in establishing a clean starting point for the new code. Components in execution have state, and initiating a new component into execution involves initializing the state of that component appropriately, which may require scavenging the state from other components. Sometimes this challenge can be simplified if architectural reconfiguration is only permitted at explicit reconfiguration points, where behaviour is quiescent and state has been externalized to be accessible. In the case where component replacement is necessary to repair a failure, scavenging state and cleaning up debris from previous components in order to facilitate a clean start for the new component can be particularly difficult.

Note that many of the discussions in the literature make the simplifying assumption that it is sufficient for the architecture to represent only the current system state. Unfortunately, there are circumstances where this is not enough: the architecture needs to represent history also. As a concrete example, consider an information architecture where there is a legal obligation to represent managerial responsibility for an item at all previous times, not just as the incumbent of each position changes, but in the face of corporate restructuring, mergers, acquisitions, divestitures, and changes in pertinent legislation. Each contortion of the organization would be hard to anticipate and possibly awkward to represent as it occurred, but maintaining a representation that could sustain all earlier lines of responsibility, through organizational units that possibly no longer exist, could be much more difficult.

Note also that in some systems, it is not sufficient to record stable states, before and after transitions. It may be necessary to reflect the detailed operation of each transition itself.

Where a system is fundamentally concerned with setting up and tearing down connections between diverse components, as happens in job shop work allocation or in shipping goods by competing commercial transport, dynamic software architecture can provide a convenient representation for the system even when online upgrading is not actually necessary. Brief service outages for reconfiguration may be quite tolerable in such systems, and considerably simplify the run time requirements.

KEY IDEAS OF DYNAMIC SOFTWARE ARCHITECTURE

The study of dynamic software architecture has produced some key ideas that may also find applicability in other dynamic architectures, if only because software support assists almost any human endeavor.

1. Provision for on-line upgrade is essential
2. Components should be explicitly represented as objects which can be dynamically allocated and dynamically (re)linked
3. Connectors (relationships) should also be explicitly represented as objects which can be dynamically allocated and dynamically (re)linked
4. Objects should be defined sufficiently generically that they will be stable across all imaginable specialization. Specialization should be done at run-time to object instances
5. Inheritance is a powerful mechanism for facilitating various different specializations of a common generic object. Unfortunately, class-based inheritance as provided by conventional object-oriented language only permits those specializations represented by subtyping and subclassing in the source code. . In many systems subsequent extensions cannot be predicted at the time source code is written. Such extensions can be supported through instance-based object-oriented techniques, where new instances can be created as modifications of existing instances. Extending methods is more challenging than merely extending instance variables. PostScript and XML are examples of languages supporting instance-based object-oriented techniques
6. Web services, such as .Net, provide a restricted but useful support for dynamic software architecture.

CONCLUSIONS

Dynamic software architecture has met a real need for certain important kinds of software systems, Many successful systems have been deployed, and our understanding of the theory, although still incomplete, has suggested best practices. The limited form supported as web services is easily implemented.

We have less experience in using dynamic architecture in architectural contexts other than software, although the driving forces to represent and work with architectural change exist there too. For example, in network architecture one of the refinements that IPv6 introduced over IPv4 is procedures for automatically reconfiguring and rebalancing subnets for internet routing, because experience has taught us that manual reconfiguration was too labour-intensive. Being able to analyze such changes would be a benefit dynamic architectures might have to offer.

BIBLIOGRAPHY

[1] Kramer, J., and Magee, J. "Dynamic Configuration for Distributed Systems" IEEE Trans. on Software Eng., SE-11 (4), (1985), 424-436.

[2] Kramer, J., Magee, J., and Sloman, M.S., "Managing Evolution in Distributed Systems" IEE Software Engineering Journal , 4 (6), November 1989, 321-329

[3] Kramer, J., IEE Software Engineering Journal 8 (2), Special Issue on Configurable Distributed Systems, March 1993, 51-52

[4] Ch. Hofmeister: Dynamic Reconfiguration of Distributed Applications, Ph.D. Thesis, Dept. of Computer Science, University of Maryland, College Park, 1993

[5] B. Agnew, Ch. Hofmeister, J. Purtilo: Planning for Change: A Reconfiguration Language for Distributed Systems, Proceedings of CDS'94, 1994

[6] Magee, J., Kramer, J. “Dynamic Structure in Software Architectures”, (4th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE 4), San Francisco, October 1996), SEN, Vol.21, No.6, November 1996, 3-14.

[7] Karl Lieberherr, *Adaptive Object-Oriented Software: The Demeter Method*, 1996, ftp://ftp.ccs.neu.edu/pub/people/lieber/book/aoos.PDF

[8] Kramer J. and Magee J., "Analysing Dynamic Change in Distributed Software Architectures", IEE Proceedings - Software, 145 (5), October 1998, 146-154.

[9] Oreizy, P. and Taylor, R.N. On the Role of Software Architectures in Runtime System Reconfiguration. IEE Proceedings - Software Engineering. 145(5), p.137-145, October, 1998

[10] Oreizy, P. and Taylor, R.N. On the Role of Software Architectures in Runtime System Reconfiguration. In Proceedings of the Fourth International Conference on Configurable Distributed Systems. p.61-70, IEEE Computer Society Press. 1998

[11] Aguirre, N. & Maibaum, T.S.E., “A Temporal Logic Approach to Component-based System Specification and Reasoning”, Proceedings of 5th ICSE Workshop on Component-based Software Engineering, ICSE 2002 www.sei.cmu.edu/pacc/CBSE5/, 2002

[12] Aguirre, N. & Maibaum, T.S.E, “A Temporal Logic Approach to Specifcation of Reconfigurable Component-based Systems”, Proceedings of 17th International Conference on Automated Software Engineering, ASE 2002, IEEE CSE Press, 2002, 271-274

[13] Robert P. Bialek., “The Architecture of a Dynamically Updatable, Component-based System" Workshop on Dependable On-line Upgrading of Dist. Systems in conjunction with COMPSAC 2002, (Oxford, England), Aug. 2002.

[14] Craig A. N. Soules, Jonathan Appavoo, Kevin Hui, Robert W. Wisniewski, Dilma Da Silva, Gregory R. Ganger, Orran Krieger, Michael Stumm, Marc Auslander, Michal Ostrowski, Bryan Rosenburg, and Jimi Xenidis, `System Support for Online Reconfiguration". In Proc. of the Usenix Technical Conference, 2003.

[15] *Dynamic Architecture: How to Make Enterprise Architecture a Success* Martin van den Berg, Joost Luijpers, Marlies van Steenbergen, Roel Wagter, Klaas Brongers, ISBN: 0-471-68272-1, 256 pages, January 2005