

System Development Using the Integrating Component Architectures Process

J. Payton, R. Keshav, and R.F. Gamble

University of Tulsa

600 South College Avenue

Tulsa, OK 74104 USA

+1 918 631 2988

{payton, reshma, [gamble](mailto:gamble@euler.mcs.utulsa.edu)}@euler.mcs.utulsa.edu

ABSTRACT

The purpose of this paper is to present a software development process that considers interoperability problems associated with integrating in-house systems with COTS products. The Integrating Component Architectures Process (ICAP) outlines a process that includes predicting interoperability conflicts among independent components, COTS included, based on architectural differences. Previously defined integration elements, along with a taxonomy of integration solutions based on these elements, direct movement through the process to an integration architecture for implementation. The ICAP predominantly falls into the elaboration phase of the Rational Unified Process, providing greater detail of the issues surrounding interoperability. We believe that such a development process will allow easier and more cost-effective maintenance and migration of legacy systems, as well as the use of COTS products in system development.

Keywords

Integration, Rational Unified Process, interoperability, taxonomy, COTS

1 INTRODUCTION

COTS products are normally well tested and can provide a reliable system that meets the needs of the users. Recent developments in software engineering include the use of COTS products with custom or pre-existing software systems to enhance functionality. However, integrating the existing system with a COTS product has proven to be complicated and expensive since reuse is feasible only if existing components can be easily adapted for interoperability. Also it can be a formidable task to discover the correct solution to interoperability problems since published descriptions of these solutions often

overlap or they may be language dependent.

Fortunately, many of the interoperability problems can be predicted if the overall system is evaluated at the architectural level before implementation. Architectural analysis can detect potential conflicts and guide the developer towards the resolution of the problems. It can also reduce the cost and increase the stability of the system being integrated, as the use of ad hoc methods of integration during implementation are minimized. Although current accepted software engineering development processes exist that consider the inclusion of COTS products with a software system [1, 5], they do not provide a method to analyze components with respect to integrating COTS products with other systems.

In this paper, we introduce a software design process that provides for the early architectural analysis of software systems and in turn provides a better method to integrate COTS products with pre-existing systems. We called this method the Integrating Component Architectures Process (ICAP).

2 LIFE CYCLES

The Rational Unified Process [5] and the MBASE life cycle [1] both provide a method to systematically build high quality software. Because of their similarities, we use the Rational Unified Process in our discussions.

The Rational Unified Process is divided into four phases in which iteration and incremental development are used to achieve the milestone requirements. These phases are:

Inception. The success requirements, risk assessment and resource estimation are established. The Lifecycle Objectives (LCO) requirements should be met before continuing to the next phase.

Elaboration. An architectural foundation for the system is established, a project plan is developed, and risks are resolved. The Lifecycle Architecture (LCA) requirements should be met before continuing to the next phase.

Construction. The components and features are implemented and tested. Before continuing to the next phase the Initial Operational Capability (IOC)

requirements should be met.

Transition. The product is passed on to the user for further testing. The Product Release (PR) requirements should be met.

In order to meet the LCA requirements in the elaboration phase, the architecture of the product should be stable, major risks should be eliminated, and detailed plans for the implementation of the system should be in place. We use the Integrating Component Architecture Process (ICAP) to facilitate fulfilling part of the LCA requirements. The ICAP consists of three major steps whose titles are borrowed from database integration research [4]: *pre-integration*, *correspondence identification* and *integration*. The ICAP meets part of the LCA requirements by detecting interoperability problems at the architectural level, thereby evaluating certain risks, providing an implementation guideline for overcoming those risks, and establishing a stable integration architecture.

4 ICAP

Figure 1 illustrates the use of the ICAP. Imagine a company has an in-house payroll system (IH) that they want to extend with a custom retirement contribution package (E), which is still in the design phase. Integrating a commercial tax software package requires choosing between two different products that are equivalent in cost and overall functionality (COTS A and COTS B). The entire system should be web-based, so development to construct the GUI (SE) is in the design stages.

Pre-integration. Characteristics, such as those defined by Shaw and Clements [6], may be used to describe the architectures of each participating component. First, developers must compare the characteristics and use them to determine interoperability problems. Partial architectural characteristics must suffice as full descriptions are often unavailable. Currently, detection of these problems is based upon previous case studies, research, and experience. For example, if the data flow of IH in Figure 1 is shared, then designing E with passed data would incur unnecessary interoperability conflicts. Likewise, a COTS product that can obtain data from an external repository might be easier to integrate than one that requires all data to be passed to its functions directly. More research is needed to establish exactly which characteristics contribute the most to interoperability and why.

Once the conflicts are known, a plan to solve the problems can then be discovered. In our example, E and SE are still in their design phases, so changes can readily be made to them to avoid predicted conflicts. Unfortunately, manipulating the design and code in COTS products is not always an option. Therefore, a plan for integration must include external additions to the system

components in the form of integration elements [3], on which the foundations of interoperability solutions are based.

Given known conflicts from architectural characteristics, architecture integration elements can be combined to form an overall solution strategy. By bringing the strategies, the system and the conflicts to the same architectural level the developer can prescribe the underlying integration architecture that is needed to solve known interoperability problems. With this approach, there is less surprise on the part of the implementor to handle particular interoperability problems.

Correspondence Identification. Given a previously defined taxonomy of popular integration strategies and the results from the pre-integration stage, the developer is now aware of which integration elements are needed to achieve interoperability and can use the taxonomy to identify strategies that contain the specified elements. Though choices are narrowed, the developer may find that there are several applicable solutions.

Choosing an integration strategy is the final step in this phase. In order to make a better judgement the developer must iterate over the requirements defined during the inception phase. Using this information, the developer can study the potential strategies and choose one from the taxonomy that is best suited to the overall problem. By bringing the systems, the conflicts, and the solution to a common denominator, it has not only made the process of finding a suitable strategy much simpler, but it will also increase reuse of both tested code and proven strategies.

For instance, assume that COTS B in Figure 1 requires passed data. Then a controller integration element [3] can be used to choose and obtain the correct data from the repository in IH. Multiple translator integration elements [3] would place the data in the correct form to and from the COTS product and the repository. Integration strategies that embody this functionality would be chosen from the taxonomy. The choice might depend on implementation platform, available resources, and requirements such as performance and reliability. If COTS A had the ability to interface directly with the repository and use the shared data, it would be a better choice with respect to interoperability.

The ICAP aids the LCA requirements to ensure a more stable integrated system architecture through highlighting interoperability conflicts and then providing a foundation and process to eliminate these risks. By understanding these architectural requirements, the developer can more aptly choose the appropriate COTS product for the final system.

Integration. The above phases provide an integration plan that helps the developer transition to the construction phase of the Rational Unified Process which embeds the

final phase in the ICAP. The ICAP integration phase involves implementation, evaluation and testing of the system to determine if the chosen strategy is correct, meets the interoperability needs, and adheres to performance requirements.

As long as the developer adheres to the possible solutions given through the correspondence identification phase that embody the key integration elements, the architectural foundation needed to resolve existing conflicts will be maintained. However, system architecture design changes will require iteration through the ICAP.

5 CONCLUSION

Integrating software systems with COTS products has become increasingly important to reduce cost and time-to-market. For the most efficient design and implementation, software developers should use a development process, which considers the integration of COTS products. The process should include a phase in which the involved systems are considered at an architectural level. Distilling information about the architectural characteristics allows for pinpointing interoperability problems at a high level of abstraction [2]. At this level of abstraction, implementation details need not be involved in decision making. The ICAP introduces a process on which to perform this analysis. Once the applicable integration elements are determined, a taxonomy is used to assist developers with the difficult task of choosing an interoperability solution. Finally, an integration architecture is designed, implemented, and evaluated. We do not propose to restructure the entire software design process, but to infuse our technique into the existing lifecycles.

Acknowledgement. This research is supported in part by AFOSR (F49620-98-1-0217). The US Govt. is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright violation therein[†].

REFERENCES

- [1] B. Boehm, D. Port, A. Egyed, and M. Abi-Antoun. The MBASE Life Cycle Architecture Milestone Package, *1st Working International Conference on Software Architecture*. Feb. 1999.
- [2] D. Garlan, R. Allen, and J. Ockerbloom, Architectural mismatch or why it is so hard to build systems out of existing parts, *Proceedings of the 17th International Conference on Software Engineering*, April 1995.
- [3] R. Keshav and R. Gamble. Towards a Taxonomy of Architecture Integration Strategies, *3rd International Software Architecture Workshop*. Nov. 1998
- [4] C. Parent and S. Spaccapietra, Issues and approaches of Database Integration. *Communications of the ACM* 41(5): 166-178. (1998).
- [5] Rational Software. Rational Unified Process: Best Practices for Software Development Teams. http://www.rational.com/sitewide/support/whitepapers/dynamic.jtmpl?doc_key=100420
- [6] Mary Shaw and Paul Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. *Proc. COMPSAC97, 1st Int'l Computer Software and Applications Conference*. August 1997, pp. 6-13.

[†] Disclaimer – The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the AFOSR or the US Govt.

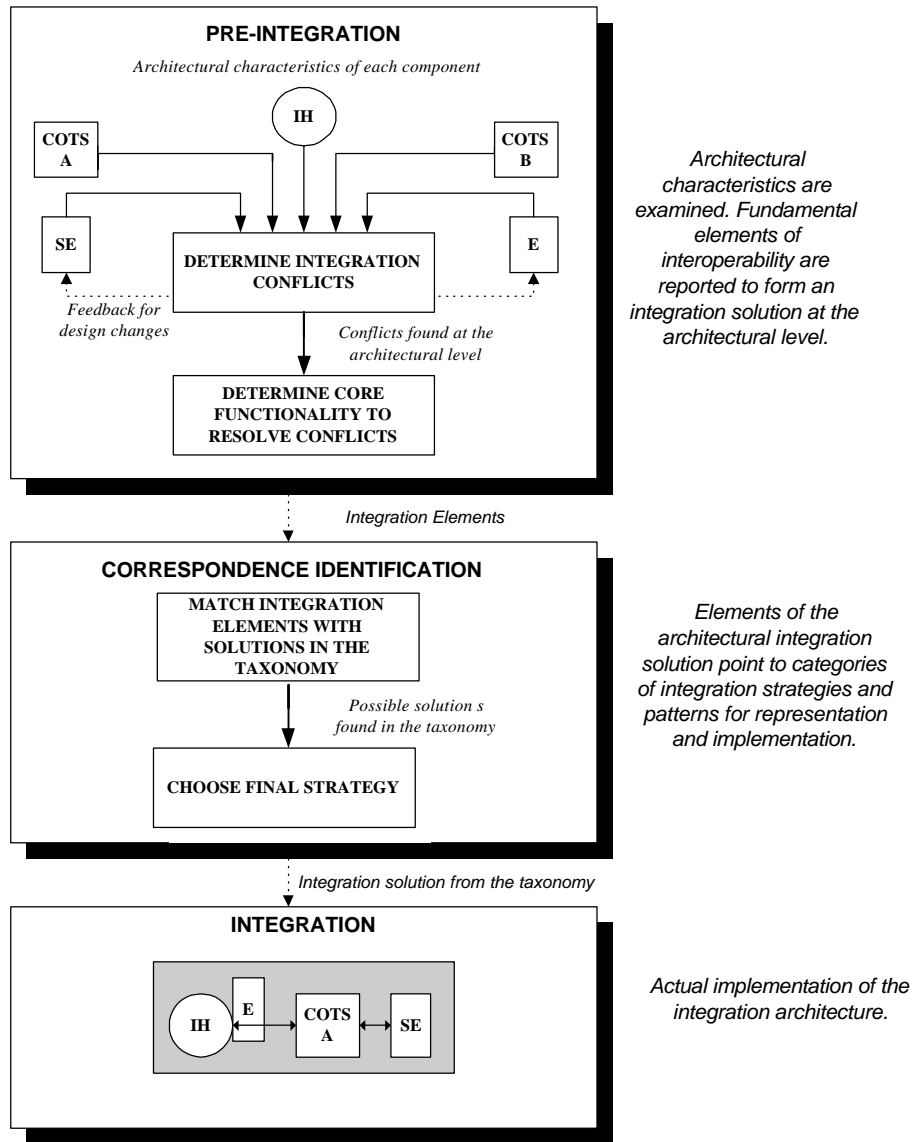


Figure 1: Integrating Component Architecture