IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITB12844

This paper appears in the book, *Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2* edited by Mehdi Khosrow-Pour © 2006, Idea Group Inc.

Building a Methodology for the Design of Reference Architectures that Integrates Legacy Systems

Juan Muñoz López, Arévalo Mercado, Instituto Nacional de Estadística, Geografía e Informática, Dirección General de Innovación y Tecnologías de Información, Av. Héroe de Nacozari Sur #2301, Fracc. Jardines del Parque, Aguascalientes, Ags., 20270, México, P: (52 449) 9104332, F: (52 449) 9104392, juan.munoz@inegi.gob.mx

Jaime Muñoz Arteaga & Carlos Argelio
Universidad Autónoma de Aguascalientes, Centro de Ciencias Básicas, Av. Universidad #940, Fracc. Campestre, Aguascalientes, Ags., 20100, México, Phone: (52 449) 9108417, F: (52 449) 9143222, {jmunozar, carevalo}@correo.uaa.mx

ABSTRACT

This work faces two problems originated on the organizational computational systems environment and proposes a methodology to design a reference software architecture oriented to facilitate the evolution of legacy systems and the integration of autonomous applications.

The methodology improves software architectures design process and adds elements to take advantage of the retroalimentation that comes from applying architectural models to the development of new systems. In this way it helps to create a framework that simplifies and reduces costs of evolution, integration and replacement of software systems.

INTRODUCTION

In an organization commonly are present three problems that affect its computational environment, they are: evolution and functionality enhancement of its legacy systems; integration of their systems; and designing of software architectures to solve the two problems stated before.

The first two problems begin when an area of an organization builds a system to solve its specific needs. The solution will be integrated to the area's processes and it will work on an autonomous way. During its lifetime, systems receive a lot of maintenance, first to stabilize them and later to adapt them to new requirements and changes on its environment.

Every time that a system receives maintenance, it becomes more complex. The software will reach a point where no more adaptations can be made. A system that cannot adapt itself to changes in environment will inhibit the development of the organization.

A legacy system is that which has received a lot of effort and financial resources to be maintained on working conditions, but due to its architecture it's too hard to adapt to new organization's needs.

Generally, it's not easy to replace a legacy system; a lot of issues must be considered, like: amount of resources invested, how critical is the process and how much knowledge has people about it, development team skill level, replacement process costs, etc.

ESTATE OF THE ART

Legacy software evolution

Different purposes can drive changes in a system. Usually, three strategies have been applied to evolve legacy systems [1, 3]:

1. Software maintenance. Changes are made to fix defects on software, to improve system's performance, to add functionality or to adapt it to changes in its environment; but the structure of the system remains the same.
2. Architectural evolution. The architecture of the system is modified to enhance its adaptability.
3. Software reengineering. This approach doesn't add new functionality to a system, nor modifies its architecture, but it alters its structure to make it easier to understand and to adapt.

Maintenance makes evolve a system, but it also adds complexity. As the system becomes more unintelligible, more resources and time are needed to apply new maintenance.

Software reengineering increases maintainability and functionality of a system; but, benefits of this approach will be narrow if the architecture is not modified [1, 2]. Architectural evolution increases adaptability, but it's a complex and expensive task.

Systems Integration

Generally, exchanging information among software systems that works on an autonomous way implies manual reworking. Different factors can harden this labor, like: multiple platforms, different protocols, heterogeneous data and metadata structures, diversity of contexts, domains and taxonomies, etc.

Heiler [4] defines interoperability as the capacity that distributed system components have to exchange data and services among them. When a system has been designed to interoperate, information exchange and process can get a high efficiency grade. There are different proposals to create a framework that increase interoperability between systems [5], but is still needed to develop and architecture to integrate systems, data and metadata on a holistic, modular and flexible way. This kind of architecture must allow gradual evolution and replacement of elements of the integrated system.

Eval and Milo [6] have proposed an architectural model based on wrappers to homologate and to integrate web applications using XML. But, their model doesn't support legacy systems and its scope is reduced to web applications.

Chiang [7] introduces an architectural model based on CORBA to integrate legacy software reengineering commercial systems on heterogeneous distributed environments. The architecture has been conceived to integrate systems that are similar and only unifies the final results from the systems; but it doesn't provide a mechanism to exchange information internally.

Efforts on data integration are relevant to this work too: Federated databases [8], heterogeneous database management systems [9], architectures for queries integration [10] and tools for integration from different structured and unstructured sources [11] shows different approaches that must be studied and complemented to integrate data and metadata with different structures and semantics.

Design of Reference Software Architectures

Software architecture gives elements to share a common vision of a system and to take important decisions about the quality characteristics that a system will incorporate.

Reference architecture can be described as a meta-architecture that helps to design software architectures. Reed [12] says that creating reference architectures from its basis requires of a lot of effort that combines effective tools, technology and actual approaches of the organization with a set of best practices, patterns and approaches that in the past has been successful.

There are some methodologies oriented to design a software architecture [13, 14, 15, 16] but is still needed a methodology to design reference software architectures and to evolve them taking advantage of the retroalimentation obtained when it is applied to make the architectural model of a new system.

PROPOSALS

We propose a methodology for developing and evolving reference software architectures. The methodology is focused on designing reference architectures for systems that integrates legacy and new systems and provides mechanisms to easier the evolution and replacement of the system elements.

In figure 1 we describe our proposal. It starts with a validation process of the required functionality and quality characteristics of the system. Each requirement will be assigned to a sponsor who will be responsible of ensuring that it is clearly stated on the architecture.

Non functional characteristics will be stated using a formal model that will establish the basis of a future measurement of them in the architectural model. Quality requirements will be stated in a checklist and negotiated by the stakeholders using a method of negotiation, similar to ATAM [14].

Architectural design team will transform functional and no functional requirements to architectural elements, and then, depending if exists a previous reference architecture, they will apply it to make the design or they will develop a design based on patterns and reference models.

The document and the designed architectural model must be evaluated by all the stakeholders if it is satisfactory the systems will be constructed based on it, else the architectural design team must redesign their proposal.

Once the system has been developed it will be necessary to evaluate how much the system conforms to the architectural model, variability will be an important issue to consider when the reference architecture is created or refined.

If we haven't had a reference architecture we will apply a process of analysis and extraction of the reference software architecture. It is foreseen that practices taken from production lines will be helpful on this task.

When we have had a reference architecture already, the retroalimentation mechanism acts as a permanent evolutionary mechanism that helps to maintain it current and adaptable to environmental changes.

Figure 1. Methodology to design reference software architectures

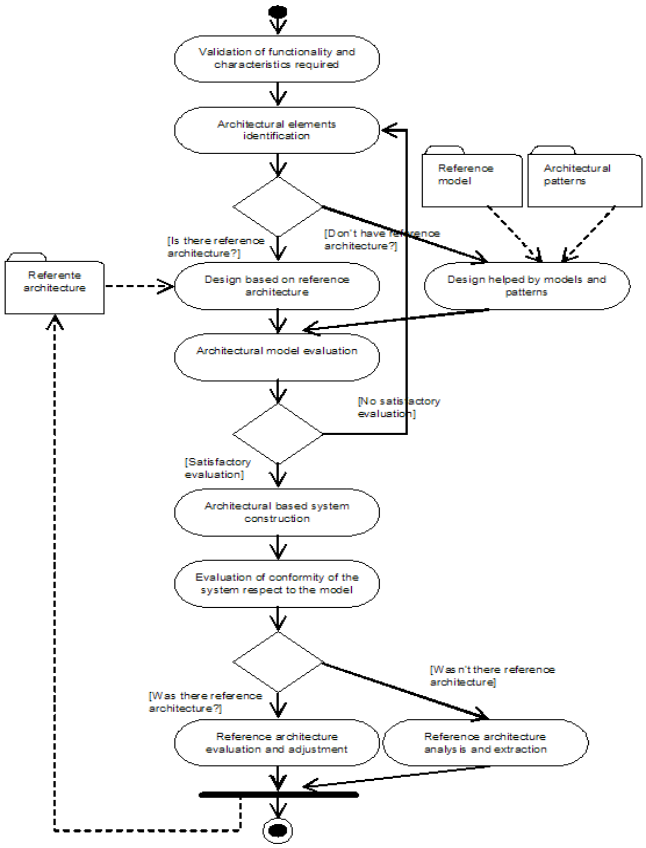
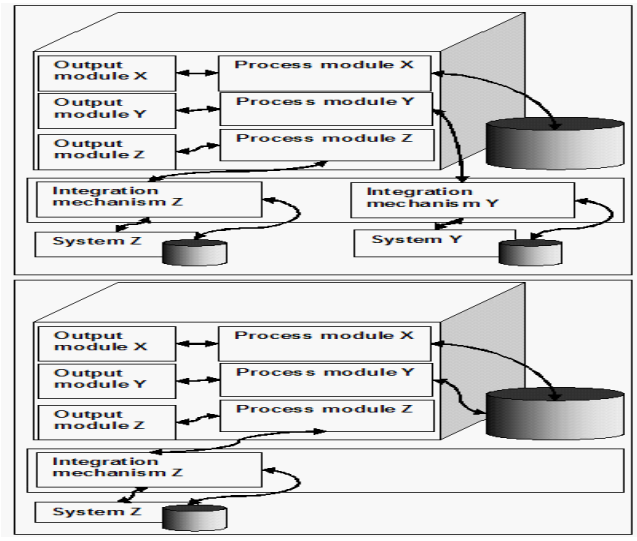


Figure 2. Integration mechanism as a result of applying methodology's first steps



CASE STUDY

In order to prove our methodology we propose to apply it to integrate systems like accounting, sales and inventory systems. Those systems works on an autonomous way on separate areas, but information produced for some of them is needed as an input for others; for example: information produced by sales and inventory systems are an input of the accounting system.

A lot of manual effort is required to transport and convert information from one system to other. Also, it's common that each system must add new reports and queries to satisfy new needs. This task generally is accomplished with artisan work using tools like spreadsheets and word processors.

If an architectural design is made under the methodology that we propose in this document, it will be established a framework that will easier integration of the systems.

We have applied the first steps of the methodology and with a design based on models and patterns we have extracted a model that integrates different systems as modules and can gradually evolve or replace them. A process module may use data and functionality of a legacy system and could replace it on the future with a new version of the system on a progressive way (see figure 2). The proposed mechanism allows us to make a smooth migration of data and functionality where the benefits of an integrated interface and of a modular architecture can be perceived from the beginning.

CONCLUSIONS AND FUTURE WORK

This work has proved that it has a sound potential to bring elements that will help to solve the problems related to evolution of legacy systems and integration of autonomous systems.

The future work is to refine, detail and complement the proposed methodology with schemas for the formal and graphical development and evaluation of the documentation, the model for valuation of non functional characteristics, the evaluation of variability, the verification of functional requirements, the abstraction, validation and evolution of the reference architecture, etc.

REFERENCES

[1] I. Sommerville, *Ingeniería de Software*, Pearson Education, México, 2002

[2] R. S. Pressman, *Software Engineering; A Practitioner's Approach, Fifth Edition*, McGraw Hill, USA, 2001

[3] O' Cinnéide, M., *Automated Application of Dessign Patterns: A Reffactoring Approach, Doctoral Thesis*, University of Dublín, Trinity College, Irlanda, Octubre 2000

[4] Heiler, S., "Semantic Interoperability", ACM Computer Surveys, Vol. 27 No. 2, junio 1995, pp 271-273

[5] Egyhazy, C., Mukherji, R., "Interoperability Architecture Using RM ODP"Comunications of the ACM, Vol 47-No 2, Febrero 2004, pp 93-97

[6] Eyal A., Milo, T., "Integrating and customizing heterogeneous e-commerce applications", The VLDB Journal 10, Springer-Verlang, USA, Agosto, 2001, pp 16-38

[7] Chiang, Ch., "A Distributed Objetc Computing Architecture for Leveraging Software Reengineering Systems", SAC 2001, ACM, Las Vegas, NV, USA, 2001, pp 653-657

[8] Sheth, A., Larson, J., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", ACM Computing Surveys, Vol. 22, No. 3, Septiembre, 1990, pp 183-236

[9] Thomas, G., Thompson, G., Chung, Ch., Barkmeyer, E., Carter, F., Templeton, M., Fox, S., Hartman, B., "Heterogeneous Distributed Databases Systems for Production Use", ACM Computing Surveys, Vol. 22 No. 3, Septiembre 1990, pp 237-266

[10] Davis, S., "SQL Integrator: A Data Request Broker for Heterogeneous Data Access", Novell App Notes, Novell, Inc., USA, Mayo 1998, pp 57-70

[11]Williams, D., Poulovassilis, A., "An Example of the ESTEST Approach to Combining Unstructured Text and Structured Data", School of Computer Science and Information Systems, Birkbeck College, University of London, Inglaterra, 2004, pp 1-5

[12] Reed, P., "Reference Architecture: The Best of Best Practices", the Rartional Edge, Rational Software-IBM, USA, Septiembre 2002, pp. 1-14

[13] Garland D., Shaw, M., "An Introduction to Software Architecture", CMU-CS-94-166, School of Computer Science, Carnegie Mellon University, Pittsburg, PA, USA, Enero 1994, pp 1-39

[14] Bass L., Clements, P. y Kazman R., Software Architecture in Practice. Second Edition, Adison Wesley, U.S.A., 2003

[15] Mehta, N., Soma, R., Medvidovic, N., "Style-Based Software Architectural Compositions as Domain-Specific Models", USCCSE 2004, Department of Computer Science, University of South California, Los Angeles, CA, USA, 2004, pp 1-8

[16] Albin, S., "The Art of Software Architecture", Wiley Publishing Co., USA, 2003