JOURNAL OF OBJECT TECHNOLOGY

Online at http://www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2004**

Vol. 3, No. 5, May-June 2004

Software Architecture

John D. McGregor, Clemson University and Luminary Software, U.S.A.

Abstract

Software architecture has received much attention in the past few years. This is not a block diagram that gives a rough functional decomposition of the system. It is a multi-perspective, quality-based approach to ensuring that software is built to fit its purpose. In this edition of Strategic Software Engineering we will consider how software architecture provides strategic support to the organization.

1 INTRODUCTION

This column began in the last issue with no overview of its scope and I would like to provide that now. As the name "Strategic Software Engineering" implies, I intend to discuss aspects of software engineering that assist a company in achieving and maintaining strategic advantage. As software becomes omnipresent in products in all domains, we have increasing opportunities to not only help achieve the vision of our companies but also to help shape what that vision is. Since "strategy" is not a prominent idea in technical education and training, I hope this column will raise awareness and provoke discussion.

In the last issue I provided an overview of the software product line strategy. This approach to software-intensive product development has allowed a number of companies to achieve strategically significant objectives that have affected their market position. In this issue I will discuss the strategic impact of modern software architecture approaches. In future issues I will talk about topics such as domain-centric development, the range of development process models, and the strategic implications of testing. I look forward to hearing from readers with differing opinions or who have suggestions for topics.

The software architecture practiced today is not your mom's architecture. The days of drawing a simple block diagram to identify separate functional units or handcrafting each individual application are over [Lloyd 99]. Today's software architecture provides a more comprehensive, but also more exacting, model of the completed system. Software architecture techniques support sophisticated analyses of the system before it is implemented. These techniques, such as Attribute Driven Design (ADD) [Bass 03], ensure that the software implemented using the architecture is fit for its intended purpose.

Cite this column as follows: John D. McGregor: "Software Architecture", in *Journal of Object Technology*, vol. 3, no. 5, May-June 2004, pp. 65-77.
http://www.jot.fm/issues/issue_2004_05/column7

There are a number of examples of using architecture for strategic purposes. The Common Object Request Broker Architecture (CORBA) is one well-known example. CORBA was used to bridge together various legacy systems, integrate systems written in multiple languages, and facilitate interaction between machines with different hardware architectures. This was intended to provide legacy systems with new life while allowing new applications to be quickly integrated giving the company a strategic advantage over companies that modified legacy systems. In this column I will explore a more recent example: a very brief case study on the Eclipse open source IDE platform.

In this column I will provide an overview of modern software architecture concepts. Then I will show how software architecture can have strategic impact. I will conclude by suggesting some actions that archiects can take to influence business strategy.

2 OVERVIEW OF SOFTWARE ARCHITECTURE

The architecture of a software system defines its structure. In fact, it defines several structures, each of which comprises elements and the relationships among those elements. The elements may be computational entities related by control flow or business entities connected by semantic constraints. (The software architecture web page at the Software Engineering Institute, http://www.sei.cmu.edu/architecture/definitions.html, has a large number of definitions. Take your pick.)

The basic architecture design process consists of a systematic decomposition of elements into aggregates of more detailed elements. In **Figure 1** the initial monolithic element is decomposed into a Model, Controllers, and Views elements. Each of the new elements contributes to the total behaviors that the original element had.

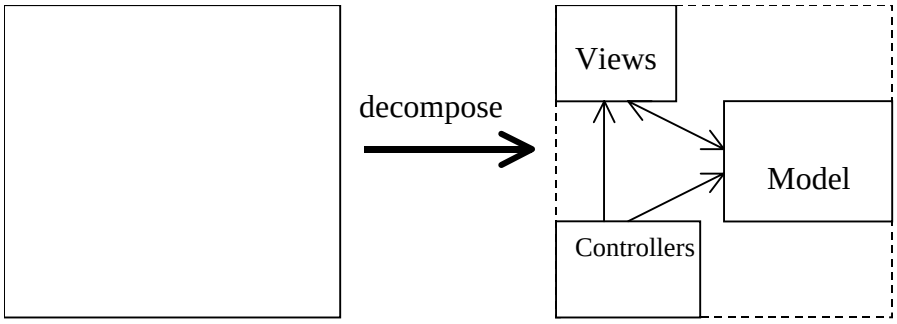


Figure 1 - Architectural decomposition

Using an approach such as ADD, the architect chooses a specific decomposition in order to enhance certain of the qualities the final product should possess. Each decomposition also usually degrades certain qualities as well. In **Figure 1**, the Model-View-Controller decomposition is chosen to make the system more modifiable. Specifically to enhance the ability to add new views on the model. The decomposition also degrades performance because of the overhead of the Model notifying Views when a change has occurred. This

is an acceptable trade-off for the architect since the system only operates in human time as opposed to being a "real-time", i.e. computer time, system.

The architecture creation process seeks to develop a system with specified functionality and specified levels of certain qualities, which have been prioritized. Beginning with a monolithic structure that embodies all of the required functionality, represented by the box on the left in **Figure 1**, the architect systematically decomposes the functionality and allocates it to members of the newly created structure.

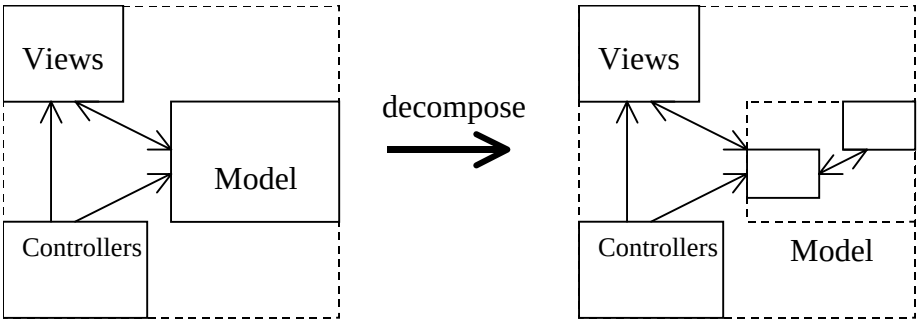


Figure 2 - Further decomposition

The decomposition process continues, as shown in **Figure 2**, until the members of the architectural structure are sufficiently fine-grained. That point is reached when the optimum mix of quality levels is achieved. Then, individual teams can be assigned to design and implement each "chunk" of the architecture.

The drawings in **Figure 1** and **Figure 2** intuitively show the elements and relationships in our simple architecture but they are not very specific. The drawing in **Figure 1** communicates to those who already know the story of MVC but it does not have sufficient detail for the novice to understand. A number of architecture description languages have been developed [Luckham 95][Allen 94] that support detailed specifications of architectures; however, these have never gained widespread acceptance. Many of the concepts represented in these languages have been incorporated into the latest version of the Unified Modeling Language, UML2.0[OMG03]. **Figure 3** illustrates some of the notation for the information in the decomposition on the right in **Figure 1**.

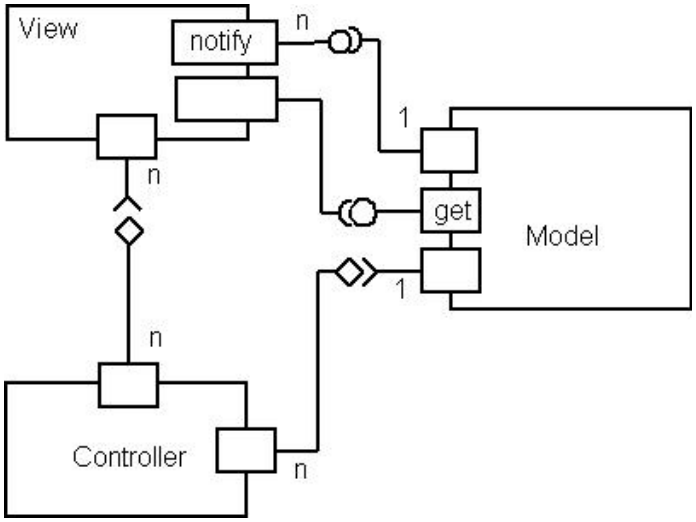


Figure 3 - UML architecture diagram

In **Figure 3**, the UML diagram provides additional information about the provides/requires and signals/catch interfaces for each module. Each small rectangle on the boundary of a larger rectangle represents a port, an abstraction of an interface which hides exactly how the interface is related to the module, the bigger rectangle. Each "lolipop" and "cup" represent a provides and requires defintion respectively. This shows diagrammatically the relationships among modules but does not provide the details of the required and provided behaviors. I am not trying to give a tutorial on UML2.0 or architecture documentation. This diagram should be sufficient for the references that I will make later in the column.

This still captures only a small fraction of the information needed. In this case, largely information about the static structure. The architect creates various diagrams to depict the component structure, component communication, and the component deployment. The architecture description contains several "views" of the architecture that serve to provide different types of information about the structure of the software. One of the dynamic interactions among the static architectural elements from **Figure 1** is shown in **Figure 4** using a UML sequence diagram.

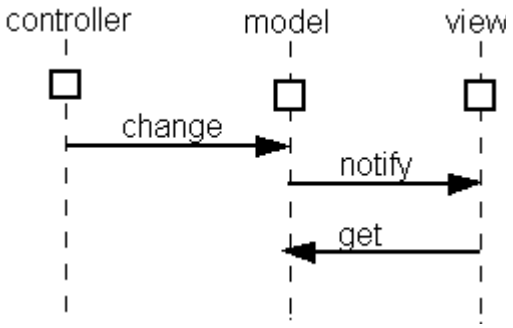
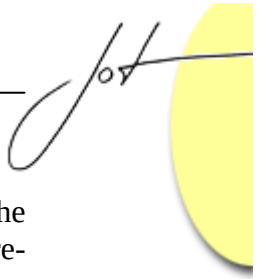


Figure 4 - A change to the Model is propagated



While this describes basic architecture design actions, it does not realistically portray the typical architecture development approach. In most cases, the architect begins with a pre-defined, i.e. reference, architecture. This may be the proverbial "de facto industry standard", such as J2EE for e-commerce, web-based applications or it may truly be a requirement such as the Command, Control, Communications, Computer, Intelligence, Surveillance, and Reconnaissance (C4ISR) architecture framework for certain military systems in the United States. These reference architectures provide a high-level decomposition that sets the basic levels of design qualities but leaves room for the architect to perform low-level decompositions that further differentiate the quality attribute values for their product.

Selecting a specific reference architecture leads to the use of components that have been developed with that architecture as a design base. The commercial community that grows around an architecture is an important factor in selecting which architecture to choose. The larger and more diverse the community, the more likely that components can be purchased to speed product development. Also patterns, books of advice and examples, and other assets are available for that architecture.

The architecture development process is part creative construction and part management planning. The architecture business cycle [Bass 03] provides the opportunity for the organization to think at a strategic level about the competition, trends in their domain, and the evolution of technology. In the next section I will provide some examples of these activities.

3 STRATEGIC USE OF SOFTWARE ARCHITECTURE

In any company, architects help translate business strategy into technical strategy. In companies that produce software-intensive products[1], the architect can have input into business strategy [Malan 02]. This ensures the architecture fits the corporate strategy and it helps the company take advantage of strategic opportunities provided by the architecture. Porter describes strategy as "defining a company's position, making trade-offs, and forging fit among activities"[Porter 96]. I want to examine a number of architecture-related situations in terms of that definition.

Strategic Planning

Architecture can serve as a vehicle for positioning the company. Maccari and Galal define an architecture view, termed the architectonic view, which provides a means for explicitly representing the actual and planned evolution of the architecture [Maccari 02]. This view considers the system being architected as a set of layers that divide the modules of the architecture according to the likelihood the module will change. This can be based both on historic data and forecasts for technology and domain changes. **Figure 5**

[1] By software-intensive I mean that the production of the software content requires a significant portion of the development time or represents a significant portion of the investment on the part of the company.

shows the architectonic view of the MVC where the addition or deletion of a View is most likely change to the system. This view can capture strategic information from domain studies and other business intelligence.

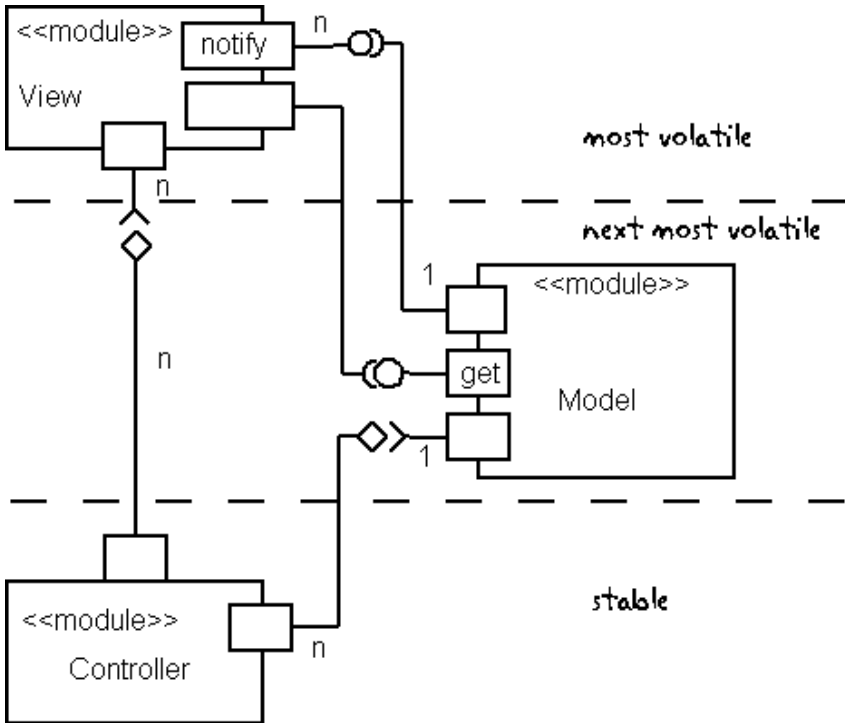


Figure 5 - Architectonic view of MVC

The architecture can provide a strategic roadmap for the company's products, perhaps as a product line [McGregor 04]. Architecture analysis techniques provide a means of validating the direction of the evolution of the architecture and the products.

Enterprise Architectures

An enterprise architecture is used to provide top level executives with an integrated view of the totality of their Information Technology (IT) enterprise. An enterprise architecture shows how all of the facets of an organization's computing environment fit together. While the enterprise architecture encompasses more than software architectures, it does provide a context in which software architectures are coordinated.

In a series of articles Malveau describes how an enterprise architecture bridges the gap between business and technical strategies. He makes the point that "crucial high-level business decisions should not be contrained by technical concerns.[Malveau 04]" I agree but with the emphasis on the "should." In fact, technical concerns sometimes do need to be considered in companies whose chief business is the development of software-intensive products. Telecommunications and, most recently automotive, executives are

finding that their companies have become software dependent. They are adopting strategies that flow from standard architectures in their domain. Several automobile manufacturers including BMW, VW, and DaimlerChrysler announced plans to create, and market, a standard software architecture for automobile electronics, AUTOSAR [Hansen 03].

The Zachman Framework for Enterprise Architecture [Zachman 87], **Figure 6**, provides a context in which the input from technical staff is blended with input from executives to establish the information technology strategy for an organization. The framework guides the enterprise architects to trade-offs that must be made among the information represented in each cell of the table. By resolving these trade-offs, there is close fit among the activities represented in the cells.

For example, consider the Application Architecture cell, at the intersection of the two arrows in **Figure 6**. Adjacent cells relate the application architecture to business processes, up, the system design, down, logical data model, left, and distributed architecture, right. The row provides an architectural view while the column represents a hierarchical relationship from abstract to concrete, from domain to implementation.

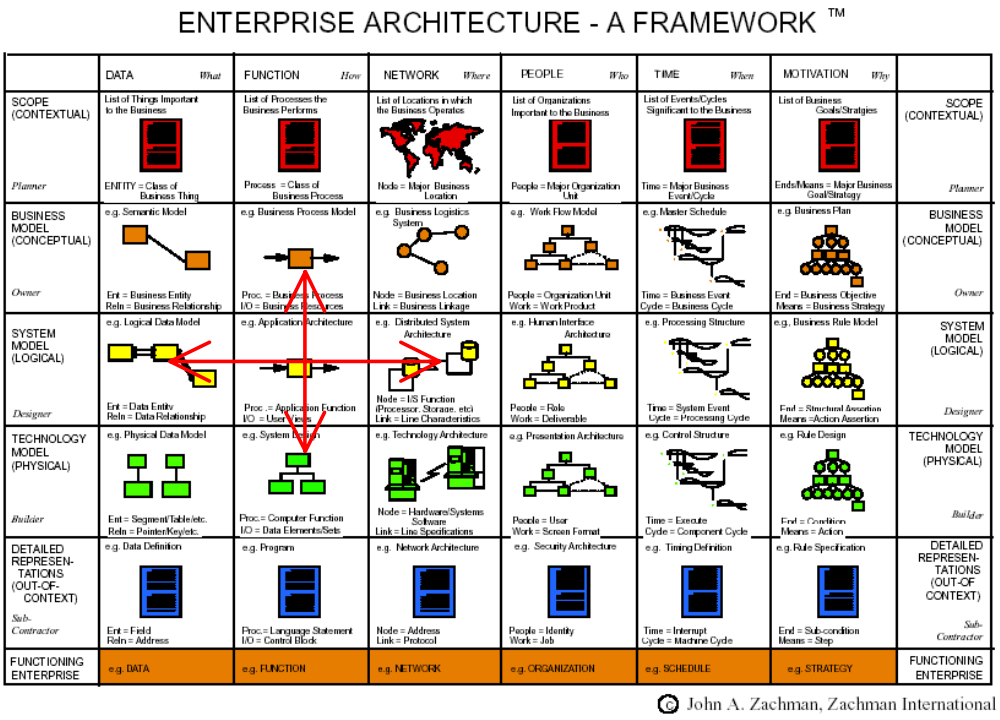
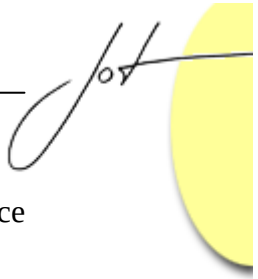


Figure 6 - Zachman Framework for Enterprise Architecture

Reference Architectures

Choosing a reference software architecture, say the J2EE architecture, as the starting point for a product or product line has strategic implications. For one thing, when you choose a reference architecture you join a community, something like marrying into a family. The community has a history and a culture.

- A major consideration in selecting one architecture over another is the availability and expense of artifacts. The community that builds up around an architecture will have a "cultural" bias about how the community operates. Consider the availability and cost of artifacts for the J2EE environment and for the CORBA environment. The selection of one community over another has major cost implications. The J2EE community has much more of a low-cost, open source history than does the CORBA community.
- One of the technical concerns that should significantly influence business decisions is the company's prior investment and required future investment in staff expertise. Most product development requires technical knowledge and skill that can not be developed within the development schedule for a single product. Technical managers make investment decisions in advance of product requirements. The complexity of reference architectures and their accompanying infrastructure require substantial time to learn. A business decision to adopt a different strategic direction that will make obsolete much of the technical knowledge of the development staff and require substantial investment in acquiring new expertise must include consideration of these costs.
- Although the time between releases of versions of a reference architecture is relatively long, the amount of change is often rather large. The anticipated sequence of releases has strategic significance since it influences the refresh rate of the development assets. Major, frequent changes to the architecture requires that assets be modified often. Suppliers work to the latest version of a standard. Not upgrading to the latest version can make it difficult to purchase components and can signal to customers that obsolecence is just ahead. I have had success with a view in the architecture that captures release dates for technologies, products, and standards. It can be combined with the architectonic view shown in **Figure 5**.
- The appropriateness of the reference architecture and its generality affect the applicability to a wide range of products. Earlier I described a decomposition technique that ensured that the architecture possessed the required qualities. When a reference architecture is to be selected, its qualities are already determined. The candidate architectures must be evaluated, using a technique such as the Architecture Trade-off Analysis Method (ATAM) [Bass 03], to determine which most completely supplies the quality levels that products built from the architecture are expected to possess. For example, I once had a client who was using the J2EE architecture for an application that consisted largely of streaming



video. The transaction-orientation of J2EE was resulting in very poor performance for the video feed.

Architecture as Product

After you have optimized delivery to your primary markets, the next strategy is to position yourself as a supplier to your competitors. For example, Nokia has achieved a strategic advantage in the cellular phone industry by establishing their architecture and component libraries, the S60 platform, as a product that provides several competing cell phone manufacturers a large percentage of the software needed for their cell phone products [Nokia 04]. They have provided a platform and an architecture that have attracted additional independent suppliers who provide products based on the architecture. Nokia has made this a strategic approach by providing an architecture and component libraries for several different types of cell phones. Because of its position as supplier, Nokia is now in a position to chart the direction of future evolution not only of their own products but of those of their competitors as well. They have achieved a close fit between their activities as a producer of products and as a supplier of parts.

4 CASE STUDY: ECLIPSE

Eclipse is an open source project producing a platform upon which integrated development environments can be built. This is a project with an architecture that is intended to enhance the quality of supporting distributed development and maintenance of behaviors. Beyond a basic core functionality, behaviors are added to Eclipse by defining and implementing a plug-in. A plug-in is a package that combines code that implements the functionality being added to Eclipse and an xml-based manifest that provides configuration information. The manifest describes extensions to menus, functionality that must be present for the plug-in to work correctly, and new windows that will be part of the Eclipse view. **Figure 7** shows a screen print of Eclipse I took recently while using the Java Development Tools (JDT) perspective. The task list at the bottom right of the screen is a plug-in that can be removed, not only from the screen but from the product.

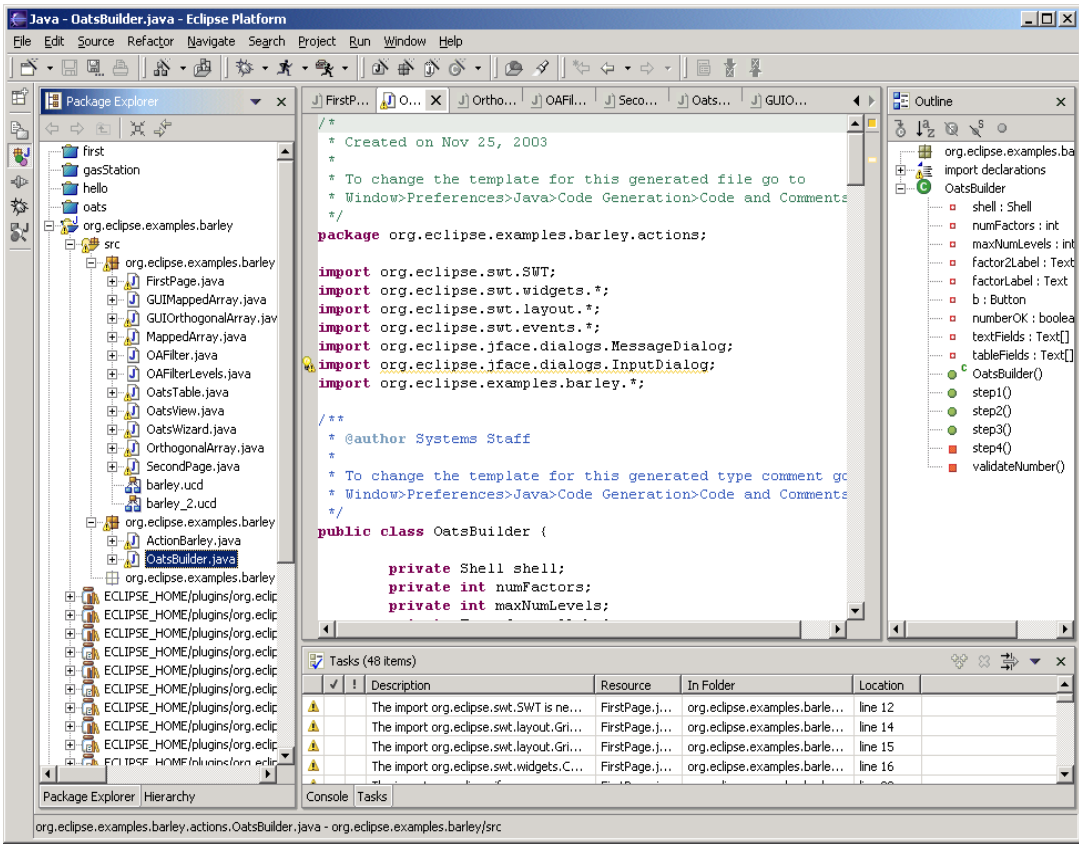
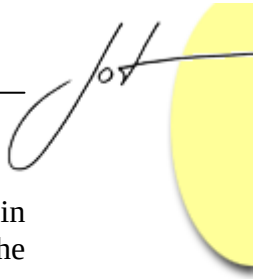


Figure 7 - Eclipse interface

The Eclipse architecture is defined to faciltate the addition of new plug-ins and the removal of irrelevant ones. A plug-in is deployed by creating a sub-directory, in the plug-ins directory, into which is placed the implementation files as well as the manifest. Plug-ins are bound to Eclipse at start time for an instance of Eclipse. The plug-ins directory is searched and each sub-directory that contains the manifest is processed to bind the new behaviors into the running instance. More details on the Eclipse project can be found at http://www.eclipse.org.

Eclipse was initiated by IBM a number of years ago and more recently was released to the open source community. The plug-in architecture is a strategic move to support a broad spectrum of development purposes. Eclipse is a part of IBM's e-commerce on demand strategy. The ease of extension provided by the architecture is intended to allow IBM to remain visible in niche markets where it can not satisfy every tool request but it can provide the basis upon which others develop their products. The open source nature of Eclipse allows IBM to create implicit relationships with other tool vendors through mutual dependence on a common architecture.

The Eclipse architecture has made possible a rapidly expanding new market for both new and existing products. A recent visit to just one web site located 50 Eclipse plug-ins available either as open source or as commercial products. For example, the latest version



of AcmeStudio [Acme 04] is now built as an Eclipse plug-in. The granularity of a plug-in allows small, non-profit organizations as well as large companies to participate in the marketplace.

Several strategies come together here. The strategy of open source is well known and provides many benefits. The Eclipse project has chosen an architecture that facilitates the integration of contributions from a diverse group. The strategy of providing a configurable tool for developers builds the community of users who will, at some point, be in the market for commercial goods and services.

5 ACTION LIST

Lets consider some steps that you might take to increase the strategic value of your software architecture.

- Develop a comprehensive, modern, validated software architecture for every product you build. Too many companies still neglect this first, and most fundamental step. Use UML, as shown in **Figure 3**, or similar architecture description notations so that the architecture description is precise and unambiguous.
- Coordinate the individual software architectures in your organization. Use software product line techniques for sets of products that are closely related. Use enterprise architecture techniques to organize in-house activities and products.
- Ensure close fit among product development activities by organizing around the architecture. Use ADD or similar architecture design techniques to establish traceability between the software architecture and corporate priorities.
- Ensure that trade-off decisions are traceable to the qualities and ultimately the strategic objectives of the organization. Provide architectural views that show the mapping between corporate strategic goals and architectural decisions and the evolution of the architecture.
- Scan the environment continually for emerging technology trends, industry consortia that form around a specific technology such as the Object Management Group (OMG) and stay current with software architectures that are defined by domain specific groups for a particular domain such as automotive or telecommunications.

6 SUMMARY

The software architecture is a key ingredient in a software-intensive product development effort. It provides a means of defining and achieving corporate strategic objectives. I have illustrated several ways in which this is accomplished. Finally I listed some possible actions that can be taken by those who wish to have a strategic impact in their company.

REFERENCES

[Acme 04] AcmeStudio http://www-2.cs.cmu.edu/~acme/AcmeStudio/AcmeStudio.html.

[Allen 94] Robert Allen and David Garlan. „Formalizing Architectural Connection", *Proceedings of the 16th International Conference on Software Engineering*, May 1994.

[Bass 03] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*, Addison-Wesley, 2003.

[Clements01] Paul Clements and Linda Northrop: *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[Hansen 03] Paul Hansen. "Software as Product". *Automotive Industries*, Oct. 2003.

[Lloyd 99] P.T.L. Lloyd and G.M. Galambos. "Technical reference architectures", *IBM Systems Journal*, v 38, n 1, 1999.

[Luckham 95] David C. Luckham and James Vera. "An Event-based Architecture Definition Language", *IEEE Transactions on Software Engineering*, v 21, n 9, pp 717 – 734, Sept 1995.

[Maccari 02] Alessandro Maccari, Galal H. Galal. «Introducing the Software Architectonic Viewpoint". *WICSA 2002*, pages 175 – 189.

[Malan 02]Ruth Malan and Dana Bredemeyer. Strategy Architect Competency Elaboration, http://www.bredemeyer.com/pdf_files/StrategyCompetency.PDF, 2002.

[Malveau 04] Raphael Malveau. "Bridginging the Gap: Business and Software Architecture, Part 1 – 4", http://www.cutter.com/research/2004/edge040113.html.

[McGregor 04] John D. McGregor: "Software Product Lines", in *Journal of Object Technology*, vol. 3, no. 3, March-April 2004, pp. 65-74. http://www.jot.fm/issues/issue_2004_03/column6

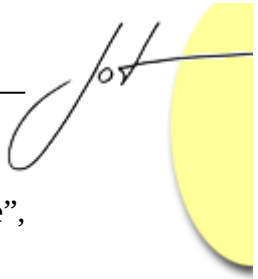
[Nokia 04] Nokia, http://www.nokia.com/BaseProject/Sites/NOKIA_MAIN_18022/CDA/Categories/Phones/technologies/

[OMG01] Object Management Group: *Model Driven Architecture*, Doc # ormsc/2001-07-01, Object Management Group, 2001.

[OMG03] Object Management Group: *OMG Unified Modeling Language Specification Version 1.5*, Object Management Group, 2003.

[Porter 96] Michael E. Porter. "What is Strategy?" *Harvard Business Review*, Nov. – Dec. 1996.

[UML 03] Object Management Group. Unified Modeling Language 1.5, 2003.



[Zachman 87] John A. Zachman. “A Framework for Information Systems Architecture”, *IBM Systems Journal*, vol. 26, no. 3, 1987.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests are software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.