2008 IEEE International Conference on Web Services

A Model-Driven Framework for Enabling Self-Service Configuration of Business Services

Xin Zhang1, Pei Sun1, Ying Huang2, Wei Sun1
1IBM China Research Lab, Beijing 100094, P.R. China
{zxin, sunpei, weisun}@cn.ibm.com
IBM Watson Research Lab, NY 10598, USA
{yxh}@us.ibm.com

Abstract

In seeking more profits, many business service providers have turned their attention to the technologies that can enable the delivery and operations of the network-delivered business services more efficiently and highly automated. To that end, many service providers have adopted the shared service model where the service offerings are shared across multiple customers to reduce the costs associated with the services' delivery. One key challenge for service providers is how to configure various parameters and options of the services with highly automated and self-service modes to meet the customers' diverse requirements with minimized costs. Therefore, effective service configuration lifecycle management technologies and approaches have become the differentiations of successful service providers. In this paper, we propose a common service configuration framework as one of the infrastructure features of the service operation platform based on the shared service model to ease and unify the configuration lifecycle across service offerings. In this framework, we adopt the model-driven technology to automate the service configuration lifecycle. This novel approach combines the wizard technology and rule-based validation technology to enable the service customers to perform self-service configuration. Furthermore, the effectiveness of the framework is discussed at the end of this paper.

1. Introduction

With the advent and maturity of the service-oriented technology, more durable and profitable network-delivered business services have emerged and prospered on the Internet. Such services include the Business Process Services (BPS, e.g., the payroll services from the ADP.com as in [1]) and Software as a Services (SaaS, e.g., the Customer Relationship Management (CRM) services from salesforce.com as in [2]). More and more Information Technology based companies, large or small, are enthusiastically developing the network-delivered business services. One major trend is the shared service model that has been derived from the traditional Application Service Provider (ASP) model [3][4]. The traditional ASP providers used to develop or customize an application into a new service instance to meet individual customer's needs, this approach is more appropriate for large enterprise customers with major budget for such customizations. For those service providers who target Small Medium Business (SMB) market or service offerings with large volume customers, they have to transform themselves to a shared service model to seek service delivery efficiency and cost-saving of the resulting service production and maintenance. With the market reality and intensive competition, the service providers with highly standardized service offerings, have to shift their paradigm from "Those who want strong configurability are not our customers" to a more flexible shared service model with strong configurability. The shared service model is characterized by pre-built and configurable service offerings that can be shared among different subscribers. Subscribers' specific requirements are satisfied by specifying the offerings' configuration points.

There are several key enabling technologies for the shared service model of network-delivered services, such as the multi-tenancy technology [5][6] and service configurability. The latter capability reflects the competency and quality of the service providers' offering. As the service consumers are motivated to ensure the subscribed service meet their specific business requirements, the configurability of service is treated as one key evaluation criteria by customers [4]. A customer usually subscribe several business services, e.g., CRM services and Human Resource Management (HRM) services, so they have to define dozens or hundreds of configuration points and options in the services to support their individual requirements.

978-0-7695-3310-0/08 $25.00 © 2008 IEEE
DOI 10.1109/ICWS.2008.85

While some research work [7][8] investigated the service configuration in the context of service deployment, the service configuration is much more than that, which can actually be divided into two tiers: the deployment level configuration and the application level configuration. The deployment level configuration is usually a one-shot task normally conducted by service providers, e.g., configuring the infrastructure resource quota according to the service level agreement. The application level service configuration usually happens more than once either at the customer's on-boarding stage or at the service's regular delivery stage. Preferably by service provider, the application level configuration should be performed by customers themselves to reduce service provider's delivery cost on an individual customer. Examples of application level configuration could include the specification of the "Product Code" in an inventory management service or the "Fiscal Period" in a Finance and Administration (F&A) service. Existing research efforts for service configuration have explored service composition and semantic approaches [9][10][11][12]. Most of the research focuses on the extensibility and flexibility for service configuration. The overall lifecycle of service configuration has not been fully studied.

In this paper, we analyze the challenges of the application level service configuration in the context of shared service model. To achieve the goal to minimize the service configuration related cost to service providers through automation and self-service approaches, we separate the configuration concern from service offering designs and propose an enabling framework from the platform perspective. The framework covers the end-to-end lifecycle of configuration and has an emphasis on the quality improvement and guarantee. The service configuration lifecycle can be "engineered" from the configurability development, user interaction and configuration validation, to service instantiation and regular delivery. We further apply the model-driven methodology [13] to the framework design with the benefits of development process automation, maintenance effort saving, and quality improvement. Base on this, we develop and incorporate the wizard and validation technologies into the framework to facilitate customers to perform self-service configuration. With the self-service configuration capability, the customers themselves can follow a guided configuration process to perform the service configurations based on their specific needs.

In the following sections, we start by analyzing the service configuration in the shared service model in Section 2. In Section 3, we present the model-driven framework of the service configuration. We discuss the key technologies to enable the self-service configuration in Section 4. We further describe the implementation consideration and several real cases in Section 5. Finally, we provide a summary of our work in the last Section.

2. Shared Service Model and Service Configuration Challenges

2.1 Shared Service Model

As the competition among service providers becomes more intensified, many have turned to the shared service model. Since the shared service model enables one pre-built service offering to serve many customers while sharing the same infrastructure and resources, the service providers can share their investment on the infrastructure and cost on service development and maintenance among the customers. On the other hand, the service consumer can regard their subscription as personalized and dedicated service to satisfy their individual requirements. Actually all the customers' are served by one service application code base with isolated configurations for each customer at the service operation platform level. Therefore, the shared service model has clear advantages of the low cost, service agility, responsiveness, and reliability, and it is special appropriate for services targeting SMB market.

The Web service architecture [14] reveals the publish-find-bind mode and different roles in the Web service ecosystem. With a deeper study of the service provider side of the shared service model, we find a shared service has two modalities in its lifecycle: the pre-built service (or service offering), and the configured service (or service instance), as shown in figure 1. The pre-built service is the service implementation developed and packaged with configurability. It is deployed on the service operation platform which provides the infrastructure as well as management components. The service provider publishes its description to the service broker. The service requester can find the pre-built service by searching service broker's registry. By developing an understanding of the service and business negotiations, the service requesters subscribe the service with their specific requirements through configurations. The pre-built services are configured accordingly and the service instances are launched to serve the specific requesters while physically they share exactly the same code base of the service offering.

Though there have been point technologies to support service configuration, e.g., template technology [15], configuration schema specification [16], the overall service configuration lifecycle is handled in a service dependent approach with little or none platform support. Service developers define the configuration schema, interface and configuration in the same manner as what they do in the traditional software development. This model works well for the traditional software development in which a relatively long development and delivery cycle time is allowed. But the network-delivered services with shared delivery model usually expects lower provisioning cost, automated delivery cycle, minimized maintenance effort, and responsive to dynamic requirements from different customers, it requires a more effective way to systematically handle the service configuration lifecycle.

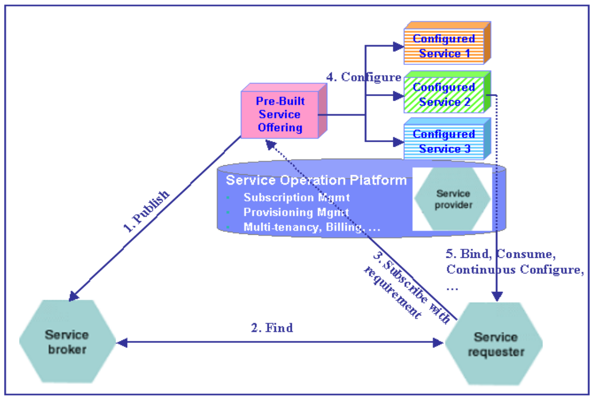

Fig.1. Shared Service Model

In the shared service model, we can identify the following service configuration related efforts:

- The configurability needs to be implemented in service offering.
- The configurability needs to be identified and modeled.
- The service provider needs to develop a configuration portal as the User Interface for service requesters to configure the service.
- The configuration portal will be launched and used to interact with the requester to collect the configuration data.
- The configuration data needs to be stored and maintained in a repository.
- The configured service instance will be launched to serve the corresponding service requester.

2.2 Self-service Configuration and its Challenges

While software configuration can be performed by an individual person that serves the customer to implement or deliver the software, the network-delivered service requires a more agile and cost-effective way to do service configuration as the customers have higher expectations of fast on-boarding and the service providers want to minimize the delivery cost as much as possible for each customer. By introducing self-service into the service configuration lifecycle, we can provide customers with the flexibility to configure the subscribed services by themselves. Through this approach, not only the delivery cost can be lowered for service provider, but also a more flexible and agile user-experience can be provided to the customers. Moreover, the customers tend to prefer the services that they can control more. After all, the customers know their own business requirements better than anyone else.

In the context of service configuration in the shared service model, there are two critical issues to be considered: the first one is, how to enable the customer easily understand the configuration points and their relationships (e.g., correlation, constraints, etc.). That knowledge is usually accumulated by service specialists through many customer engagements. Even with rich documentation describing the meaning of these configuration points and their relationships, it is still hard for the customer to map such configurations to their daily business needs. Moreover, it always costs a lot of effort to develop an easy-to-use user interface for the customers to configure the services. The second issue is how to assure the quality of the configuration inputs made by the customers. As the customers specify their configurations in a self-service mode, a built-in mechanism is highly demanded to ensure that the configurations are valid in logic and consistency. In other words, certain technology should be devised to ensure the quality of customers' configuration inputs.

3. The Self-Service Configuration Enabling Framework

The self-service configuration enabling framework tries to separate configuration design from the service application itself, which enables the service development focus on its own business function implementation and leaves the configuration related development as the platform common feature. With this design, the service configuration lifecycle can be "engineered" from the aspects including configuration development, user interaction, configuration validation, and service instantiation and execution. Therefore, different services share the same platform methodology and supporting framework to enable the service configuration in cost-effective, agile, and stable manners. The model-driven methodology is applied in the framework to accelerate the development process and ensure the development quality. The wizard and validation technologies are also adopted in the framework to support self-service configuration. We will discuss the technologies in more details in the next section.

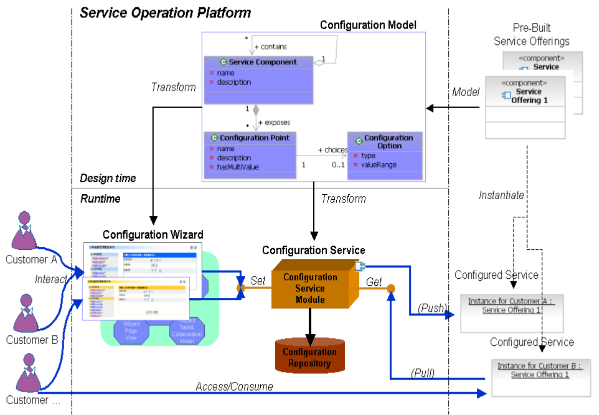

Fig.2. Self-Service Configuration Enabling Framework

The designed framework is shown in figure 2. There are three key components: the Configuration Model, the Configuration Service, and the Configuration Wizard (Portal). The Configuration Model is a design time schema to model the service configurability. Three key artifacts are defined in a basic Configuration Model: Service Component (the architectural entity in a service offering), Configuration Point (the elementary unit that a service offering exposes and can be set to customer's specified value), and Option (the definition of the type or range of a candidate value for a Configuration Point). Configuration Point is attached to Service Component where the Configuration Point is implemented and interpreted at runtime. The Configuration Points are organized in a hierarchical structure of the Service Components in a service offering. A Configuration Point can either be as simple as an atomic data type (e.g., a string) or as complex as a composed data structure (e.g., a table or a hierarchical tree). Configuration Service is a runtime repository for managing configuration data. It provides a standard set of APIs (e.g., Web services interface) for storing or retrieving configuration data from the repository. The Configuration Service is designed with extensibility that is able to store the set of configuration data for each of the configured services with compliance of the Configuration Model schema. Some design in software configuration management, e.g., versioning, can also be applied in the Configuration Service. Besides the "pull" mode of the configuration service interface, the "push" mode can also be implemented through a plug-in mechanism that enables the configuration data to propagate into the service. Configuration Wizard is the runtime engine which functions as a portal to interact with user to collect input of configurations.

With this framework, the service configuration lifecycle can be accelerated on the service operation platform. When a configurable service has been developed, tested and deployed onto the platform as a new platform service, then the service provider defines the service's configurability according to the Configuration Model. Once the model is established, two threads of automatic transformations will be triggered. One is to transform from the configuration model into the repository of the Configuration Service. Another is to transform from the configuration model to the configuration UI codes, which will be interpreted by the Configuration Wizard to interact with the user to collect configuration data. In this way, the configuration development can be done through modeling tool without even writing a single line of code. During the time of service regular delivery, the Configuration Wizard launches a set of user interfaces automatically to interact with the customer to collect configuration data when the customer gets on-boarding to the service. After that, the collected configuration data is uploaded and managed in the configuration service repository. A service instance can be refreshed by either pull or push mode which attaches the configured data with the service offering for the specific customer.

To leverage the configuration capability of the platform, the service need to be able to retrieve its configuration data managed in the Configuration Service repository, or, an adaptor need to be developed and plugged into the Configuration Service to propagate the configuration data to the service in the service specific manner. Though that may requires some extra effort at service development phase, we think it's worthy considering the benefits it brings.

4. Key Technologies to Enable the Self-Service Configuration

4.1 From Configuration Model to Configuration Wizard Model

One important aspect to enable the self-service configuration is to pre-codify the configuration knowledge as much as possible into the configuration portal so that customer can be guided to complete the configuration process easily. The Configuration Model, which has been briefly depicted in the framework description, mainly focuses on the structural part of configuration, with little considerations of the behavioral aspect of the configuration. However the customer usually takes a task-oriented approach to conduct configurations through a certain sequence of configuration steps. In order to facilitate the configuration process, the following behavioral aspects of configuration need to be considered in the configuration wizard: the constraints among the configuration points, the correlations among different configuration steps, the flow logic to meet a specific configuration objective, the collaboration relationship among roles to fulfill the configuration task. All these capabilities are the important factors for the customers to complete the self-service configuration with assured correctness.

Therefore we extend the Configuration Model into the Configuration Wizard Model shown in figure 3. Beyond the existing Configuration Model that defines the data schema of service configurability, which can be classified as a domain-oriented model, a set of sub-models is extended on top of it. We named it as the task-oriented wizard model. In the task-oriented wizard model, the Constraint Rule Model describes the correlation and constrains among the configuration models. For example, if the 'PaymentType' configuration point is set as 'CreditCard' option then the 'PaymentId' configuration point is required to have a number with the length between 13 and 20. The Presentation Model aggregates configuration points that are closely related into the same activity. Each activity can be presented and interact with the user in one configuration step. The Navigation Model defines the conditions to perform a specific activity and the dependence among the activities. The Collaboration Model defines the role responsibility for activities. It can be directly mapped to some human task mechanism in the runtime. The Configuration Flow model is an aggregation of the above Models into a task flow for one configuration. The rule patterns defined in the Task-Oriented Wizard Model can be categorized in three tiers. The first tier rules define the constraint among different configuration points within a configuration activity, denoting the constraint among UI elements in a single configuration step. The second tier rules define the correlation among different configuration activities, which will be referred by the Navigation Model in its pre-condition or post-condition. The third tier rule defines the consistency policy among the configuration data that will be validated after all the configurations are completed to ensure the quality of the configuration inputs.

With the enhancement of the task-oriented wizard model, more semantic and behavioral information can be fabricated into the configuration wizard by means of model transformation. Then the guided configuration process built-in with such information can be automatically enabled, which helps the customer complete the configuration process in self-service mode with certain quality.

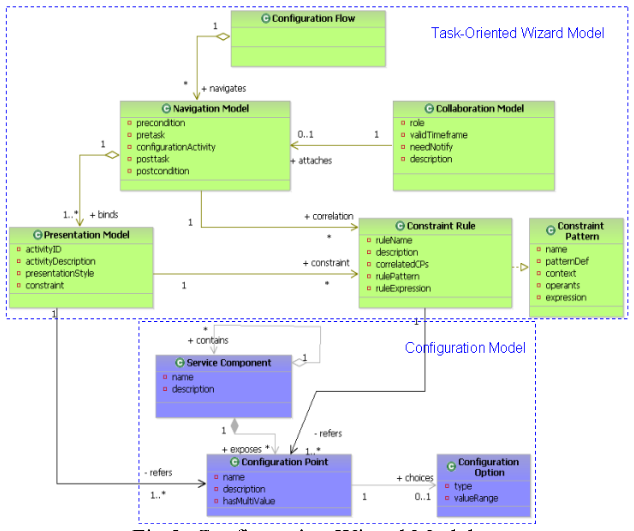

Fig.3. Configuration Wizard Model

4.2 Transformation Design

Model transformation is an important design in the framework to accelerate the configuration development. There has been significant progress in the model-driven development methodology and corresponding tools and practice in the recent years [13] [17] [18]. However there are also some challenges and issues regarding the model-driven development that has been depicted in [19].

In this framework, the Configuration Wizard Model described in the earlier section is the PIM (Platform Independent Model) [13]. The PSM (Platform Specific Model) [13] could be implemented with different executable languages, e.g., jsp, servlet, xforms, portlet, etc., depending on the programming environment of the service operation platform. The transformation can be implemented by using different technologies, e.g., jet [20], xslt [21], or even java code.

One particular challenge we encounter, which hasn't been fully addressed by others, is the readability of generated PSM code. As in the framework, the custom development will most likely happen if some features or information are not well captured in the PIM model. Although some extension points can be predefined so that custom development can be done to a certain scope in a non-intrusive way, the source-code level custom development cannot be fully avoided. Therefore in the transformation design, the readability of generated code has to be seriously considered. From the design perspective, there are three key factors impacting the readability of the transformed code:

- The PSM programming model. A relatively low-level programming language, e.g., a script language like javascript, is definitely less readable comparing to a programming language in higher abstract level, e.g., declarative programming language, such as xforms [22].
- The code structure design in PSM. A layered and componentized code structure adopting architectural patterns, e.g., Model View Controller(MVC) pattern, can help the generated code easy to be parsed and understood.
- The naming convention. The naming convention is an important yet easy-to-neglect design artifact in the transformation design. A consistent naming convention can associate the generated code elements and capture the relationships among these elements. It is also helpful to trace back to the PIM model for better understand the PSM code.

4.3 Validation for High-quality Configuration

Validation is essential to ensure the quality of configuration, especially in the customer self-service mode. Beyond the syntax-level validation, there are more requirements for semantic validation, e.g., the 'Level2Approval' shall have different value than the 'Level1Approval' to avoid duplication. XML schema supports syntactic validation very well. However it has only limited support for the semantic validation. In order to support semantic validation, the validation framework is designed as shown in figure 4. Some XML constraint specifications can be leveraged as executable language, e.g., schematron [23], XincaML [24].

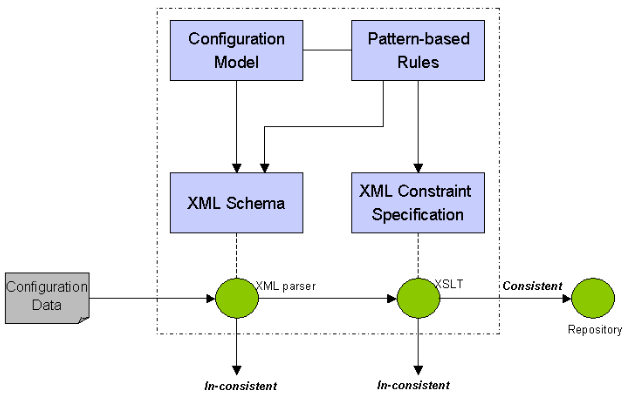

Fig.4. Validation Framework

By defining model-level rule patterns on top of the Configuration Model, we can be assured the correct transformation into the execution level schema and constraint expressions. The model-level rule patterns have various types. It can be simple as the Occurrence pattern that defines the occurrence of a configuration point, or it can be complex as the Repulsive pattern, which defines when one configuration point is set to a value another configuration item should comply to a specific value restriction to avoid conflicts. The calculation can be the other pattern that defines one configuration item as a calculation by values of multiple other patterns.

5. Implementation and Case Study

The framework has been applied to two service operation platforms to support the configuration lifecycle for multiple services. The implementations are slightly different depending on the different PSM models selected by the service operation platform. Based on our practice, the result was evaluated to be beneficial by accelerating the configuration development cycle time with a 60% decrease. The customer feeds back with remarkable improvement of user experience.

In one implementation, we take a lightweight approach to model the configuration wizard model using Eclipse Modeling Framework(EMF) model. We develop the toolkit in the Eclipse platform. The transformation is done in XSLT. The target PSM model is xforms at browser-side and servlet at server-side adopting the MVC architectural pattern. The configuration service uses the DB2 UDB, the IBM database server, as the repository and is planned the further shifting to XML repository by upgrading to the DB2 XML supported version. The configuration service exposes the Web services interface for the configuration data storage and retrieval. In the design of the configuration service, the customer id, service name plus the configuration point id are combined into a unique id for the identification of the specific data in the repository. With this design, the configuration data of multiple services and different instance of one service can be easily managed in the repository. The validation is implemented by transforming the model level constraint rule into XSLT code. The runtime wizard engine is devised to support wizard access management and maintenance.

The platform has been in production for a network-delivered business service platform serving SMB customers. The service deployment lifecycle has been accelerated and the self-service is enabled with the adoption of the model-driven self-service configuration framework. The service deployment time is significantly reduced to two person weeks in average comparing with six to eight person weeks previously. The customers are getting used to conduct self-service configuration to their services, either in the on-boarding phase or in the steady state. The service maintenance has also been considered easier as all the customers' configuration data can be viewed, managed and backup in a consistent manner through a simple SOAP based utility.

6. Summary

In this paper, we explore service configuration in the context of network-delivered service platform with a shared model. An enabling framework is developed to "engineer" the service configuration lifecycle with highly automated and self-service approaches across service offerings on a delivery platform. The framework adopts a model driven approach to accelerate the configuration development and management process. In order to meet the self-service requirements, we develop relevant technologies to extend the configuration from a domain-oriented configuration model into a task-oriented configuration wizard model, a guided configuration process with pre-built knowledge into the configuration portal is provided. Furthermore, with the validation rule patterns introduced, customer can easily define the semantic level constraints among configuration points. The configuration quality can be better assured accordingly.

The framework has been deployed to support the configuration lifecycle for several network-delivered business services. It significantly streamlined the service configuration's development, definition, provision and management. Currently, this framework can support customers' configurations within the scope defined through parameterized configuration points by service providers when they build and deploy the services. Our next step will be exploring how to further extend the flexibility for configurations to support less constrained configurations at user interface, process and data layers.

References

[1]Gartner Report, "Product Review: ADP Brokerage Processing Services", 2006
[2]Gartner Report, "Salesforce.com Helps SMBs Accept Hosted CRM", 2005
[3]Summit Strategies Report, "From Myth to Reality: Traditional ISVs' Evolving Software-as-Services Strategies", 2004
[4]Summit Strategies Report, "Software-Powered Services: Net-Native Software-as-Services Transforms The Enterprise Application", 2005
[5]Greg Gianforte, "Multiple-Tenancy Hosted Applications: The Death and Rebirth of the Software Industry", RightNow Technologies, Inc., 2005
[6]"Multi-Tenant Architecture", Cincom's Synchrony Whitepaper, 2005
[7]Eelco Dolstra, Martin Bravenboer, and Eelco Visser, "Service Configuration Management", Proceedings of the 12th international workshop on Software configuration management
[8]Rainer Anzböck Schahram Dustdar Harald Gall, "Web-based tools, systems and environments: Software configuration, distribution, and deployment of web-services", Proceedings of the 14th international conference on Software engineering and knowledge engineering
[9]B. Srivasta, and J. Koehler, "Web service composition – current solutions and open problems". In ICAPS 2003 Workshop on Planning for Web Services, 2003, pp. 28-35.
[10]D. Karastoyanova and A. Buchmann, "Automating the development of web service compositions using templates" In Proceedings of the Workshop "Gesch¨aftsprozessorientierte Architekturen" at Informatik 2004.

[11]K. Sivashanmugam, J. Miller, A. Sheth, and K. Verma, "Framework for semantic web process composition", Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, Special Issue of the International Journal of Electronic Commerce (IJEC), C.Bussler, D. Fensel, N.Sadeh (Eds.), Vol. 9, No. 2, 2004.
[12]A. ten Teije, F. van Harmelen, B.J. Wielinga, "Configuration of web services as parametric design". E.Motta, N. Shadbolt, A. Stutt, N. Gibbins (Eds.): Engineering Knowledge in the Age of the Semantic Web, 14th International Conference, EKAW 2004, Proceedings. LNCS 3257, ISBN 3-540-23340-7, Springer, UK, 2004, pp. 321-336.
[13]OMG Model Driven Architecture, http://www.omg.org/mda/
[14]Karl Gottschalk. Web Services architecture overview. IBM whitepaper, IBM developerWorks, September 2000. http://ibm.com/developerWorks/web/library/w-ovr/
[15]Andy Clarke, Christopher Gibson, ed., Component Templates for Assets and Artifacts, http://www.alphaworks.ibm.com/tech/ct4aa
[16]Service Component Architecture Assembly Model Specification, http://www.osoa.org/download/abttachments/35/SCA_AssemblyModel_V096.pdf
[17]L.Balmelli, D.Brown, M.Cantor, M.Mott, Model-driven Systems Development, IBM System Journal, Vol 45, No 3, 2006
[18]A. W. Brown, S. Iyengar, S.Johnston, A rational Approach to Model-Driven Development, IBM System Journal, Vol 45, No 3, 2006
[19]B.Haipern, P.Tarr, Model-driven development: The good, the bad, and the ugly, IBM System Journal, Vol 45, No 3, 2006
[20]Java Emitter Templates(JET), http://www.eclipse.org/modeling/m2t/?project=jet#jet
[21]XSL Transformations, http://www.w3.org/TR/xslt
[22]XForms 1.0, http://www.w3.org/TR/2006/REC-xforms-20060314/
[23]L. Doodds, "Schematron: Validating XML Using XSLT", Proceedings of XSLT UK Conference, 2001, Keble College, Oxford, England
[24]IBM alphaWorks, eXtensible Inter-Nodes Constraint Mark-up Language (XincaML), December 2002. http://www.alphaworks.ibm.com/tech/xincaml