International Journal of Gird and Distributed Computing
Vol. 2, No. 4, December, 2009

An Architecture and Supporting Environment of Service-Oriented Computing Based-on Context Awareness

Tianxiao Ma, Jun Huang, and Gang Wu

Shanghai Jiao Tong University Dongchuan Road,
800 Num, Shanghai, China
Matianxiao17951@hotmail.com, sbdwhj@gmail.com, wugang@cs.sjtu.edu.cn

Abstract

Service-oriented computing (SOC) is emerging to be an important computing paradigm of the next future. Based on context awareness, this paper proposes an architecture of SOC. A definition of the context in open environments such as Internet is given, which is based on ontology. The paper also proposes a supporting environment for the context-aware SOC, which focus on services on-demand composition and context-awareness evolving. A reference implementation of the supporting environment based on OSGi[11] is given at last.

Keywords: *Service-oriented computing, Context awareness, Architecture*

1. Introduction

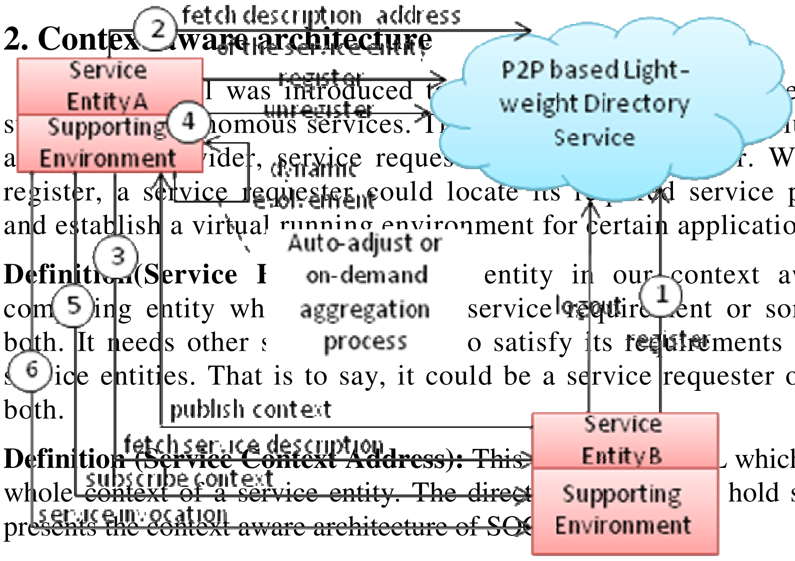
Open, dynamic and hard to control are inherent attributes of Internet platform. These attributes bring challenges to develop, deploy, run and maintain software in such a platform. In order to tackle these challenges, service-oriented computing (SOC) has emerged as a promising computing paradigm for the next future. In SOC, services are autonomous and platform-independent computational elements which can be described, published, discovered, orchestrated, and deployed with standard protocols[1]. Under this paradigm, the most attractive thing is to let autonomous stand alone services cooperate with each other by certain kind of mechanism which will finally generate a distributed computing network.

To accomplish on-demand service composition and context-aware evolving, many aspects should be studied including context representation, context-aware mechanism, self-adaptation of services, and co-evolution among services.

Context description and collection are the basis for auto-evolvement of SOC style application. Therefore, this paper gives a definition of the context[13] based on ontology, including the static description and dynamical runtime information of services. Then a context-aware architecture is proposed in this paper. The mechanism of service on-demand aggregation and dynamically evolvement under this architecture are described. Moreover, a design of the supporting system which will work as a middleware to help build the application under this architecture is given. Finally we give a reference implementation of the supporting system.

The remainder of this paper is organized as follows. Section 2 proposes the context-aware architecture, and explains how the autonomous services composed together cooperate and evolve dynamically with the awareness of the context changing. Section

3 gives the detailed definition of the context ontology. A model of the supporting environment base on the architecture is presented in section 4. The reference implementation of the environment is given in section 5. Section 6 discusses the related work and section 7 concludes the paper with future work.



2. Context-aware architecture

[illegible] was introduced t[illegible] relationship between the s[illegible] autonomous services. T[illegible] includes three actors which a[illegible] provider, service reque[illegible]r. With the help of service register, a service requester could locate its [illegible] service providers automatically and establish a virtual running environment for certain application requirement.

Definition (Service E[illegible] entity in our context aware architecture is a computing entity wh[illegible] service requirement or some serving abilities or both. It needs other s[illegible]o satisfy its requirements as well as serves other service entities. That is to say, it could be a service requester or a service provider or both.

Definition (Service Context Address): This [illegible] which can be used to fetch the whole context of a service entity. The direct[illegible] hold such addresses. Figure 1 presents the context aware architecture of SOC.

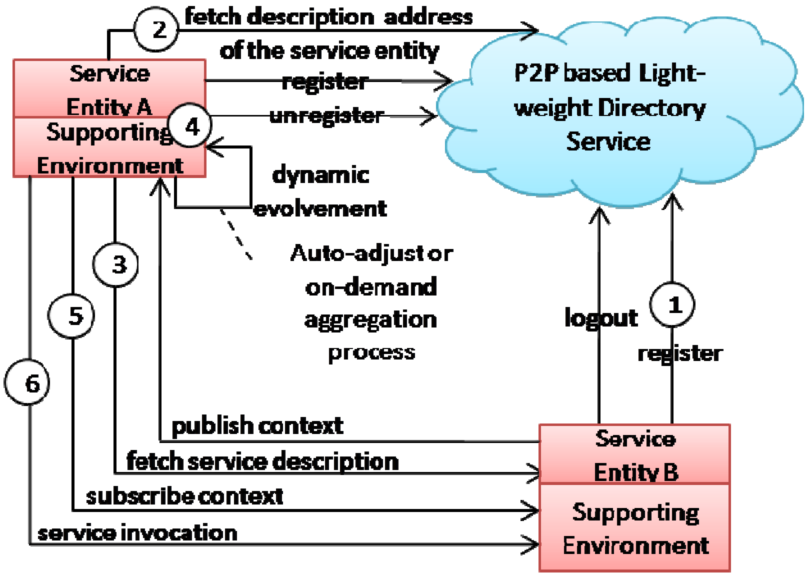


Figure 1. Context-aware Architecture for SOC.

A typical running scene under this architecture is shown as follows. (1) Service Entity B registers itself to the light-weight directory service. (2) After registration, Service Entity A looks for candidate services which can satisfy its requirement through the directory service. The result returned from the directory service is the addresses of context description of candidate service entities. (3) Entity A fetches the context of each candidate service, which includes static information such as ability description and dynamical information such as runtime status. (4) Entity A compares the acquired context with its own requirements to choose a suitable service entity and then builds an aggregation relationship with the selected service entity. Repeating the previous steps several times will form a virtual running environment (VRE). (5) After that, Entity A will subscribe to context transformations which it cares about from every service entity in the VRE. (6) Start the execution of the application by Entity A. If the subscribed context transforms during the execution, it will be published to Entity A. Entity A will adjust the execution according to the transformation based on some rules, such as looking for a new entity which will satisfy the requirements. The VRE evolves dynamically then.

2.1 Light-weight directory service

In this paper, the directory service is light-weight because the information stored by the directory service is not the whole context of a service entity but an address of the context.

The context which can influence the composition of service entities includes both static and dynamical information such as function description, interface, running status and environment description. Even the static information would change during the service entity's lifecycle. If all these information are maintained in the directory server, it will cause bandwidth consumption and heavy loading pressure. As a result, we just store the address instead of the complete context in the directory server since it is relatively stable.

Centralized directory service is also not suitable. For services running on the Internet platform it will become the performance bottle-neck of the whole system. In this paper, P2P technology is used to construct a distributed directory server. The service name is recognized as hash information. Each directory server acts as a P2P node. The address of the context is stored on corresponding server by using the chord algorithm[7].

2.2 Service discovery and on-demand aggregation based on context

As mentioned in the previous section, the directory service will store the context address of various service entities. So, through these addresses we can get complete context information of other services entities, which include the services provided by those service entities and related non-functionality abilities. By comparing such context information with the context that belongs to the service requester, we can finally determine which service is most suitable for the service requester. When the context of the service provider transformed, the service requester will reevaluate the suitability of the service provided by the service provider based on the transformed context and determine whether or not to choose another more suitable one.

In the SOC context, an application requirement will be accomplished by several composited service entities. The service entity which issues the requirement is called the main service entity which is responsible for discovering and compositing relative service entities according to their context and the application requirements. The VRE will then be formed after the on-demand integration process. This could be a nesting process, that means service entity A which was chosen by the main service entity might initiate a new process to discover and composite other service entities to help itself accomplish the service used by the main service entity.

This paper focus on how to realize on-demand integration based on context, so we make following assumptions. One is that we only concern which elements in the context will be used to match the application requirements, the exact value of the element is not concerned. We assume that among service entities there is a same semantic interpretation between the value of the context elements and the value of the requirements. For example, the main service entity's functional requirement is "order", and the candidate service entity's context has a functional description called "purchase". The problem whether the two terms have the same meaning is out of the scope of this paper, it could be answered by ontology reconciliation or other same technology [2]. Another assumption is that the service provider and requester can reach a consensus on the service interface on semantic level. When the requester gets the interface description from the context, it knows exactly the meaning of each parameter and the return value. Context based service discovery is accomplished by using the light-weight directory service.

2.3 Dynamic evolvement based on context awareness

The VRE is dynamically changing with the transformation of the context of each involved service entity. This paper introduces a subscribe/publish mechanism to support context awareness. As soon as the composition relationship is formed, the service requester will subscribe to its interested contexts from the selected service entity. When the transformed context of the service provider is published, the service requester can be aware of such transformations and it will check the updated context of the service provider to see if the service provided by the service provider is still satisfy its requirement and if it found the service can never satisfy its requirement it will begin to self-adjust or start a new composition process to implement dynamic evolvement.
Self-adjust behaviors include waiting for some time to see whether the context of the service provider will transform or not. If the context transformed, the service requester will reevaluate the context to decide whether the service provided by the service provider can still be used or not. If the result is positive, there is nothing needs to be changed, or a service discovery process will be launched by the service requester to found new services. The user can also specify their own self-adjust behaviors and associate such behaviors to designated contexts, when these contexts transformed, related user defined self-adjust behaviors will be activated. The context awareness and dynamic evolvement process is transparent to the application user.

3. Service ontology

In an open environment, an explicit and semantic context description for services is needed as the foundation for the evolvement of services and services composition. We have defined an internet ontology use RDF/OWL [10] to model service context. For detail about internet ontology, please see [13]. The SOC supporting environment proposed by this paper will use this ontology to represent and maintain the context related to service entities. Note that the service context ontology could be extended and improved continuously.

4. Supporting environment of context-awareness service composition

The supporting environment for context-awareness services composition is composed of context aware supporting system and runtime supporting system shown in Figure 2.

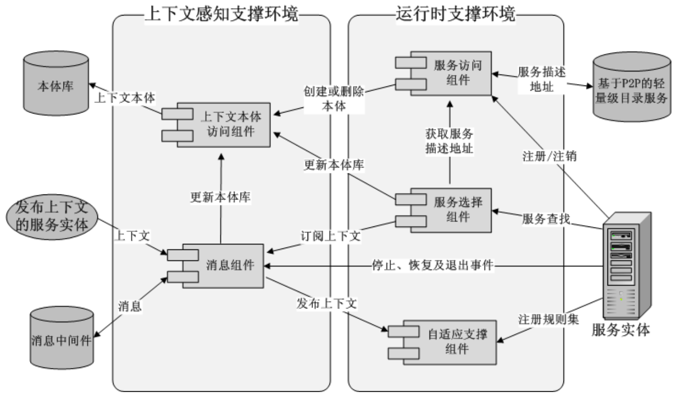


Figure 2. Supporting environment for context-aware service composition

The context aware supporting system is responsible for collecting, managing, subscribing and publishing context, to give fundamental support for context awareness of the on-demand service composition. Runtime support systems include life cycle management, service invocation and auto-adaptation framework, which provided runtime support for the service entity. The supporting system runs as a middleware for every service entity. The functions of the main components are introduced as follows.

- **Server access component.** This component is used to communicate with the P2P-based directory server, to complete the registration and logout of the service entity, and to fetch the context description address of service entities.
- **Service selecting component.** This is the core component to implement on-demand composition, used by the service entity to select suitable services. With the help of the server access component it gets a list of contexts of candidate service entities and selects a suitable one from them based on the context. After the service provider is chosen, this component will use the context aware supporting system to accomplish two things. One is to update the run-time context ontology library to show which requirement is satisfied by which service entity. The other is to subscribe to the interested context elements from the service provider.
- **Auto-adapt supporting component.** This component is the receiver and processor of the context transformation. In this paper, we use a rule based reasoning mechanism[4] to construct the auto-adapt supporting component to make the service entity evolve dynamically. The service entity can register a set of customized rules to the component for responding to transformation of specific context elements.
- **Context ontology access component.** This component is responsible for creating and maintaining the run-time context ontology library.
- **Message component.** This component is used to maintain the context subscribing relationship. When the subscribed context element changes, it will be published to relevant service entities through message oriented middleware. In this paper we use the Probe-Gauge model in the Rainbow system[8] to acquire the context transformation, and use the Context Provisioning method in paper[9] to transfer the message.

5. Reference implementation

To validate the work of our paper, we implement the supporting environment based on OSGi[15]. Each OSGi platform holds several bundles and each bundle represents a single service entity. A bundle need to register itself to a OSGi platform along with its service description address as parameter, so other bundles can get the service description based on the address and to check whether to use it or not. We also use R-OSGi[12] to integrating stand alone OSGi platform to form a OSGi platform cluster. The context will be collected from three different sources: the environment, the service entity itself, and other service entities. The context of other services will be collected through the publish-subscribe mechanism.

In our implementation, we treat scalability as the most important part of our supporting environment. We believe our supporting environment can suitable for as many application scenarios as possible. So, many important components in our reference implementation can be extended easily. These include local and remote probes, persistent strategy, context acquisition strategy, service selection strategy and self-adaptation strategy and so on. We also separate the context awareness support system and the runtime support system, so both can be used with other context awareness strategies or on-demand aggregation and self-adaptation strategies. For more detail, please see [13].

6. Related work

Two key issues need to be tackled when implementing a context-aware service-oriented system. The first one is how to describe the context and second one is how to establish and maintain composition relationships based on the acquired context.

For the former issue, ontology seemed to be the most popular answer for its widely acceptation and semantic functionality. There are two different ways to using ontology to represent context. The first way is to allow independently designed ontology exist and then use ontology reconciliation[2] to hide the difference among ontologies. The second way is to define a unified ontology, and use this ontology as a standard to represent the context. This strategy eliminates the need of reconciliation. The problem of the former way is its complexity and imprecision, as the structure of different ontology may vary very much. And the result of the reconciliation is dependent on the algorithm used. The problem of the second is it is impossible to define an ontology which can represent every concept in every area. In this paper, our strategy to represent the context is much more like the second way aforementioned. The difference is that we are not trying to propose an ontology which includes everything, but we provide an extensible one. If somebody finds the original one is not sufficient, it can be extended easily.

Toward service composition, many research efforts have been put into service on-demand composition and auto-adaptive evolvement. The traditional way to composite services need people to specify composition rules first, for example, specifying the functional and non-functional requirements of each service which form the composition service. It makes the composition service difficult to adapt to the changing environment. This approach is clearly not suitable for a SOC paradigm, since applications are rarely constructed from scratch, but are always composed from existing service components. So, an application transparent method needs to be proposed for this type of service composition. Moreover, such a method also needs to monitor the execution status of the composition service and dynamically adapt the behavior of the composition service response to the rapid changing environment.

Some research focus on how to composite different services based on a context awareness way. Paper [3] and [4] use historical information to aggregate services to form a composition service. Paper [3] proposes an application-specific middleware to composite services adaptively based on a role structure. An organizer is responsible to compose different service based on some QoS information. This method is more suitable for coarse granularity services. For a service which provides the adding operation, it is hard to relate such fine granularity service to a role. Paper [4] uses data mining technology for service selection. The invocation history of a service will be recorded in a warehouse, and during the selection process, the historical information will be used to generate a service selection model. A service will finally be selected based on the user defined quality goals together with its history records.

Agent based solution are also used to tackle the problem of service composition and autonomous evolvement[5][6]. But until now, it is still very hard to construct agents, and this will bring significant complexity to systems for adopting such methods.

The supporting environment provided by this paper aims to balances application transparency and implementation complexity for context-aware service composition. The supporting environment proposed in this paper will support the whole life cycle of context-aware composition service, from service selection to composition service evolvement. With low invasiveness and high extensibility, our supporting environment will satisfies the requirement of context-awareness service composition.

7. Conclusion

This paper focuses on context aware SOC. We concluded which elements should be included in the context of the Internet computing environment and gave a definition of the service context based on ontology. A context-aware architecture for SOC is proposed. This paper explains how the service entities are aggregated together to cooperate under this architecture and evolve dynamically with the awareness of the context changing. Furthermore, the structure of the supporting environment is given and the function of each component is described. The supporting environment acts as the middleware to support the implementation and execution of the context-aware service composition. Finally, a reference implementation is proposed.

The future work includes improving the service context ontology and our reference implementation. We will make our reference implementation support P2P based addressing and provide a better service selection.

References

[1] M.P.Singh and M.N.Huhns, Service-Oriented Computing : Semantics, Processes, Agents, John Wiley & Sons, 2005

[2] Jingshan Huang, Jiangbo Dang, Michael N. Huhns. Ontology Reconciliation for Service Oriented Computing. IEEE International Conference on Services Computing (SCC'06), 2006, pp.3-10

[3] Alan Colman, Linh Duy Pham, Jun Han, Jean-Guy Schneider. Adaptive Application-Specific Middleware. MW4SOC, 2006, Vol. 184

[4] Fabio Casati, Malu Castellanos, umesh Dayal, Ming-Chien Shan. Probabilistic, Context-Sensitive, and Goal-Oriented Service Selection. International Conference On Service Oriented Computing, 2004, 316 - 321
[5] Pablo Rossi, Zahir Tari. Software Adaptation for Service-Oriented System. MW4SOC 2006.
[6] Zakaria Maamar, Soraya Kouadri Mostefaoui, Hamdi yahyaoui. Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. IEEE Transactions on Knowledge and Data Engineering, Volume 17 Issue 5
[7] Stoica, I., Morris, R., Liben-Nowell, etc. Chord: a scalable peer-to-peer lookup protocol for Internet applications. IEEE/ACM Transactions on Networking, 2003,11(1):17 – 32.
[8] Garlan D, Cheng S W, Huang A C. Rainbow: Architecture-based self-adaptation with reusable infrastructure. Comput, 2004, 37(10): 46-54
[9] Pavel D, Trossen D. Context Provisioning for Future Service Environments. International Conference on Computing in the Global Information Technology, 2006, Page(s):45 – 45.
[10] http://www.w3.org/2004/OWL/
[11] http://www.osgi.org.
[12] http://r-osgi.sourceforge.net/
[13] http://casei.googlecode.com/

Authors

Tianxiao Ma, Jun Huang

Master student of Shanghai Jiao Tong University, Shanghai, China. mainly focus on distributed computing and other related areas and technologies.

Gang Wu

Professor of Shanghai Jiao Tong University, Shanghai, China. Research area includes distributed computing, grid computing, service-oriented computing and related areas.