Accepted to the 3rd International Software Architecture Workshop, Nov. 1-2, 1998.

TOWARDS A TAXONOMY OF ARCHITECTURE INTEGRATION STRATEGIES

R. Keshav and R. Gamble
Mathematical and Computer Sciences
University of Tulsa
600 S. College Avenue
Tulsa, OK 74104 USA
+1 918-631-2988
{reshma, gamble}@euler.mcs.utulsa.edu

1. ABSTRACT

By detecting interoperability problems at the architectural style level, properties of an integrated system can be defined early in the design process. There are many integration strategies, but their definitions do not provide the developer with a foundation to select a strategy for its appropriateness in implementing a correct solution. In this paper, we discuss our preliminary findings from architectural style integration analysis. We form a partial integration taxonomy that shows the relationship between existing integration strategies and the integration elements defined at the architectural level.

1.1 Keywords

Interoperability, architectural styles, taxonomy, integration.

2. INTRODUCTION

The importance of software component integration has become widely recognized with the migration of legacy systems, web-based computing, and the year 2000 problem. As with most component integration, interoperability problems arise. It is clear from research and empirical studies that interoperability problems can be detected at a high level of abstraction when integrating software architectural styles [1, 5, 7, 14, 15]. Once the problems are detected, interoperability issues can become part of the early design decisions made for architectural integration. This abstract level of design allows for integrated system properties to be defined and later ensured. Once the integrated system architecture is in place, more specific strategies can be analyzed uphold the desired properties. Because there may be multiple, composite, even competing, implementation solutions for a particular interoperability problem, it can be difficult to determine which are applicable. Thus, proper integration design decisions made at the software architecture level can pinpoint the target functionality required to alleviate these implementation problems.

LEAVE BLANK THE LAST 3.81 cm (1.5")
OF THE LEFT COLUMN ON THE FIRST PAGE
FOR THE COPYRIGHT NOTICE

There are many reliable, well-defined integration strategies, whose details may be in the form of patterns, processes and techniques, that are scattered across the literature with little relationship information. For many, the definitions do not provide the developer with a foundation to select a strategy for its appropriateness in implementing a correct solution for the integrated architecture design. In this paper, we discuss our preliminary findings from architectural style integration analysis. We present an initial set of integration elements that form the foundation for resolving interoperability problems that result from integrating architectural styles [2, 14]. Using literature on software component integration, e.g., [4, 5, 6, 7, 9, 15], and design patterns, e.g., [3, 8, 11, 12], we form a partial integration taxonomy that shows the relationship between existing integration strategies to the base functionality defined at the architectural level.

3. THE INTEGRATION TAXONOMY

3.1 Basic Elements of Integration

As a result of experimental analysis of integrating architectural styles, we partition the functionality defined within an integration strategy into three basic *integration elements*. These elements are: *Translator*, *Controller*, and *Extender* as depicted in Figure 1. A Translator converts

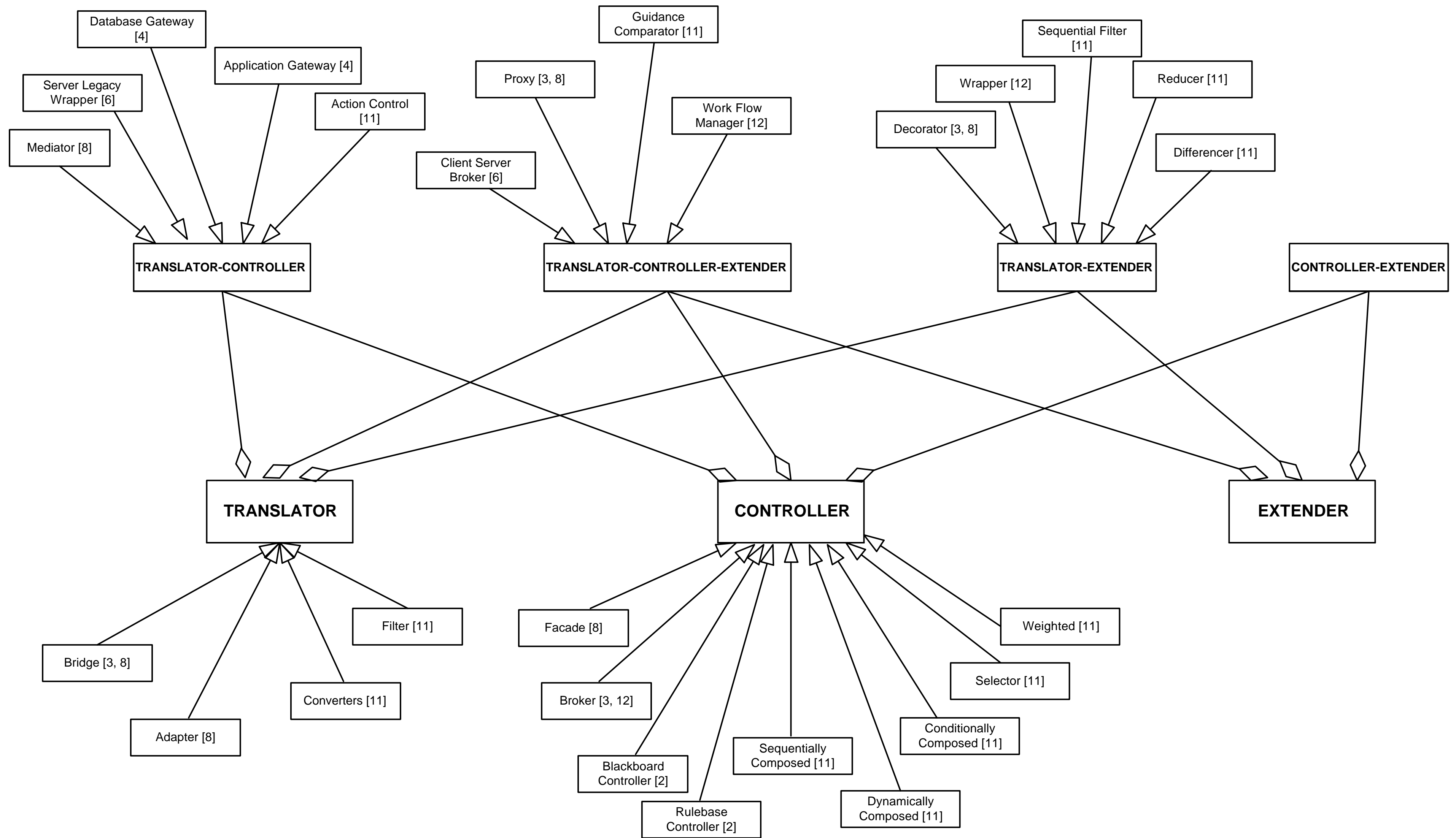

Figure 1: A Taxonomy of Potential Solutions for Architecture Integration

data and functions between component formats, without changing the content of the information. It does not need knowledge of where the data came from or where it is sent. A Controller coordinates and mediates the movement of information between components using some predefined decision-making process. The Controller needs to know the component identities for which decisions are made. An Extender adds new features and functionality, therefore enhancing component capability. Whether or not the Extender needs to know the identity of the components with which it interacts depends on the specific application. Components and connectors for each of the integration elements can be formally defined at the level of abstraction of an architectural style to depict its functionality [2].

During architecture integration design, it may be necessary to compose a strategy that uses more than one of the above integration elements. For example, if two components with different data formats need to share a repository, then both a Translator (to neutralize the different data formats) and a Controller (to mediate access to the repository) would be required in the high-level design. Thus, we include combinations of multiple integration elements to form a composite integration element, whose name in the taxonomy (Figure 1) is simply the concatenation of the elements employed.

3.2 Using a Taxonomy

Using IHArch, an experimental platform developed for analyzing architecture style integration [14], we determine potential integration strategies that resolve the interoperability problems detected between two architectural styles. The results from the architectural style integration analysis can be used to incorporate specific functionality into the integrated system design to overcome known problems. In addition, the results can be used to guarantee properties of the integrated system architecture to be upheld by implementation choices. In cases where there are competing products to be integrated with a legacy system, interoperability analysis can result in choosing the better product [14]. Once the base integration elements are determined, the taxonomy can be used to target suitable strategies for implementation.

Consider integrating an object-oriented architectural style with an event-based style. Given that the object-oriented style has explicit invocation of methods via message passing and the event system style has implicit invocation of processes via event announcements, an interoperability problem can result. This conflict can be resolved by using a Translator to change the invocation of one system and a Controller to determine its correct path. Another style characteristic that may conflict is the control structure. In this example, both are decentralized, but one uses an event bus, while the other uses procedure calls. To resolve the conflict, a Translator, Extender, and a Controller may be needed. In order for the Controller to synchronize the passing of information, the object-oriented system needs to be wrapped to form a central point for its control flow, while still working independently of the event system. This example, taken from [6], implements the Wrapper and Broker pattern from [12] to create a web-based interface for a legacy system.

3.3 Relating Integration Strategies to the Taxonomy

In this section, we highlight certain integration strategies and how their functionality places them within the taxonomy displayed in Figure 1. We use the term integration strategy to encompass processes, techniques, solutions, gateways and patterns that address issues in software component interoperability. An integration strategy is considered for the taxonomy if it (1) states that it resolves an interoperability problem, (2) is related to an accepted integration strategy, (3) describes a mechanism to decouple functionality, increase functionality, or develop a new interface for a component, and (4) has functionality that is statically well-defined. Due to space, we are unable to define and justify the placement of each strategy as shown in Figure 1. This information can be found in [10]. However, we define one typical integration strategy for each integration element and composite integration element, where applicable. Integration strategies are placed in the taxonomy with respect to how they resolve interoperability problems, but are not restricted to having additional functionality and implementation considerations outside of integration.

Translator. One example of a Translator is a Bridge [8]. A bridge converts data from one form to another without knowing its clients, therefore, allowing the clients to vary independently.

Controller. A Broker is a typical Controller [3, 12]. It explicitly coordinates communication between interoperating applications.

Extender. We have yet to find any strategies that only fall into this category. One reason may be that adding new functionality is almost its own integration problem, requiring the use of a Controller or Translator.

Translator-Controller. A Mediator [8] is an example of a Translator-Controller. A mediator promotes loose coupling by keeping objects from referring to each other explicitly. It is responsible for controlling and coordinating the interactions of a group of objects.

Translator-Controller-Extender. The Proxy [3, 8] is a good representation of a Translator-Controller-Extender composite integration element. It has the ability to translate information and perform pre and post data processing. It controls access to an object in order to defer the full cost of its creation and initialization until it is

actually needed. It can be built to protect components from unauthorized access.

Translator-Extender. The Wrapper [12] is a pattern that is composed of a Translator and an Extender. It enables the developer to add new functionality to a legacy system that needs to work both independently and with another system. It is designed so that the old and new functionalities are transparent to the user.

Controller-Extender. We have yet to find any documented strategies that only fall into this category.

3.4 Loosely Defined Integrated Strategies

The Shared Repository pattern [12], the Information System gateway (IS Gateway) [4] and the mediation process [9] are examples of loosely defined integration strategies, such that each could be composed of one or more of the integration elements in Figure 1. One of the criteria we use to place the integration strategies into the taxonomy is that they need to have certain static requirements. Since the composition of integration elements in the Shared Repository, IS Gateway, and mediation process can vary broadly, the developer can dynamically choose what should be implemented as part of the strategy without changing its initial definition. Thus, these strategies are not represented explicitly within the taxonomy.

4. DISCUSSION AND CONCLUSION

The main functional elements, *Translator, Controller*, and *Extender* and their compositions within the taxonomy are found to be inherent to architectural style integration. The elements are derived from experiments using IHArch system that embeds conflicts and solutions to interoperability problems associated with integrating specific characteristics of the architectural styles Pipe & Filter, Event System, Object-Oriented, and Main/Subroutine [17]. An examination of other architectural styles, such as Repository and Rule-based System is underway. This future research may indicate additional functional elements for integration, as well as to fine-tune the existing elements. Additional empirical studies are being examined for a more robust integration taxonomy.

Acknowledgement. This research is supported in part by NSF (CR-9708643) and AFOSR (F49620-98-1-0217). The US Govt. is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright violation therein†.

† 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.

5. REFERENCES

[1] A. Abd-Allah, Composing Heterogeneous Software Architectures, Ph.D. Diss., Dept. of CS, USC, 1996.

[2] P.R. Butler, An Assessment of Architectural Styles and Integration Components, M.S. Thesis, Dept. Math/CS, Univ. of Tulsa, 1998.

[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons Ltd. New York, NY, 1996.

[4] M. Brodie & M. Stonebraker. *Migrating Legacy Systems: Gateways, Interfaces & the Incremental Approach*. Morgan Kaufmann Publishers, Inc. San Francisco, CA, 1995.

[5] J. Callaham & J. Purtilo, Using an architectural approach to integrate heterogeneous, distributed software components, ftp://research.ivv.nasa.gov/pub/doc/TRs/tse94.ps.

[6] D. Cole & R. Gamble, Integration strategies for the web, TR UTULSA-MCS-98-10, Dept. MCS, Univ. of Tulsa, 1998.

[7] D. Garlan, R. Allen, & J. Ockerbloom, Architectural mismatch or why it is so hard to build systems out of existing parts, *Proc. of the 17th Int'l Conf. on Software Engineering*, April 1995.

[8] E. Gamma, R. Helm, R. Johnson, & J. Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[9] Int'l Telecomm. Union, Principles for a Telecommunications management network, ITU-T M.3010, 1996.

[10] R.M. Keshav & R.F. Gamble, Defining a Taxonomy of Integration Strategies, TR UTULSA-MCS-98-6, Dept. MCS, Univ. of Tulsa, 1998.

[11] D. Lea. Design Patterns for Avionics Control Systems. SUNY Oswego & NY CASE Center, DSSA Adage Project ADAGE-OSW, 1994.

[12] D. Mularz. *Pattern-based integration architectures*. PLoP, 1994.

[13] M. Shaw & D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[14] R. Sitaraman & R.F. Gamble, Choosing interoperable components, *USENIX Adv. Topics Wksp on Software Components: Integration & Collaboration*, 1997.

[15] W. Tracz & L.Coglianese An adaptable software architecture for integrated avionics, IBM Corporation, Federal Systems Company, 1993.