A Method and Case Study of Designing Presentation Module in An SOA-based Solution Using Configurable Architectural Building Blocks (ABBs)

Liang-Jie Zhang*, Jia Zhang**¹, Abdul Allam***

*IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532

**Department of Computer Science, Northern Illinois University, USA

*** IBM Global Business Services, USA

zhanglj@us.ibm.com, jiazhang@cs.niu.edu, allam@us.ibm.com

Abstract

Service-Oriented Architecture (SOA) projects typically pose challenging requirements on software developers such as flexibility, speed to market, and adaptability to ever changing business requirements. In this paper, we propose a novel method of rapidly designing presentation module for an SOA-based solution using fine-grained configurable architectural building blocks (ABBs). An ABB-based presentation pattern derived from industry best practices is presented, associated with a tailored ABB configuration framework. We also introduce a formal way of modeling ABBs and their interactions. Real-life experiences of applying our ABB-based model are condensed into sixteen architectural decision points, which are further applied into a variety of SOA solution projects. Experiences accumulated over these decision points are summarized into a set of guidelines to help engineers select and configure ABBs specific to services scenarios and requirements.

1. Introduction

Service-Oriented Architecture (SOA) has emerged as a powerful model that opens a new cost-effective way of engineering software to develop and deploy various applications (e.g., Web applications), by dynamically integrating other existing service components into new business processes [5]. Business applications designed and developed using the SOA technology are usually called SOA solutions or services solutions.

In an SOA solution, a presentation module is typically identified as a dedicated component to handle all interactions with end users via a Web interface or a program interface, as well as interactions with back-end systems or other services to retrieve data to serve user requests. Acting as the linkage between back-end data or services and front-end user interfaces, the presentation module is a critical component of most SOA solutions.

The traditional way of designing a presentation module is based on project-specific requirements. Such a presentation module developed may not be reusable for another system. Furthermore, it may lack flexibility and extensibility in reconfiguring its building blocks to adapt to business requirement changes. Therefore, many new techniques have been created to build a more reusable and adaptable presentation module in a business solution. They are generally categorized into two types: technology-driven approaches (e.g., JavaScript [1], Asynchronous JavaScript And XML (Ajax) [2], and Portlet [3]) and architecture-driven approaches (e.g., Model-View-Controller (MVC) model [4]).

The SOA paradigm stimulates the present market place to increasingly demand a services solution to possess flexibility, speed to market, and adaptability to ever changing business requirements. Technology-dependent solutions and current architecture-based solutions do not sufficiently address these needs. Therefore, there is a necessity and demand for creating a method of configuring flexible and extensible presentation patterns oriented to SOA solutions.

This paper proposes a novel method featured by unified presentation and definition of Architectural Building Blocks (ABBs) and an associated configuration framework to support rapid construction of a presentation module in an SOA-based solution. This configuration capability is effective since the ABBs are fine grained: they can be configured based on specified rules and enabled by a novel configuration framework comprising a set of architectural patterns. The ABBs are identified based on service scenarios rather than technologies; therefore, they help in better aligning business needs with IT environments. This research creates an opportunity to offer new services around service enablement based on abstract building blocks residing at a higher level of abstraction than service components.

The remainder of this paper is organized as follows. In Section 2, we discuss related work. In Section 3, we

¹ The major work of this research was conducted while Jia Zhang was an Academic Visitor at IBM T.J. Watson Research Center.

introduce the concept of ABB and propose an ABB-based presentation pattern. In Section 4, we propose a formal way of modeling ABBs. In Section 5, we present a configuration framework. In Section 6, we present a case study resulting in guidelines for presentation module-oriented ABB configuration and management. In Section 7, we draw conclusions.

2. Related work

In recent years, certain techniques have been created to help build a more reusable and flexible presentation module in a business solution. Most of them are technology driven such as JavaScript [1], Asynchronous JavaScript And XML (Ajax) [2], and Portlet [3]. JavaScript is a scripting language adding interactivity to static HTML pages. A JavaScript comprises lines of executable code embedded in HTML code. Ajax is a technique that allows for creating better, faster, and more user-friendly Web applications. With Ajax, a front-end JavaScript uses a special XMLHttpRequest object to directly communicate (trade data) with a back-end server, typically for a particular portion on the page, without reloading the entire page. A portlet is represented by a window on a composite Web page, which is associated with specific business logic and directly communicates with back-end data. Each portlet is in charge of a fragment of markup as a part of the entire page. These technology-driven techniques suffer from their dependency on specific technologies. When technologies evolve, corresponding presentation implementations have to undergo consequent changes.

Another category of presentation module design is referred to as architecture-driven presentation approaches, center around the well-known Model-View-Controller (MVC) model [4]. It decouples code for data access, business logic, and data presentation and user interaction into three loosely coupled layers of components. Literature has witnessed numerous examples where the MVC model is used to successfully build Web applications, such as [6-8]. In addition, many applications extend the basic MVC model for a variety of purposes. For example, Rehman et al. [9] integrate context concerns into the MVC model. A controller may dynamically bind with some context model to better manage application views at a particular moment. However, the common issue of these MVC-centered models is that they are generally too coarse grained. Therefore, they can only provide very high-level guidance for developers to design a presentation module.

Model-driven methodology has been extensively used to simplify and automate the development of Web applications, especially for alleviating the complexity of platforms and expressing domain concepts effectively

[10]. Among the numerous efforts, Belhajjame and Embury [11] propose a model-driven approach that automatically generates a composite service specification from a declarative definition. Benatallah et al. [12] adopt a model-driven framework to manage the lifecycle of Web services-based system. Gray et al. [13] use model transformation and aspect-oriented techniques to automate design decision explorations. HDM [14] presents a front-end presentation model derived from the famous Entity-Relationship Model [15] for hypermedia application design, which divides conceptual schema into two categories: structural and navigational. AutoWeb [16] proposes a variant notation of HDM called HDM-lite, by adding a presentation schema, to conceptually design Web application and store conceptual schema together with data content in databases. In contrast with these approaches, our research intends to develop a fine-grained, configurable, and extensible model-driven framework for building SOA-based presentation modules.

3. Architectural Building Blocks

Since our research focuses on investigating development of a presentation module in the context of SOA solutions, it is worthwhile to briefly examine the relationships between a presentation module and other components inside of an SOA solution.

Figure 1. Presentation Module ABBs.

As shown in Figure 1, the presentation module typically interacts with other modules in an SOA solution reference architecture [17], which can be logically organized into (1) business process layer that composes and choreographs service components, (2) service layer that manages business services, (3) integration layer that mediates, routes, and transports service requests from service consumers to proper service providers, (4) Quality of Service (QoS) layer that provides QoS management in various aspects, (5) data layer that provides unified representation and

enablement frameworks integrating with domain-specific data architecture, and (6) governance layer that provides design guidance to ensure proper design of the SOA solution.

To simplify the presentation, we will assume in the rest of the paper that a “layer” is realized by a “module” by the same name, and will use the two terms interchangeably.

3.1. Introduction of ABB

To enable configurable and reusable presentation module design and development in a systematic manner, we introduce the concept of Architectural Building Block (ABB) as the fundamental unit of a presentation module. An ABB is an autonomous component that encapsulates internal states and functions and can be configured and extended. A functioning presentation module is comprised of multiple ABBs, which collaborate to carry all presentation activities and organize them in an orderly manner. ABBs interact with each other through message passing. Some ABB interactions may be internal to a presentation module; some may interact with outside modules (other modules in an SOA solution system) and external users (customers). Our ABB-based presentation module is event-driven, meaning that all ABBs are triggered when events happen (e.g., a user requests a quote).

Our ABB model provides a uniform mechanism for building an SOA-based presentation module, in the sense that it is capable of modeling different service scenarios, i.e., synchronous, asynchronous, parallel, serial, and nested presentation paths, in a uniform way with configurability and extensibility. In accordance with our ABB model, a formal presentation module can be modeled mathematically as a set of interrelated ABBs, each carrying a piece of work that contributes to the goal of presentation scenarios.

3.2 ABB pattern for SOA-based presentation module

We propose an ABB pattern, tailored to SOA solutions, for building presentation modules. The critical criteria are high coherence inside of an ABB and low coupling between ABBs. In other words, the defining of ABBs breaks the responsibilities of a presentation module into sub responsibility areas, each providing a distinct logical group of related cohesive functions.

As shown in Figure 1, we identify eight fundamental ABBs for a presentation module: (1) consumer, (2) presentation (view), (3) presentation controller, (4) consumer profile, (5) access control, (6) format transformation, (7) configuration rule, and (8) cache.

A *consumer* building block represents an external user of the presentation layer. It can be either a program or an individual who requests a service. A *presentation (view)* building block is responsible for receiving input via Query from a consumer and providing or displaying response to a consumer. In other words, it is responsible for communicating to and from external consumers. A *presentation controller* building block manages the navigation logic for consumer interactions. It is the coordinator for all other ABBs in the presentation layer. A *consumer profile* building block is responsible for getting customer-specific information (enabled through the data layer) to be used by the presentation controller for navigation and content presentation purposes. An *access control* building block provides access to authentication/authorization capabilities (enabled through the QoS layer) to be used by the presentation controller to allow/prevent what content can be presented to the consumer. A *format transformation* building block is responsible for translation of Query content format required by the integration layer and to convert content returned from the integration layer to a consumer response format. A *configuration rule* building block is responsible for hosting rules that dictate how the ABBs can be configured based on consumer request scenarios. This unit enables presentation configuration on demand. It also allows the use of only appropriate ABBs. A *cache* building block is responsible for temporarily storing consumer interaction-related data (e.g., a customer profile) to enhance system performance. These data typically are maintained for the duration of the entire interaction.

Figure 1 also shows the interaction points between the consumer module and other modules in an SOA solution. The *consumer profile* and *cache* build blocks interact with the data layer; the *access control* building block interacts with the QoS layer; the *format transformation* building block interacts with the integration layer; the *presentation control* building block interacts with the business process layer, the service layer, and the governance layer.

4. Modeling of ABBs

To further enhance reusability and configurability of ABBs, we provide a uniform and extensible way to model traceable information that can be carried by ABBs. All ABBs are modeled as individual Web services that can be universally accessible and interoperable. To better manage states of ABBs, we define ABBs as universal resources by applying the Web Services Resource Framework (WSRF).

Figure 2. Hierarchical ABBs.

Initiated by IBM, Computer Associates, Oracle, and other collaborators, WSRF [18] defines a system of specifications for managing and accessing stateful resources using Web services. To date, WSRF has been extensively used for enabling stateful Web services. In short, WSRF is a reputable XML-based presentation method to capture resources. It enables access to internal states of a resource via Web service interfaces, i.e., data values that can persist across and evolve as a result of Web service interactions.

To model ABBs with further reusability, flexibility, and extensibility, we define ABBs in a hierarchical manner. As shown in Figure 2, various types of ABBs are organized into a hierarchical inheritance tree-like structure. The root of the tree represents the generic ABB. Its direct-child level comprises various modules' ABBs (i.e., module-ABB) in an SOA solution. For example, the SOA solution may identify modules such as *presentation module*, *business process module*, and other modules. A dedicated ABB type is constructed to represent each module, such as *PresentationLayerABB*, *BusinessProcessLayerABB*. For each module, identified ABB types extend the corresponding module-ABB with specific features. For example, as shown in Figure 2, the *PresentationLayerABB* is extended into eight ABBs: *ConsumerABB*, *PresentationABB*, *PresentationControllerABB*, *ConsumerProfileABB*, *AccessControlABB*, *FormatTransformationABB*, *Cache*, and *ConfigurationRuleABB*.

With the organized hierarchy of ABBs, we model the ABBs in an incremental manner. We first model the generic ABB, then we model the module-ABBs by leveraging the generic ABB. Afterwards, we can leverage a module-ABB to model its extended ABBs. The following segment illustrates a part of the resource properties document definitions for *ABBResource*.

```

<xsd:schema
  targetNamespace="http://sc.org/abb/ABBResource"
  xmlns:tns="http://sc.org/abb/ABBResource"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

```

```

...
<xsd:element name="ABB">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:ABBIId"/>
      <xsd:element ref="tns:ABBName"/>
      <xsd:element name="ABBLayer"
        type="abbLayerEnumeration"/>
      <xsd:element name="ABBState"
        type="abbStateEnumeration"/>
      <xsd:element name="ABBPProtocol"
        type="abbProtocolEnumeration"/>
      <xsd:element name="ABBInputDataType"
        type="dataTypeEnumeration"/>
      <xsd:element name="ABBOutputDataType"
        type="dataTypeEnumeration"/>
      <xsd:element name="ABBIOType"
        type="ioTypeEnumeration"/>
      <xsd:element
        ref="tns:ABBAnnotationSchemaURL"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>

```

As shown, the WS-Resource properties specification document is defined using XML Schema [19]. The state of an *ABBResource* contains nine elements: (1) *ABBIId*, (2) *ABBName*, (3) *ABBLayer*, (4) *ABBState*, (5) *ABBPProtocol*, (6) *ABBInputDataType*, (7) *ABBOutputDataType*, (8) *ABBIOType*, and (9) *ABBAnnotationSchemaURL*. each being of an XML Schema Definition (XSD) type, either a simple XSD data type or a user-defined data type.

```

<wsdl:definitions
  ...
  <wsdl:portType name="ABBPType"
    wsrp:resourceProperties="tns:ABBResource">
    <operation name="getABBIId"/>
    <operation name="getABBName"/>

```

```

<operation name="getABBLayer"/>
<operation name="getABBState"/>
<operation name="getABBProtocol"/>
<operation name="getABBInputDataType"/>
<operation name="getABBOutputDataType"/>
<operation name="getABBIOType"/>
<operation name="getABBAnnotationURL"/>
</wsdl:portType>
</wsdl:definitions>

```

In order for an ABB user to know that the “ABBResource” defines the WS-Resource properties document associated with the Web service, the WS-Resource properties document declaration is associated with a Web Services Description Language (WSDL) *portType* definition in the WSDL definition of the Web service interface, through the use of a standard attribute *resourceProperties*. As shown below, the *portType*, with the associated resource properties document, defines the type of the WS-Resource. The *portType* “ABBPType” defines a set of nine operations to allow users to access the state information of the defined *ABBResource* (e.g., *getABBIOType*).

After the resource properties of the generic ABB are defined, the resource properties of its extended sub-ABBs (i.e., module-ABBs) can be defined by leveraging those of ABB and adding specific resource properties. The following segment shows the resource properties of *PresentationLayerABB* in WSRF. Since *PresentationLayerABB* is a subclass of the generic ABB, it automatically inherits all resource properties defined in the generic ABB. The state of a *PresentationLayerABB* contains two elements: ABB and an added element *PresentationLayerABBType*, which is with defined type *presentationLayerABBEnumeration* containing predefined values. The WS-Resource properties document declaration for *PresentationLayerABB* is associated with WSDL *portType* definition for user to access the state information of the stateful resource.

```

<xsd:element name="PresentationLayerABB">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="tns:ABB"/>
      <xsd:element
        name="PresentationLayerABBType"
        type="consumerLayerABBEnumeration"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Similarly, the resource properties of a third-module ABB *ConsumerABB* can be defined by leveraging those of *PresentationLayerABB* and adding specific resource

properties.

In this section, we introduced how to formally model ABBs, using *ad hoc industry standard* WSRF, as universal resources. As a result, any ABBs can easily interact and communicate with each other. It should be noted that ABBs can be modeled using other technologies.

5. ABB configuration framework

Considering the adaptive feature of business scenarios, we establish a rule-based configuration framework for ABBs. The configuration is handled in two ways: static template-based configuration and dynamic template creation.

5.1 Static template-based configuration

Using static template-based configuration, a user selects a template, comprising pre-configured ABBs, from a set of predefined pool based on specific service request scenarios. The ABB interface and attributes help in selecting these templates. A template is also associated with a set of rules. When selected, the rules are applied to create a configured set of ABBs. This approach is analogous to creating a Microsoft Word file using a predefined template, e.g., a business letter [20]. These templates can be customized to support any variations.

Figure 3. Static template-based configuration.

Figure 3 illustrates an example, where three presentation module templates are available: one supports Interactive Voice Response (IVR) systems; one supports PDAs; one supports Portals. Each of these three templates possesses its own characteristics and rules. As shown in Figure 3, if a user intends to create a presentation module supporting PDA applications, the PDA template will be selected, carrying three preconfigured ABBs: cache, access control, and default PDA transformation.

5.2 Dynamic template creation

In a dynamic template creation approach, a user

specifies certain characteristics; appropriate rules will be determined and appropriate ABBs will be dynamically selected and configured by the system when they are actually created. This feature also supports ABB variations that can be configured and adapted during run time. Figure 4 shows an example, where a user specifies four requirements: (R1) supporting device type is PDA, (R2) data changes once a day, (R3) user-specific information has to be presented, and (R4) users only have to go through security checking once. These characteristics are used during the run time to configure appropriate ABBs, by applying configuration rules.

Figure 4. Dynamic template creation.

As shown in Figure 4, the user requirements are automatically parsed and analyzed (for the interest of space, the detailed analysis method with rules and algorithms will not be discussed in this paper) to determine appropriate templates with associated ABBs. For example, requirement (R1) implies that a PDA template will be adapted; (R2) implies that a caching ABB is needed; (R3) implies that a PDA transformation ABB is needed; (R4) implies that an access control ABB is needed with a proper access control mechanism, say a security token. As a result, a PDA template is adopted and configured with three ABBs: cache, access control, and PDA transformation.

5.3 Comparison of two approaches

The static template approach requires little user effort; however, it requires that scenario templates be created and stored in a repository ahead of time. Furthermore, user cannot change these pre-configured templates. If flexibility or customization is required, then the option of dynamic templates should be adopted. Compared with the static option, the dynamic template creation method allows users to precisely customize and configure fine-grained ABBs, while the template repository offers a quick starting point with adaptability. However, this method implies significant development

efforts. Furthermore, how to control the degree of configurability remains a challenge.

6. Lessons learned on ABB identification and management guidelines

To realize the dynamic template approach in an effective and efficient way, we have two options. The first is to predefine configurable attributes of a template as well as allowable options. For example, we can specify that the security algorithm of the session control ABB of a PDA template is configurable, and it has a default algorithm and a list of selectable algorithms. The second option is to allow every attribute of any template to be configurable. For each attribute, a default value is predefined; a set of predefined options are also provided. Meanwhile, each attribute may be associated with some constraint rules. Thus, a user may decide to define a customized value for the attribute conforming to the constraint rules.

Both options have advantages and disadvantages. The first strategy is easy to implement and indeed adds certain configurability to each template. However, its drawback is apparent: the configurability of each template is preset and remains immutable afterwards. The second strategy provides flexibility for users to configure and customize presentation templates for specific usages. However, its leaving all decisions to users adds extra burden and ambiguity to many inexperienced users.

Our strategy is to adopt the dynamic template creation approach, but equip it with guidelines and best practices that can help users to identify and manage ABBs specific to project scenarios and requirements. These guidelines are valuable since architectural design can be enabled through these decisions for the presentation module in an SOA solution. Our challenge thus turns into how to establish a knowledge repository comprising guidelines and best practices.

6.1 Experimental design in practices

To realize this goal, we apply the aforementioned ABB-based presentation pattern into many projects of SOA-based system design and development. In each project, we selected a presentation template to build the presentation module comprising ABBs based on specific project requirements.

We documented important architectural decisions about various aspects of ABB identification and configuration for the presentation module, including the structure of the system, the provision and allocation of function, the contextual fitness of the system and adherence to standards. We call these considerations

architectural decisions.

At the end of each project, we captured top architectural decisions by working with other project architects. Considering the fact that different projects possess different features, we did not capture further weighting among the architectural decisions. We accumulated all projects' top architectural decisions. Finally, we obtained a list of sixteen architectural decisions as our base line. For each architectural decision item in the list, nine categories of information are requested: issue or problem statement, motivation, assumptions, alternatives, decisions, justifications, implications, derived decisions, and related decisions. Each SOA solution project could exploit the above sixteen architectural decision items to customize and configure their presentation modules using the dynamic template creation approach.

6.2 Experiences and lessons learned

Table 1 summarizes the top 16 architectural decisions (numbered as AD-X) based on our practices in the field. It should be noted that a specific application may require further architectural decisions; however, the descriptions here can be used as templates and examples to extract further decision points.

According to the practices, the relationships between the architectural decisions are summarized in Figure 5. A solid line denotes that an architectural decision item at the tail relates to an item at the head; meaning that if an architectural decision item at the tail is considered, the corresponding item at the head has to be considered as well. A dotted line denotes that an architectural decision item at the tail derives from the item at the head (item AD-03 derives from item AD-01 and AD-02).

Table 1. Top 16 Architectural Decisions for Presentation Module.

ID	Architectural Decisions	Descriptions
AD-01	Identification of ABBs	For each type of ABB, decide on yes/no and how many questions.
AD-02	Connections among ABBs	Specify connections among ABBs? e.g., connectivity patterns & relationships.
AD-03	Configuration of the interfaces of ABBs	How to configure and decide internal structures and operations of each identified ABB?
AD-04	Interaction patterns for ABBs with other modules	Will the identified ABBs connect with other modules?
AD-05	Stateful vs. stateless ABBs	Whether state information needs to be captured and tracked in ABBs?
AD-06	Granularity enablement of state management	To which granularity to capture and track state information? e.g., log all interactions?
AD-07	Federated vs. individual state management	State information be associated with individual ABBs or will a federated state management mechanism be used? Who will have access to logged states?
AD-08	Enablement of access control on ABBs	Who, when, what about accessing ABB.
AD-09	Business performance management	To which level of management? What kinds of information need to be monitored?
AD-10	Exception handling management	Which ABB instance creates exceptions? What kind? When? How to address and handle exceptions? What are the implications and impacts of the exception?
AD-11	Consumer types	Types of consumers the presentation module deals with, an individual or a program?
AD-12	Presentation models	Presentation models and techniques, e.g., Model-View-Controller (MVC) model, portlet technique, and Ajax technique.
AD-13	Message exchange for ABB interactions	Different internal schemas for ABB data descriptions. message exchange mechanisms.
AD-14	Message presentation transformation strategies	Input/output aggregation, presentation translation strategies for pre- & post-processing.
AD-15	Connectivity to the Business Process module	How to enable connectivity between presentation module & business Process module?
AD-16	Lifecycle management of ABBs	How to manage the lifecycle of an ABB instance?

As shown in Figure 5, architectural decision items AD-04, AD-06, AD-09, AD-10, AD-11, AD-12, AD-16 relate to item AD-01. Pairs (item AD-01 and AD-02, item AD-01 and AD-03, item AD-13 and AD-15) relate to each other. Items AD-05, AD-06, and AD-07 relate to each other. Item AD-04 relates to item AD-01, AD-02, and AD-03. Item AD-14 relates to item AD-15.

Interdependencies among the items help us divide the sixteen items into four groups for eliciting design guidelines: group 1 (AD-01~AD-04), group 2 (AD-05~AD-07), group 3 (AD-13~AD-15), and group 4 (AD-08~AD-AD12, AD-16). Phase two results in a set of guidelines over each of the sixteen architectural decision points.

Take AD-01 as an example. Unless there are sound technical or business reasons, ABBs are recommended to be used to construct the presentation module for goals of higher flexibility, extensibility, scalability, variability, and reusability.

Figure 5. Architectural decision relationships.

When simplicity, budget, and time are major concerns for a project, however, an alternative integrated architecture option may be adopted by identifying more

(a)

heavy-weighted components combining multiple ABBs in one unit.

Take AD-02 as another example, after ABBs are identified, they need to be properly connected with each other to collaborate appropriately. The predefined interaction patterns are recommended to be used, unless sound technical or business reasons exist and require stochastic interaction patterns. Typically, all ABBs should connect with each other through a presentation controller ABB; a cache ABB is recommended to enhance presentation performance and it should interact with the corresponding consumer profile ABB; a consumer ABB and a presentation ABB need to interact with each other.

Our case study also results in suggested steps for designing and managing ABBs, as summarized in Figure 6 (a). In addition, our case study also results in suggested steps for configuring fine-grained ABBs, as shown in Figure 6 (b).

(b)

Figure 6. (a) design & manage ABBs, (b) configure fine-grained ABBs.

7. Conclusions

In this paper we presented a method of representing and configuring flexible and extensible presentation patterns based on fine-grained Architectural Building Blocks. The framework helps to develop unified interface descriptions for each ABB. This allows ABBs to be discovered and composed in a standardized manner because the interface descriptions are based on

industry standards. We provide a summary of the top architectural decision points and we detail the relationships between those architectural decisions.

Currently, our ABB-based method is manually applied to solution design. Guidelines are provided to solution architects. However, the effectiveness of specific ABB identification, configuration, and management depends on the skill sets and experiences of corresponding architects. A tailored design and development environment should be constructed, which

will be our further research topic.

8. References

- [1] Sun. "JavaScript," Available from: <http://java.sun.com/javascript/>.
- [2] W3C. "Ajax," Available from: <http://www.w3schools.com/ajax/default.asp>.
- [3] B.W. Perry, *Java Servlet & JSP Cookbook*. 2004: O'Reilly.
- [4] "Model-Viewer-Controller," Available from: <http://java.sun.com/blueprints/patterns/MVC.html>.
- [5] L.-J. Zhang, J. Zhang, and H. Cai, *Services Computing*. 2007: Springer & Tsinghua University Press.
- [6] D.M. Selfa, M. Carrillo, and M.D.R. Boone. "A Database and Web Application Based on MVC Architecture," in *Proceedings of 16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06)*, 2006, Feb., pp. 48-53.
- [7] Y. Ping, K. Kontogiannis, and T.C. Lau. "Transforming Legacy Web Applications to the MVC Architecture," in *Proceedings of 11th Annual International Workshop on Software Technology and Engineering Practice (STEP'03)*, 2003, Sep., pp. 133-142.
- [8] X. Qiu. "Building Desktop Applications with Web Services in a Message-based MVC Paradigm," in *Proceedings of IEEE International Conference on Web Services (ICWS'04)*, 2004, Jun., pp. 765-768.
- [9] K. Rehman, F. Stajano, and G. Coulouris, "An Architecture for Interactive Context-Aware Applications," *IEEE Pervasive Computing*, 2007: pp. 73-80.
- [10] D.C. Schmidt, "Model-Driven Engineering," *IEEE Computer*, Feb., 2006: pp. 25-31.
- [11] K. Belhajjame and S.M. Embury. "A Model-Driven Approach to Service Composition in Virtual Enterprises," in *Proceedings of IEEE International Conference on Services Computing (SCC'06)*, 2006, Sep., pp. 214-221.
- [12] B. Benatallah, F. Casati, F. Toumani, J. Ponge, and H.R.M. Nezhad, "Service Mosaic: A Model-Driven Framework for Web Services Life-Cycle Management," *IEEE Internet Computing*, Jul., 2006: pp. 55-63.
- [13] J. Gray, Y. Lin, and J. Zhang, "Automating Change Evolution in Model-Driven Engineering," *IEEE Computer*, Feb., 2006: pp. 51-58.
- [14] F. Garzotto, P. Paolini, and D. Schwabe, "HDM - A Model Based Approach to Hypermedia Application Design," *ACM Transactions on Information Systems*, Jan., 1993, 11(1): pp. 1-26.
- [15] P.P. Chen, "The Entity-Relationship Model: Toward a Unified View of Data," *ACM Transactions on Database Systems (TODS)*, 1976, 1(1): pp. 9-36.
- [16] P. Fraternali and P. Paolini, "Model-driven Development of Web Applications: the AutoWeb System," *ACM Transactions on Information Systems (TOIS)*, Oct., 2000, 18(4): pp. 323-382.
- [17] A. Arsanjani, L.-J. Zhang, M. Ellis, A. Allam, and K. Channabasavaiah, "S3: A Service-Oriented Reference Architecture," *IT Professional*, May, 2007: pp. 10-17.
- [18]; Available from: <http://www-128.ibm.com/developerworks/library/specification/ws-resource/>.
- [19] W3C. "XML Schema," Available from: <http://www.w3.org/XML/Schema>.
- [20] Microsoft. "Word Template Category," Available from: <http://office.microsoft.com/en-us/templates/CT101527321033.aspx?Origin=EC010227111033&CTT=6&av=ZWD>.