MSI Services Community: A Synthesized Platform for Services Lifecycle Management

Li Kuang ¹*, Jian Wu² and Shuiguang Deng ²

1 Hangzhou Institute of Services Engineering, Hangzhou Normal University, Hangzhou 310012
2 College of Computer Science and Technology, Zhejiang University, Hangzhou 310027

E-Mails: corrine.li2009@gmail.com; wujian2000@zju.edu.cn; dengsg@zju.edu.cn

* Hangzhou Institute of Services Engineering, Hangzhou Normal University, Hangzhou 310012

Received: Aug. 2010 / Accepted: Nov. 2010 / In Press: Dec. 2010 / Published: Feb. 2011

Abstract: Since the 21st century, modern service industry has mushroomed around the world, and the application systems in modern service industry are usually built based on the reuse of other third party services in a distributed and loose-coupled way. Consequently, there will be a greater need for the provisioning and lifecycle management of services. In this paper, we analyze the drawbacks of current service repository, propose a framework for services lifecycle management with many capabilities covered, including services registration and annotation with semantic information, flexible categorization and clustering of services, accurate and efficient services query and matchmaking based on interface and behavioral semantics, service orchestration and verification, and service execution and monitoring. The framework is implemented as a platform called MSI services community, and applied and verified in a test bed of AliSoft service integration platform.

Keywords: Service; Lifecycle management; Synthesized platform

1. Introduction

Modern service industry is a kind of knowledge and information intensive industry, which is developed with the aid of modern information technologies and management concepts at an advanced industrialized stage. It includes the ones upgraded from traditional services industry through the renovation of production and business mode, as well as the brand new ones generated with the rapid
development of information network. Since the 21st century, modern service industry has mushroomed around the world, and major developed countries have shifted from “industry-oriented economy” to “service-oriented economy”. The main characteristic of application systems and software platforms in modern service industry is that a large amount of third party services are involved, and they collaborate closely with frequent interactions. As the demand for higher value services increases, there will be a greater need for the provisioning and lifecycle management of services.

A common term for such products is “SOA repositories”, and one of the most representative products is UDDI (Universal Description Discovery and Integration) registry [1]. However, it just provides storage and cataloging of information about individual services, as well as a simple keyword-based mechanism for clients to find Web services. Such SOA repositories are far from meeting the demands for service lifecycle management in modern service industry.

First, there will be a great deal of interactions and collaborations among services. However, traditional service specifications, such as WSDL [2], just provide information about interfaces and invocation bindings, and they may describe the same thing in different ways due to the lack of semantics. In order to make sure that services interoperate successfully and correctly at the behavior semantic level, it requires service providers providing additional semantic information and internal working process of a service when registering it.

Second, there will be a large amount of third party services gathered, and it will be a pressing problem to be addressed that how to manage so many services, lest service consumers get lost in a sea of services. This requires service community can provide an effective and precise services discovery mechanism, and consumers can get through the overview of the results to pick up the most appropriate service in an intuitive and prompt way. Moreover, the quality of services will be a key indicator when making choices among multiple services with similar functionality; hence service community should be able to collect updating data of quality of services in real-time and rank services according to the evaluation of their performances.

Third, composite services built from atomic ones will be more and more popular as the requirements become more and more complicated. This requires service community can provide a convenient tool for service orchestration and composition. And meanwhile, verification mechanisms should be provided to ensure the correctness of internal logic of service processes as well as the compatibility between involved atomic services.

In a word, we need a more sophisticated solution to provide capabilities such as services registration and annotation with semantic information, flexible categorization and clustering of services, accurate and efficient services query and matchmaking based on interface and behavioral semantics, service orchestration and verification, and service execution and monitoring. We have developed an MSI service community with these capabilities covered. Moreover, it provides a visualized way to facilitate users with the overviews and operations on services through services lifecycle. We will introduce the architecture of MSI services community in the following section. We implement the proposed framework and introduce each component of MSI services community in Section 3. And then a case study for the application of MSI services community in a test bed of AliSoft service integration platform is presented in Section 4. Finally, conclusions and future work are given in Section 5.
2. Architecture of MSI Services Community

Figure 1 shows the architecture of MSI services community. There are five components in the framework, in correspondence with five phases of service lifecycle management. At the 1st stage, service providers register services and annotate them with rich semantic descriptions. Then MSI service community processes the metadata of services by creating/updating inverted indexing for services and tagging services with proper categories information based on an automatic clustering algorithm. At the 3rd stage, service consumers can locate candidate services according to keywords, interface semantics and behavior semantics, with the support of 3 layers of service discovery mechanism, utilizing the established indexing and visualization technique of categorical distribution. At the 4th stage, users can design composite service processes by themselves for more complicated requirements, sketching the skeleton of processes in a drag-and-drop way, and binding atomic services for each node using the results of service discovery. The customized service composition needs to be verified before execution. At the 5th stage, either the registered atomic services or the orchestrated services are executed through service execution engine, and users can monitor and control the execution of services through the monitored control tool. And finally, the performance data of service execution is used to update the quality of relevant services. So far, a general scenario of a whole service lifecycle has completed.

In the following section, we will introduce each component/phase of service lifecycle in details.

3. Components of MSI Services Community

![Figure 1. The architecture of MSI services community](image-url)
3.1. Services Registration and Annotation

Services registration and annotation module helps service providers register a service to service community through a registration guide. In general, 3 parts of information need to be provided: first, basic properties such as service name, functional description in natural language, version statement and keywords; second, service description files according to a certain specification; and third, the promised quality of service, such as its availability, reliability, security, pricing, resource management, and performance guarantees. And for the second part, we adopt OWL-S [3] as the core model for describing a service. It further consists of 3 main parts: service profile demonstrating the functionality of a service by describing its interfaces and input/output messages; process model showing the internal working process of a service by describing its primitive component operations and the routing relations among them; and service grounding providing details on how to access the service by specifying its protocols, message formats, port and so on. There are two kinds of services differing in process model, which are atomic and composite services. Atomic services correspond to the actions a service can perform by engaging it in a single interaction; while composite services correspond to actions that require multi-step protocols and/or multiple server actions [3].

Besides the support to OWL-S specification, MSI service community also integrates the support to WSDL specification, which is once a widely-adopted standard service description language. Being different from that in OWL-S, services in WSDL are all regarded as black boxes and they correspond to atomic services in OWL-S. On the whole, the content of a service in WSDL can be mapped to service profile and service grounding of OWL-S. However, WSDL files lack semantic information, which may lead to misunderstanding confusion between coalition partners. Therefore, an extra step needs to be taken when a service provider uploads service description files in WSDL specification, that is, they need to annotate all the port types, operations and messages of a service with ontology concepts. We integrate the concept database of HowNet [4], a large knowledge database of common-sense concepts with inter-conceptual relations and inter-attribute relations included, and we utilize a part of its operations, such as get the hyponym, synonym, antonym and attribute of a concept, to facilitate annotators finding proper ontology concepts for semantic annotation. Fig.2 shows the annotation tab for WSDL files, where the left side is for the parsing result of a WSDL file, the right side is for searching ontology concepts in HowNet, and the selected concept can be dragged onto the porttype/operation/message on the left side for annotation.

Moreover, we then formalize service behaviors with $\pi$-calculus [5], which is process algebra where processes interact by sending communication links to each other proposed by Robin Milner. $\pi$-calculus is designed to describe and analyze concurrent systems, and a service with complex behavior can actually be deemed as a concurrent process which is composed of a set of actions receiving and sending messages, thus it is appropriate to model services with $\pi$-calculus. Also, there are a serial of algebraic theories in $\pi$-calculus, such as bi-similarity and congruence, which can help us analyze services behavior. Service behavior refers to the dynamic properties of a service, which includes the operations involving message exchanges that can be performed by the service, and the constraints between operations of a service that define the allowed order of execution. Accordingly, the
formalization of service behavior with π-calculus can be divided into two parts: the formalization of all involved atomic services/operations and the representation of the structure of these atomic services/operations. For further information, please refer to [6].

3.2. Services Indexing and Clustering

The number of services in MSI service community is expected to increase dramatically. In order to facilitate service consumers to rapidly obtain a sound overview of services that they may be interested in and locate the candidate services precisely, MSI service community processes the metadata of services periodically when the increased services reach to a certain amount. Two dimensions of such processing are included, which are creating inverted indexing on services and clustering similar services for automate categorization.

3.2.1 Creating Inverted Indexing on Services

We use inverted indexing, a word-oriented indexing mechanism, to index all the ontology-annotated outputs in registered services. For each ontology-annotated output, there is a service list which records all the services that deliver the output. The indexing is created or updated for all the outputs of the service when it is registered in MSI service community. In addition, it has been well-recognized that we can expect that a service that advertises an output of cars provides some type of vehicles [7], hence in other words, a service can deliver not only the output it declares, but also all the ancestors of the output defined by the ontology. Therefore, index records from all the ancestors of the output to the service will also be generated and kept at the meantime. With the established inverted indexing, we do not need to match a registered service against the request one by one anymore, but just to check which services can provide all the desired outputs first to accelerate the filtering of irrelevant services.
3.2.2 Automatic Clustering of Services

Another part of the management on services metadata is to cluster services with similar functionality or similar category automatically. Such clustering and categorization of services is not only a meaningful approach to accelerating service discovery, but also an effective way for clients to get through the distribution of the query results, just like that we classify documents in different directories for easy index and reference in our daily life.

At the business layer, we use adaptive Back-Propagation neural network Model (BPM) [8] to perform the recognition of the category that a service belongs to. During the training process, the feature vectors of training services and their categories are learned by BPM. The element in the feature vector is the semantic similarity between the feature word in the ontology database and the occurrence in the feature set for a service. During the recognition process, the characteristics of the test service are analyzed by BPM and the output shows the category of the test service. Furthermore, the BPM is adapted with correct recognized test services, which results in better modeling over time.

![Figure 3. (a) services shown in Cluster Map; (b) services shown in TreeView.](image)

As a result, all registered services are tagged with the categories they belong to automatically. At the presentation layer, we adopt several interactive data visualization toolkits to present the overview of multiple services, such as Aduna ClusterMap [9] and Prefuse [10]. ClusterMap visualizes the instances of a number of selected services organized by their classification. The Cluster map in Fig.3(a) shows a collection of services according to the automatic categorization by BPM. Services with the same category and similar functionality are grouped in clusters. Distinct clusters are represented in different colors and tagged with rounded rectangles stating their names and cardinalities. The relationships among distinct clusters are captured by directed edges pointing from a specific class to a more generic one. Some service instances may be subordinate to several categories at the same time, so there may be intersections between distinct clusters connected by balloon-shaped edges. Within a cluster, each service instance is represented by a small yellow sphere, and the quality of a service will be reflected to the shape of the sphere. The higher quality a service can provide, the bigger its indicated sphere will be. TreeView of Prefuse is another visualization perspective provided by MSI service community. As shown in Fig.3(b), services are layout as a tree, where each parent node in shadow represents a category and each leaf node tagged with a name represents a service. Similarly,
the higher quality a service can provide, the larger its indicated tag will be. There are still some other visualization perspectives supported, such as Treemap, GraphView and ZoneManager, for better comprehension with the overview of a collection of services.

3.3. Services Query and Matchmaking

MSI service community provides a platform where a most comprehensive collection of services is covered. However, the same comprehensive collection of services makes service consumers spend more time on locating the most appropriate services they are interested in. This relies on an expressive representation of both service descriptions and searching requirements, and correspondingly an effective and precise service discovery mechanism. There are three layers of service discovery mechanisms in MSI service community, which are keyword-based service discovery, interface-based semantic service discovery, and behavior-based semantic service discovery.

Keyword-based service discovery allows service consumers to delimit a general scope of the desired services by matching the keywords against the basic properties such as service name and functional description.

Interface-based semantic service discovery allows service consumers to further specify the requirements on interface semantics by checking whether a service can produce all the desired outputs of a request and meanwhile the request can provide all the inputs to invoke the service. A two-phased matchmaking algorithm is introduced, which are atomic-level and composition-level service discovery. The former one checks whether there is a single service that can match the request, and if not, the latter one will then check whether there are instant compositions of services able to fulfill the request. The generated service compositions will then be registered as a new service to MSI service community automatically for reuse. The algorithm greatly accelerates the filtering of irrelevant atomic services by making use of the established inverted indexing, and increases the likelihood of finding a possible candidate by exploring service composition. A detailed introduction to the two-phased interface-based semantic service discovery can refer to [11].

However, that a service is syntactically and semantically compatible with the request in interfaces does not mean that the service can be compatible with the request in interacting behaviors such as the ordering of message exchanges and blocking conditions. For example, an online shopping service waits for payment before sending the product while the customer sticks to Cash-on-Delivery. It is obvious that the collaboration between them leads to a deadlock even they are syntactically and semantically compatible in interfaces.

Therefore MSI service community provides behavior-based semantic service discovery, which allows service consumers to lay restrictions on the interaction behaviors between the service and its potential client services in a target context and/or that between the service and the ones it invokes when realizing its functionality, so that the returned services can be behaviorally compatible with its potentially interacting services. When a service is registered to MSI service community, we have formally transformed its process model into a process expression in \( \pi \)-calculus. Similarly, we can model the requirements on behavioral semantics posed by service consumers into a virtual service in \( \pi \)-calculus expression. Of course, consumers just need to specify part of the behavioral constraints that
they really care about. The constraints are composed of several behavioral segments, which are segments of an entire execution flow and imply what the specific operations are and when the operations perform tasks in an execution flow. The unrestricted parts in an execution flow can be regarded as a $\tau$ process that can be matched to any actions. For instance, a service request that lays emphasis on payment before delivery while ignoring other parts like login, can be modeled as $\tau P_{\text{pay}} \tau P_{\text{del}} \tau$, where $P_{\text{pay}}$ and $P_{\text{del}}$ represent the operations of payment and delivery respectively. There are 6 kinds of behavioral patterns defined in [6], such as a service flow starts or ends with a specific operation, contains an operation, and contains two operations with a specific order, and the detailed explanations can be found in [6].

As a result, behavioral-based semantic service discovery turns out to be the problem to find a service which is behaviorally equivalent to the virtual service modeled by the service request in $\pi$-calculus. We then can address the problem by applying the bisimulation/ equivalence theory of $\pi$-calculus. There are mainly two kinds of equivalence, which are strong equivalence (strong bisimulation) and observational equivalence (weak bisimulation) [5]. This kind of equivalence requires two service processes can mutually simulate each other’s actions from the initial state, so as to evolve to a next simulating state until both processes terminate. The difference between strong and weak bisimulation lies in: the former requires the simulating actions should be exactly the same, including internal actions; while the latter allows leaving out some internal actions and the simulating actions just need to be the same from the outside. Meanwhile, a lot of related toolkits have been provided to help analyzing service behavior and interaction, and we can easily utilize them for the behavioral matchmaking of services.

Finally, we can rank the result set of a query according to quality of services. The ranking indicator can either be an individual one or a combination of some ones. For example, some consumes wish to use the cheapest services while some wish to use the most stable services.

3.4. Services Orchestration and Verification

At present, most services published on the internet are with simple structure and single functionality, and they cannot satisfy the requirements of complex business applications. It is a key step in the evolution of web services to compose and integrate various services distributed over the internet to achieve a powerful business-level service process for some business objective, and this is the key to the success of the application and implementation of SOC and SOA.

MSI service community provides a service-oriented composition platform, as shown in Fig.4, which extends Eclipse platform with tools for the definition, editing, deploying and validating the processes. With the tool, application developers can design processes according to various business requirements. Users create service composition in what you see what you get, defining the data, the operations and the routing rules among these operations, and specifying the binding of applications and data for each operation. The applications with expected functionalities for binding can be attained through the service search engine.
There are mainly two subjects in the verification of service composition, of which, one is to verify the correctness of the internal logic of service processes, and the other one is to verify the compatibility between the services involved in the composition. The former one is similar to that in tradition workflow area, and many researchers follow the approach for workflow, to verify the reachability of each state, terminability of processes, deadlock and livelock of processes and so on. The latter one is mainly to verify whether services in the composition can interoperate correctly and successfully. Three main levels of such verification can be distinguished: (1) syntax-level compatibility; (2) semantic-level compatibility; and (3) behavioral-level compatibility. Syntax-level compatibility deals with the verification of format and number of messages to achieve the successful interoperation at the syntax level, for example, the number and format of output messages generated from a former operation need to be compliant with that of input messages required by a successor. Semantic-level compatibility verifies that interacting services are semantically consistent in functionality, the meaning of parameters, and the content of messages. For example, an input message required by a successor can be matched to an output message generated from an anterior service with consistent semantic meaning. And behavioral-level compatibility verifies whether services can accomplish the interactions with others on the premise that it follows its own business logic.

3.5. Services Execution and Monitoring

In addition to the support of query/design time of services for consumers, MSI service community also supports the management of runtime for services, including services execution and supervision in operation.

The service execution engine provided by MSI service community is mainly responsible for parsing process models, generating instances of service processes, operations and data, and then controlling the routing of operations and data in a process as well as the invocation of corresponding services. It also provides some advanced management like fault-tolerance, security and transactional recovery, as
well as some interfaces to facilitate users attain the result of service invocation, the operation state of services, data instances and so on.

On the other hand, the monitored control tool provides users with an interface interacting with service execution engine in a visualized way. The process instance is lay out as the same structure of its designing in service orchestration, and its operation condition is demonstrated by distinguishing the executed, executing and unexecuted operations in different colors. Historical data for executed operations can also be looked over in details, such as the actual values of inputs/outputs and the executed time of operations. Moreover, the tool allows users to manipulate on instances of processes, operations and data. For example, users can modify relevant data instances, or change the routing path of the process instance to enforce the process jump to a specific operation without regard to the process definition. Fig.5 shows the interface of the monitored control tool provided by MSI service community.

![Monitored control tool of MSI service community](image)

**Figure 5.** Monitored control tool of MSI service community

4. Case Study

AliSoft (www.alisoft.com) is a rich media Internet-based application platform, offering business management solutions targeting small businesses across China. It allows various software applications, business tools and services across various industries in different types to be integrated. In the future, AliSoft platform will aggregate thousands of products and services developed by a fast growing of independent software vendors (ISVs) in China, and construct a complete value chain for customers with the support of business integration or application association among multiple of single products and services. Based on the Software as a Service (SaaS) model, AliSoft allows service consumers to choose the needed software and services freely, customize them flexibly and rent for use easily, thus the application value can be promoted largely while the input cost can be saved to maximum. And moreover, ISVs can utilize intensive customer and sales resources owned by AliSoft, and find out the
exact requirements and target customers, so as to accelerate the promotion of products and brands, and the expansion of market share.

In order to achieve the vision as early as possible, AliSoft in association with our lab have promoted a series of research subjects on “software store” platform. And our service community for services lifecycle management has been applied in the test bed of AliSoft service integration platform. The main advantages shown in the application include:

1. On one hand, ISVs can register services with more accurate semantic descriptions on functionalities, interfaces, and internal processes for better understanding; and on the other hand, service consumers can find the desired services in a more accurate way by matching requests against semantic interface and behavioral descriptions. Experimental results show that the average recall and precision of semantic interface-based discovery have been increased by 58.4% and 54.6% compared to keyword-based discovery respectively [12], and that of semantic behavior-based discovery have increased by 9.75% and 32.7% compared to semantic interface-based discovery [6].

2. On one hand, we create and update inverted indexing for registered services periodically; and on the other hand, the established indexing is utilized to increase the efficiency of filtering out irrelevant services, so the consumers can benefit with a rapid response. Experimental results show that the response time of inverted-indexing-based discovery is always less than that of sequential matchmaking, in either case when the average length of indexing increases or when the number of services increases. The performance of inverted-indexing-based discovery is especially excellent when the average length of indexing is not large and there are a large number of services. Detailed information on the experiment can refer to [13].

3. With the support of MSI service community, the test bed of AliSoft platform provides users a more intuitive way to get an overview of a collection of services, with which users can easily find out the distribution of the collection and the best ones with high quality. At the back of the visualization of multiple services, there is a clustering algorithm for the automated categorization of services. Experimental results show that the accuracy of the recognition of categories that a service belongs to can reach to 80% in average. Detailed information on the experiment can refer to [14].

4. The test bed of AliSoft platform not only provides a visualized service composition designer, where users can design the routing of a process and the binding of involved services by themselves, but also an automated service composition mechanism when a service request cannot be satisfied by single registered services. Users are more likely to find a possible candidate by exploring service composition. Moreover, both kinds of compositions will be validated with its correctness of process logic and the compatibility of involved services, so users can subscribe and use them with an easy mind. Experimental results show that the recall rate of composition-based discovery has increased by 5.24% compared to atomic-based discovery [13].

5. The integrated service execution and monitoring module facilitates users to redirect to the servers where services are deployed for invocation, and helps users interfere into the execution of services in a visualized way when appropriate operations are needed. Moreover, the feedback of user experiences for service usage can be submitted into the platform to make the data on quality of services fair.
5. Future Work and Conclusions

Current SOA repositories are far from meeting the demands for service lifecycle management in modern service industry. In this paper, we propose a framework for services lifecycle management with many advanced capabilities covered, including services registration and annotation with semantic information, flexible categorization and clustering of services, accurate and efficient services query and matchmaking based on interface and behavioral semantics, service orchestration and verification, and service execution and monitoring. The framework is implemented as a platform called MSI services community, and applied and verified in a test bed of AliSoft service integration platform. Future work includes designing more advanced functionalities according to application requirements and applying the proposed service community to more platforms to verify its performance.

References


© 2010 by the authors; licensee IEIT, Italy. This article is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/).