Int J Software Informatics, Vol.4, No.4, December 2010, pp. 473–493 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics, ISSN 1673-7288 http://www.ijsi.org
 Tel: +86-10-62661040

Middleware Support for Internetware: A Service Perspective

Chunyang Ye[1,2], Jun Wei[2], Hua Zhong[2], and Tao Huang[2]

[1] (Middleware Systems Research Group, University of Toronto, Toronto, Canada)

[2] (Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract The advent of Internet technology introduces a revolution to software application and development paradigms. Traditional software development and application patterns have been shifted to Internet-based service sharing and collaboration among partners across the Internet. This new paradigm imposes new challenges and complexity in the lifecycle of software development, deployment, execution and maintenance. Middleware, an intermediate layer to abstract the homogeneity and hide the difference of underlying systems, can be used to reduce the complexity of managing the lifecycle of Internet applications. In this paper, we exploit the needs of middleware support for Internet-based applications from a service perspective. We investigate the potential requirements and features of Internetware, and analyze the state-of-the-art solutions. We also discuss the remaining issues and their challenges, and explore the potential future research directions.

Key words: internetware; service; middleware; cloud computing

Ye CY, Wei J, Zhong H, Huang T. Middleware support for internetware: a service perspective. *Int J Software Informatics*, **2010, 4(4): 473–493. http://www.ijsi.org/1673-7288/4/i65.htm**

1 Introduction

The invention of Internet technology has introduced a revolution to our daily life. Today, the dream in last century that every one owns a PC (personal computer) has been shifted to the reality: more and more people are connecting to Internet to share their information and services. Internet services like E-mail, search engine, to just name a few, are becoming an indispensable parts of our daily life. Nowadays, Internet is regarded as an open platform to share information and services. More and more Internet services are being developed to facilitate our daily life.

The huge benefits brought by Internet technology however also impose great challenges on software development, deployment, execution and maintenance for Internet services. Compared to traditional software development and application patterns, Internet-based applications are no longer running in a few restricted individual computers. Instead, an Internet-based application may involve thousands of millions of

* This work is sponsored by the National Natural Science Foundation of China under Grants Nos.60903052, 60773028, 90718033, the National Basic Research Program of China under Grant No.2009CB320704, and the National High-Tech Research and Development Plan of China under Grant No.2007AA010301.
Corresponding author: Chunyang Ye, Email: cyye@otcaix.iscas.ac.cn
Received 2010-08-25; revised 2010-11-30; accepted 2010-12-25.

users connected to each other at the same time. Moreover, the development of an Internet-based application is no longer limited to a small group of developers. Users from all over the world may publish their developed software components as services and share the services to others. By searching and composing suitable services, users can easily develop and customize their own applications. This new software development paradigm is also referred to as *service-oriented architecture*(SOA)[64].

To address the challenges and reduce the complexity for the development of Internet applications, middleware, an intermediate layer lying between the low-level hardware (and operating systems) and high-level applications[21], is usually deployed. As illustrated in Fig. 1, by abstracting reusable solutions to hide the differences of underlying systems (hardware and software), middleware provides software developers an interface to access and manage the underlying systems in a simple way. With middleware support, software developers only need to concern about the application logic and delegate the general tasks of handling the heterogeneity of underlying systems to middleware systems. As a result, the complexity of application development is reduced.

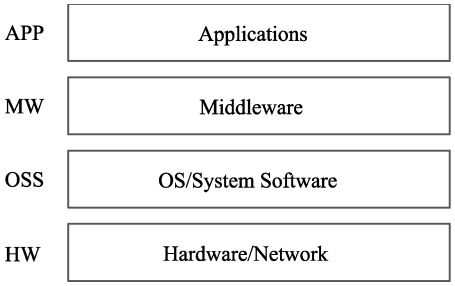

Figure 1. Middleware architecture

Nowadays, middleware technology has been widely used in applications in different domains[6,11,16,9,10,19,27,53,48,54,67,74,89,90]. For example, to increase the reliability and reduce the complexity of communications, message-oriented middleware systems[96,33] have been developed to abstract the underlying communication protocols, providing applications an interface to send and receive messages easily and reliably. To further decouple the message sender and receiver, publish/subscribe middleware[16,23] disseminates messages automatically for high-level applications based on topics or contents of interest. Application servers like J2EE[69] or .NET[51] further encapsulate the database systems and other common functionalities as services for Internet applications (e.g., transaction processing). In this way, Internet applications can be easily developed and deployed to application servers as Web services accessible to service consumers.

From a service perspective, the aforementioned middleware systems provide different services to high-level applications (e.g., communication support, transaction processing etc). To consume the services provided by middleware systems, high-level application developers usually deploy middleware systems in their own organizations, and run the high-level applications on top of the middleware systems. Therefore, the middleware systems are usually owned by high-level application developers and under the control of the organization that the high-level applications belong to.

Figure 2. Architecture of cloud computing

Today, the vision that a middleware system is a service is further extended by the concept of cloud computing[1]. In cloud computing, as illustrated in Fig. 2, not only the high-level Internet applications are regarded as services, but also the underlying infrastructure, software platform and resources (hardware and software), data center and middleware are shared as services. From the perspective of a high-level Internet application provider, it can rent and use some shareable services that provide the computing capacity and resources to run the Internet applications instead of hosting the applications in its own servers. From the perspective of software platform and hardware infrastructure providers, they provides the necessary computing utility and resources (hardware and software) to support the high-level Internet applications whose developers do not have to own the servers and infrastructure. In this paper, we restrict our discussions to the set of middleware systems lying between the high-level Internet applications and underlying computation cloud. Such middleware systems can be hosted on some rented computation cloud, and serve as public value-added services for high-level Internet applications. We also refer to *high-level applications* as the set of Internet applications for end users (e.g., online shopping services).

This new vision brings new challenges for middleware support for Internet applications. Since middleware systems are regarded as public services shared by high-level applications from different organizations, they are no longer under the control of the organizations that the high-level applications belong to. Instead, high-level applications consume the public services provided by middleware systems through restricted service interfaces. High-level applications can select suitable middleware services based on their requirements and may migrate from one middleware service to another when the requirements change. This imposes new requirements and challenges for middleware design in the cloud computing environment.

In this paper, we exploit the requirements and challenges of middleware support for Internetware in the context of cloud computing. We regard middleware as a public service and illustrate what are needed for the middleware services to facilitate high-level applications and the corresponding challenges. We review the state-of-the-art solutions and analyze what are missing in existing work. We also explore some potential future directions in middleware research for Internetware.

The rest of this paper is organized as follows: Section 2 investigates the requirements and challenges of middleware support for Internetware with an example. Section 3 reviews the state-of-the-art solutions and Section 4 investigates the remaining issues. Section 5 concludes the paper with some discussions on the potential future research directions in the area of middleware support for Internetware.

2 Requirements and Challenges

In this section, we exploit the requirements and challenges for middleware support

for Internetware using an example.

2.1 Application scenario

Let us consider an e-commerce application scenario where customers can purchase products online via shopping services, as illustrated in Fig.3. In the application scenario, customers can order the products online and pay the bill by credit card. Then, the ordered products are delivered to customers by a shipping company. To support such an application, several fundamental Web services are developed by different organizations and composed together: Online shopping services provided by merchants, banking services provided by banks, shipping services provided by express delivery companies. These services collaborate to implement the e-commerce application. The following are some abstract descriptions of these services.

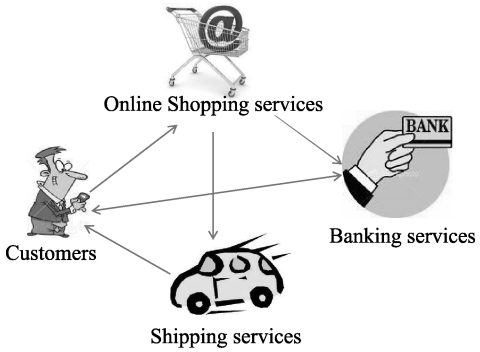

Figure 3. E-commerce application scenario

Online shopping services: To provide better services for customers, the online shopping services are hosted in different locations and customized for local customers. To reduce the costs of maintaining the servers in different locations, the shopping service providers rent public servers to run their services. The online shopping services also customize different regulation policies for different countries due to the local cultures and politics. Since customers with different backgrounds may have different shopping preferences, the online services may be customized to highlight the products that are mostly purchased by the local customers in the website. In addition, to attract more customers, the online shopping services may broadcast comments on the products from the customers to their friends based on their online social network relationships.

Banking services: To provide high-quality services for customers, the banking services may want to improve the security of their credit card services on the one hand, and provide more flexible and smarter interactions with customers on the other hand. Let us image a scenario wherein a customer travels to another country, and goes shopping online with a credit card. When the customer submits a payment request, the banking services find that the payment request is from another country. To improve the security of credit card services and protect the interests of card holders, the bank services may call the customer to confirm the transaction before approving the payment. However, the customer may decline to take the call since his/her cellphone is operating under expensive roaming charges. As result, the transaction is

rejected with all the negative consequences such a situation entails. The bank services may apply a more flexible and smarter solution. For example, if a customer travels to another country, the bank may choose to send a short message to the customer instead of dispatching a phone call. Similarly, if security concerns about the credit card are raised during the weekend or a holiday, the bank may contact the customer via his/her cellphone instead of the office phone.

Shipping services: To shorten the delivery time, the shipping services may assign the delivery task to the branch nearest to the current location of the customer. In addition, the shipping services may also need to schedule the delivery tasks proactively to provide services with high qualities. For example, the shipping services may prepare in advance more trucks before important festivals because more delivery tasks are usually needed.

To develop and compose the services in this application scenario, middleware systems are needed to manage and reduce the complexity during the lifecycle of the application. In the next section, we analyze what kinds of requirements are needed for the middleware systems.

2.2 Requirement analysis

In this section, we analyze the requirements for middleware systems to support the development and deployment of services in the aforementioned application scenario. We analyze the requirements of middleware systems from the perspective of being public value-added services for high-level Internet applications (that is, what requirements are needed for developing and running high-level Internet applications based on the middleware services). We classify the requirements into two categories: functional requirements and non-functional requirements.

Functional requirements: Since middleware systems are regarded as public services shared by different Internet applications, the functionalities the middleware systems should provide include:

1. **Context-awareness**: In the above example, all the three services are context-aware[15,87], that is, the services can adapt their behavior based on the environmental contexts. For example, the banking services choose different ways to notify card holders based on the location (traveling or not) of card holders and time (workday or weekend). Similarly, the shipping services select the nearest branch to deliver the products according to the current location of customers. In this example, information like locations and time etc are explicitly modeled as contexts. Since the tasks of sensing and managing contexts can be reused by many Internet applications, the middleware systems should provide context-awareness support for Internet applications, including gathering raw context data, aggregating contexts, reasoning contexts, and managing contexts for applications.

2. **Asynchronous communication**: In the example, the three services communicate with each other via message exchange. Middleware systems should provide reliable and dependable communication support for the services. In addition, asynchronous communication among services should be supported. For example, if the underlying network is disconnected during the communication, the

messages should be kept by the middleware and re-sent when the network is re-connected. Similarly, the customers may travel from one place to another. During the traveling, the messages sent by the bank services for the customers in one location should be accessible when the customers get accessed to the services again in another location. This requires the middleware systems to provide reliable and flexible asynchronous message routing services.

3. **Loosely coupling support**: In traditional service compositions, the number of participants is usually fixed. Services usually communicate with each other in a one-to-one way. However, in the cloud computing environment, the collaboration among services is more open and flexible. The number of participants is usually not fixed and unknown in advance. Services can join and leave the collaboration from time to time. For example, the online shopping services may provide online auctions for customers. Different merchants can join the auctions whenever they have something to sell. Similarly, customers can join or leave the auctions whenever they intend to. In such scenario, collaborating services are loosely coupled: services need not know who they are collaborating with; instead, they only need to publish their messages (e.g., bidding prices) to all the collaborators and subscribe to interested messages (e.g., the bidding price of some product is higher than some particular value) from others. Therefore, a loosely coupled mechanism to coordinate the collaboration among high-level services is needed.

4. **Flexible service discovery and integration**: In the above example, the middleware systems need to provide functionalities for services to discover and select collaborating partners in traditional way[64]. Besides that, more flexible service discovery and integration mechanisms should be provided. For example, the online shopping services can make use of the social network relationships to advertise the products to the friends of customers who share similar interests (e.g., users in Facebook can share information and comments on their favorite things to their friends). This requires the middleware systems be able to integrate third-party application data to facilitate service discovery and advertisement.

5. **Monitoring**: Since the middleware systems are public services and beyond the control of high-level applications, the middleware systems should provide the functionality for application developers to monitor and query the running status of the applications. For example, the online shopping service owners may need to check how many times each product is viewed and queried by customers. The middleware systems should be able to monitor the data and provide summary and statistical results to high-level application owners.

Non-functional requirements: Besides the above functionalities provided by the middleware systems, the following non-functional requirements are also needed:

1. **Usability**: Since the middleware systems are regarded as public services for high-level Internet applications, the middleware systems are no longer under

http://www.facebook.com/

the control of the high-level Internet application developers. That is, the high-level application developers need to invoke remote middleware services to deploy and run their applications. To do so, the middleware services should provide additional supports for application developers to develop, deploy and maintain their applications running on the middleware systems. For example, the middleware systems should provide mechanisms for developers to remotely test their services, debug and track the faults in their services.

2. **Dependability**: High-level applications running on the middleware systems may fail due to defects or network disconnections. This requires the middleware systems should provide mechanisms to restore the high-level applications from error status to normal status, and prevent value loss for the applications due to the failures. For example, the middleware systems may need to provide transactional support for critical tasks for the high-level applications. Other mechanisms like fault-tolerance may also be supported.

3. **Quality of service**: Since middleware systems provide public services for high-level applications, the quality of their services is also a major concern. In such setting, one middleware system may be shared by different applications. Some mechanisms are needed to guarantee the quality of each high-level application running on top of the middleware system. This includes how to allocate resources for different applications and how to isolate the interference between different applications. For example, suppose two online shopping services are deployed on top of the same middleware system. If the number of customers accessing the first online shopping services increases dramatically, the request-response latency for the second online shopping service may be affected (since the first one may use most of the resource and bandwidth). Therefore, some mechanisms should be provided to customize and guarantee the quality of service for each high-level application.

4. **Adaptability**: The middleware systems should also be able to adapt the configuration and behavior for changing requirements. For example, the middleware systems are deployed as public value-added services on some rented computation cloud. To save money, the middleware services may be deployed to only a few hosts in the beginning. However, if the number of high-level applications running on the middleware systems increases, the middleware systems may need to rent more hosts to deploy the services. Or, if the middleware systems on some hosts endure a higher overload, the middleware systems may need to reconfigure the applications running on the middleware systems to balance the workload for each host. This requires the middleware systems should be able to adapt and reconfigure in a transparent way.

5. **Privacy**: Since middleware systems are shared to high-level Internet applications as public services, the middleware systems are no longer under the control of high-level applications. When multiple high-level applications are running on a public middleware service, the privacy of the high-level applications should be protected. For example, the transaction data between an online shopping service and its customers should be invisible to another online shopping service

running on top of the same middleware system, unless the first service and the customers agree to share the information. Therefore, a mechanism to convince customers and guarantee privacy-protection is needed.

6. **Security**: In addition to privacy, security is also a major concern for a middleware service. Since multiple applications may be running on top of the same middleware system, the information of one application being leaked and destroyed by other applications should be prevented. As public services, the middleware systems should also be able to protect the applications from denial-of-service attacks and other intrusions.

2.3 Challenges

As mentioned in Section 1, in the cloud computing environment, the middleware systems are regarded as public services shared by high-level Internet applications and are no longer under the control of the high-level applications. This imposes new challenges for the middleware systems to fulfill the above requirements.

1. The first challenge lies in how to provide development, deployment and maintenance support for high-level applications. Since the middleware systems are not owned by and placed in the same organizations that high-level applications belong to, application developers need to develop their applications locally and deploy the applications in remote middleware systems based on the service interfaces provided by the middleware systems. This requires the middleware systems provide not only the basic functionality to support the running of the high-level applications, but also the functionality to support the testing and debugging of the high-level applications. However, the middleware systems are shared by high-level applications from different organizations. How to extract the necessary information for application developers to test and track the bugs whereas protecting the privacy of other high-level applications is a challenge for the middleware systems.

2. The second challenge lies in how to balance the concerns of context-awareness, privacy, security and reusability of middleware systems. On the one hand, middleware systems usually abstract and hide the difference of underlying systems and allow high-level applications to access the underlying systems in a transparent way. Such abstraction increases the reusability of the middleware systems for different applications and systems. On the other hand, high-level applications may need to get more context information about the underlying systems to behave adaptively. Different high-level applications may have different requirements on the reveal of contexts from the middleware systems. This may break the well-encapsulation principle of reusable middleware solutions and privacy concerns. How to trade off the different concerns is a challenge in the design of middleware systems.

3. Another challenge is how to guarantee the quality of service for high-level applications. Since the middleware systems are hosted on some rented computation cloud, the number of rented hosts may vary with the number of the high-level

applications running on the middleware systems. How to dynamically schedule and assign the high-level applications to different hosts in a transparent way to guarantee load balance is a challenge. On the other hand, high-level applications may interfere one another due to shared computing resources provided by the middleware systems. How to isolate the impacts of one high-level application on the others is also a challenge.

4. In the cloud computing environment, the number of high-level applications hosted on a middleware system may be huge, or a high-level application may have thousands of millions of customers. For example, an online market middleware system may attract a large number of online shopping services and customers. This imposes a challenge for the dependability of the middleware systems. For example, a middleware system may be hosted on a large number of distributed servers. How to guarantee reliable and efficient routing of messages among these hosts? When some hosts fails, how to recover the middleware system to protect high-level applications from being affected? Different high-level applications may have different error recovery policies, how to guarantee that the recovery of one high-level application does not affect another?

5. In the e-commerce example, the middleware systems should be able to integrate third-party services and data to provide better services for high-level applications (e.g., integrate the social network relationship data from third-party services). However, applications or services are usually developed and provided by different organizations. This imposes challenges for interoperability among different services (e.g., how to coordinate different autonomous services? How to understand the meanings of exchange data? etc).

In the remainder of this paper, we review the state-of-the-art solutions to the aforementioned requirements and challenges. We also point out what are missing in the existing solutions and propose some future research directions.

3 State of the Art

In this section, we review the major middleware systems used today and summarize their characteristics.

3.1 Message-oriented middleware

To provide reliable and asynchronous communication support for Internet applications, message-oriented middleware (MOM) is proposed and developed. The MOM middleware hides the heterogeneity of underlying network and operating systems, and provides simple primitive for applications to exchange messages. The architecture of MOM middleware can be either centralized or distributed. In the latter case, the MOM middleware systems running on different computers form a connected overlay to route messages from the message sender to the message receiver. Applications can access the middleware systems and consume the provided services by connecting to the nodes in the overlay to send or receive messages. The general architecture of message-oriented middleware is illustrated in Fig. 4.

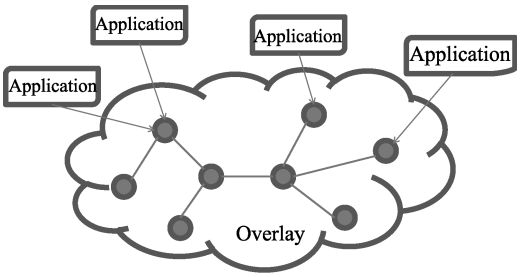

Figure 4. Message-oriented middleware

According to the coupling between the sender and the receiver, the MOM middleware can be classified into two categories: queue-based middleware and publish/subscribe middleware. In queue-based middleware, senders and receivers communicate with each other through sharing the same queue membership. In publish/subscribe middleware, senders and receivers are decoupled. Senders usually advertise to all the receivers what kinds of messages they will publish. Receivers, on the other hand, subscribe to the messages of interest based on the topics or contents of the messages. By matching the messages to the subscriptions, messages can be routed from the senders to the receivers in a way transparent to applications.

Both types of MOM middleware systems are widely used in many applications[16,20,32,33,42,44,58]. Among these MOM middleware systems, the IBM MQ series[32] is a representative of queue-based MOM. Compared to queue-based MOM middleware, publish/subscribe middleware decouples the sender and the receiver and supports many-to-many asynchronous and anonymous message delivery pattern. In topic-based publish/subscribe middleware systems, applications publish messages with different topics. Applications will receive the messages with the same topics that they subscribe to. Apache ActiveMQ[3] is one representative of topic-based publish/subscribe middleware. In content-based publish/subscribe middleware systems, messages are delivered to the subscribers whose subscriptions match the contents of the messages. Well-known content-based publish/subscribe middleware systems include[16,42,45], to name a few.

3.2 Context-aware middleware

In pervasive computing environment, applications are usually aware of the surrounding environment and adapt the behavior automatically according to the environmental changes. Applications usually model the surrounding environment as contexts explicitly and define rules to adapt the behavior when a context changes. Such applications are also referred to as *context-aware applications*[15,87]. To support context-aware applications, context-aware middleware is proposed to manage and manipulate contexts.

Figure 5 illustrates the architecture of context-aware applications and context-aware middleware. A context-aware middleware is responsible for managing different types of sensors, collecting and manipulating context data. By collecting the raw context data from sensors, context-aware middleware can organize and abstract the context data to the format needed by applications. For example, a GPS device may collect the raw data about the longitude and the latitude of a moving device holder.

On receiving such data, the middleware can derive the context about the moving speed of the device holder. By using the interfaces provided by context-aware middleware, context-aware applications can proactively query the surrounding environment or get notification from the middleware whenever a context of interest is updated.

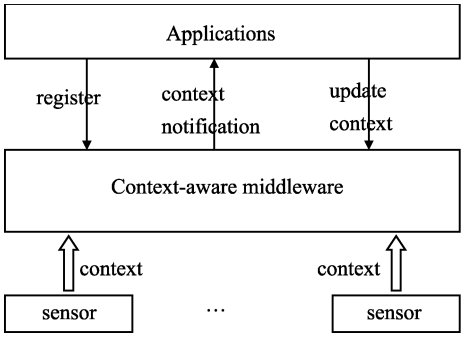

Figure 5. Context-Aware middleware

Recently, many context-aware middleware systems have been proposed to aid the development of context-aware applications. For example, Context Toolkit[17] is a well-known infrastructure for the development and execution of context-aware applications. Context Toolkit introduces context widgets to collect, synthesize and interpret contexts. It decouples the handling and usage of contexts for context-aware applications by assigning the management of contexts to the underlying middleware system. Lime[54] is another context-aware middleware system supporting both logical and physical mobility of agents in a context-aware environment. A coordination model built on top of tuple space is provided for agents to share their contexts and collaborate seamlessly. A similar middleware called Egospace[41] also applied tuple space to share contexts among agents. In addition, some context-aware middleware systems are also proposed with extra specific services. For example, Bellavista *et al.*[6] developed a middleware to manage resource in wireless sensor network. Capra *et al.*[11] proposed to resolve profile conflicts among context-aware applications at the middleware level based on the auction from partners. Chan *et al.*[9] designed a reflective middleware to support dynamic adaptation of middleware and applications. Boldrini *et al.*[5] implemented a middleware system to learn context and social information of users to predict their behavior on opportunistic networks. Other similar existing context-aware middleware systems include[7,27,36,56,68], to name a few.

3.3 Service-Oriented middleware

In service-oriented architecture, application developers can discover and compose third-party services to develop composite services. To support the orchestration of services from different organizations, service developers usually describe the business logic of a composite service as a business process. On the other hand, to support stateful interactions among collaborating services, a choreography to govern the message exchange among the services involved in a service composition is needed. Service-oriented middleware systems are developed to support the orchestration and choreography of services. The architecture of service-oriented middleware is illustrated in Fig. 6.

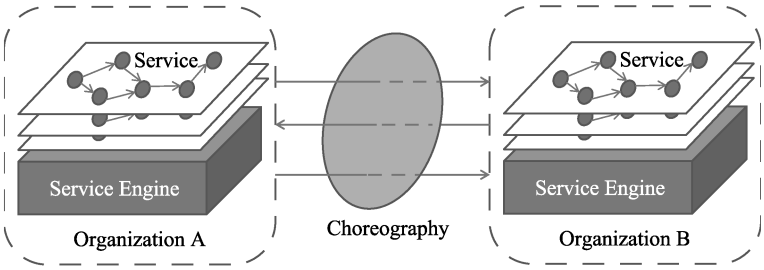

Figure 6. Service-Oriented architecture

Recently, many service-oriented middleware systems have been developed to support service compositions. Aixs2 is a new generation of Apache Web Service middleware supporting the development and execution of individual Web service[59]. Apache Orchestration Director Engine (ODE)[3] is an open-source service engine supporting service orchestration described by business process execution language (BPEL). Nino[45] is a distributed service engine built on top of content-based publish/subscribe middleware. The concept of running services on distributed engines can be extended to Grid Computing middleware[71]. Some industry application servers like WebSphere[35], BEA WebLogic[28], Microsoft .NET platform[51] also embed their own service engines to support service-oriented applications.

Besides providing orchestration and choreography infrastructure support for service compositions, some service-oriented middleware systems also concern about value-added services on how to compose web services with more flexibility and higher quality. This includes the ability to discover and select Web services with required quality, the ability to compose Web services with agility, the ability to check and resolve the incompatibility among services and so on. For example, Zeng *et al.*[89] developed a middleware system to optimize the selection and composition of third-party services with the overall QoS requirements for a service composition based on integer programming. Fujii *et al.*[24] proposed a platform to utilize contexts to support dynamic compositions of services. Foster *et al.* introduced a service to verify the compatibility of services in a service composition based on process algebra[25]. Ye *et al.* also provided a solution to detect and resolve the inconsistency for SOA applications[80,81,82,83]. Other similar middleware systems and services for service-oriented applications can be referred to [13, 22, 34, 40, 55, 72, 77, 75, 86, 91].

3.4 Application servers

To support quick development and deployment of Internet applications, application servers are proposed to encapsulate and integrate reusable solutions to common issues in Internet applications as services (e.g., transactional support, accessing database, communication support, load balance support, resource management, monitoring etc). Once applications are deployed into the containers of application servers, they can access the services provided by application servers easily. Application developers only need to concern about the application logic. In this way, the complexity of application development is reduced. The general architecture of an application server is illustrated in Fig. 7.

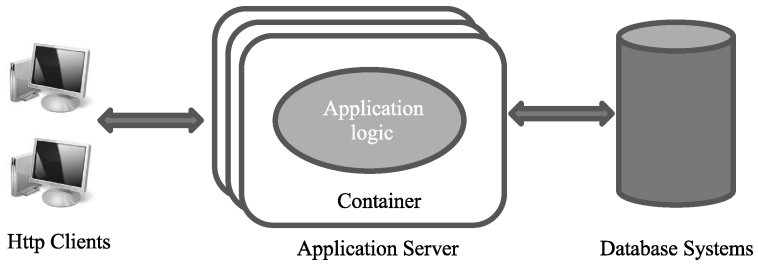

Figure 7. Application server

In industry, many application servers have been developed to facilitate the development, deployment and maintenance of Internet applications. Some well-known application servers include IBM WebSphere[35], BEA WebLogic[28], Microsoft .NET platform[51] etc. These application servers share some common characteristics. For example, they are integrated with all kinds of reusable services that can be easily accessed and consumed by high-level applications. They also provided integrated development, deployment, execution, monitoring and maintenance environment for application developers to manage and reduce the complexity in the whole lifecycle of application development and maintenance.

Current research efforts on application servers are mainly focused on how to improve the performance and scalability of application servers[43,49,73]. This includes how to improve the throughput of transaction processing[29], how to conduct load balance in a better way[46,65], how to customize the configuration of services to achieve better performance[39,79,88], to just name a few.

3.5 Mobile agent systems

Mobile agent systems are developed to support and manage mobile agents migrating from one host to another. Mobile agents are executing programs that can be executed in different agent hosts. Mobile agents can choose to migrate between hosts during their execution based on the requirement (e.g., resource, bandwidth etc). Therefore, mobile agents can be used to implement distributed business applications[14], Internet applications[12] and database access[63] and so on. The general architecture of mobile agent systems is illustrated in Fig. 8.

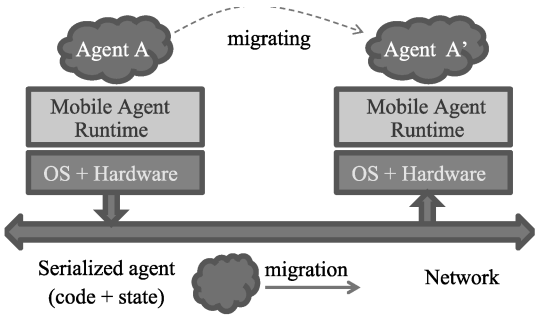

Figure 8. Mobile agent system

Many research efforts have been devoted to designing and developing mobile agent systems. Well-known mobile agent systems include Agent TCL[31], ARA[60],

Concordia[76], JADE[8] etc. The major research issues in mobile agent systems include security[14], fault-tolerance[61], locating agents[18], routing[76] etc.

3.6 Cloud computing platform

A general architecture of clouding computing platform is illustrated in Fig. 9. In the architecture, computation facilities (hardware + servers) are connected to form a computation cloud. Each node in the cloud can be a computation platform to host applications, or a data center to store and share data. Middleware systems can be built on top of the infrastructure to provide value-added services for high-level applications.

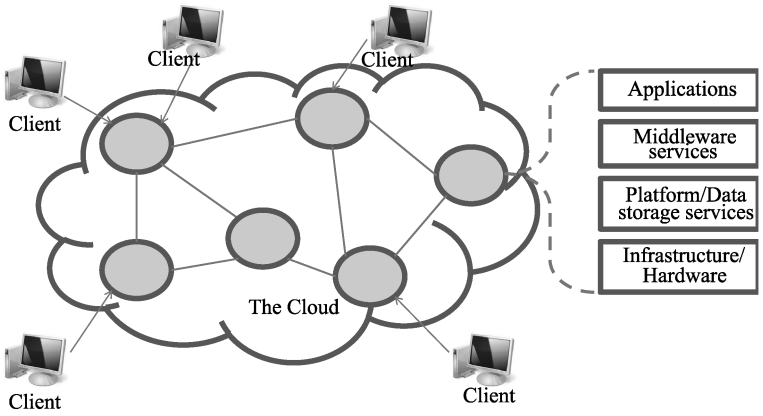

Figure 9. Cloud computing platform

To support applications in cloud computing, quite a few cloud computing platforms have been developed. For example, recent well-known industry cloud computing platforms include Google App Engine[30], Amazon Web services[2], Microsoft Azure services platform[50] etc. From a functional perspective, these platforms are similar to the role of application servers. They integrate data center and reusable solutions to common issues as services in the platform. Application developers only need to concern about the application logic. The difference is that these platforms serve as public servers to host applications from different organizations.

Current research efforts on clouding computing are focused on data centers (virtualization, scalable storage etc)[51,66], performance[47,52], security and privacy[62,78], resource provision[38,37] etc.

4 Remaining Issues

In Section 2, we analyze the requirements and challenges for supporting an e-commerce application in cloud computing. In Section 3, we review the state-of-the-art middleware solutions. In this section, we investigate how these requirements and challenges are addressed by the state-of-the-art solutions, and what are remaining.

As explained in Section 3, many requirements can be supported by some particular middleware systems. For example, the context-awareness can be well supported by context-aware middleware. Reliable asynchronous communication among applications can be implemented with the aid of message-oriented middleware systems.

Service-oriented middleware can be used to support flexible and loosely coupled service integration. Mobile agent systems can support the migration of an agent from one host to another. Existing cloud computing platforms provide public application servers to host applications for different organizations.

Since so many existing middleware systems have been developed to cover more or less the requirements mentioned in Section 2, we may ask a question: ***are these existing middleware systems adequate for Internetware in the context of cloud computing***? The answer is **No**. This is because middleware systems are regarded as public value-added services shared by high-level applications from different organizations, in the sense that they are no longer under the control of individual autonomous organizations. On the one hand, applications can select or replace the middleware services dynamically based on their changing requirements. On the other hand, middleware systems may also be hosted on some rented computation clouds so that they may be migrated from one host ot another. These new features introduce new issues with great challenges that remain unsolved in existing work and need more research efforts. In particular, the following issues need to be addressed in the middleware systems in could computing:

1. **Coordination**: Since the middleware systems are public services shared by different applications, and different applications may select different middleware services, flexible mechanisms to coordinate the collaboration among applications and middleware services are needed. For example, some online shopping services may require the participants to be the members of particular organizations, whereas some other shopping services may accept anonymous customers and allow them to join or leave at any time during the collaboration. These services may also use different middleware services with different communication patterns (e.g., synchronous vs. asynchronous, one-to-one vs, many-to-many etc). When these applications collaborate with each other (e.g., a shopping application may acquire the social relationships from another application like Facebook, or these applications exchange the context information), their underlying middleware systems need get involved as well.

   Traditional service-oriented middleware systems however are inadequate to handle above scenario. This is because traditional collaboration among services is coordinated only at the application level on the one hand, and such collaboration is usually restricted to a fixed number of participants with predefined collaboration protocols on the other hand. In an open environment, the participants (including applications and underlying middleware systems) are unknown in advance. The number of involved participants may also change from time to time. Moreover, the participating services may also evolve from time to time. As a result, how to coordinate the collaboration among high-level applications as well as the middleware services flexibly and correctly in an open environment is still a challenging research issue.

2. **Interoperability**: Another remaining issue is the interoperability among middleware services. Since middleware systems are regarded as public value-added services for applications, one middleware system may also integrate another middleware service to provide better services for high-level applications. For

example, a context-aware middleware service may integrate a third-party security middleware service to address the security concerns; or a reliable asynchronous communication middleware service incorporates a transactional middleware service to address the failure during the message delivery. Existing application servers or cloud computing platforms address the issue by integrating well-known solutions together into the platforms and mandate applications to use these integrated services. However, in more general scenarios, applications should be able to select and customize the middleware services based on the requirements and their contexts. This requires the middleware services should be open to integrate other middleware services and be integrated by others.

Besides the service level interoperability, another issue is about the interoperability at the semantics level. Different middleware services may use different data structure and data representation. For example, different context-aware middleware services may use different context models (e.g., tuple space or first-order logic). When two applications running on two different middleware services need to exchange their context information, each underlying middleware system should be able to understand the meaning of contexts represented in the other system. Although such issue can be easily solved by introducing transformation between different systems, the issue still remains when participants are dynamic and unknown in advance.

There are also some other issues related to the interoperability that need to be addressed. For example, applications may migrate from one middleware service to another (e.g., to select a better service), and some minor difference in the implementation of middleware services may cause the application to behave differently[84], how to ensure that two middleware services are replaceable? In addition, when a middleware service integrates other middleware services, how to assign the computation resources among all the involved services to guarantee the quality of service for high-level applications? When one middleware service evolves or re-configures the resource usage policy, how to analyze the impacts and prevent the impacts from being propagated to other involved middleware services?

3. **Engineering methodology**: As explained in Section 1, different from traditional application model, the middleware systems in cloud computing serve as public services for high-level applications. This increases the complexity for the development of high-level applications. The difficulties lie in how to effectively test and debug the high-level applications[1,26]. On the one hand, the middleware systems are beyond the control of the organizations that the high-level applications belong to. Due to privacy and security concerns, applications developers may not be able to efficiently test all the application scenarios, nor be allowed to track the errors inside the middleware services. On the other hand, a middleware service may integrate other middleware services that are transparent to application developers. This complicates the analysis and tracking of root cause for the errors inside an application. Therefore, new software engineering theories and methodologies are needed to manage and reduce the complexity for developing high-level applications running on top of public middleware services.

These remaining issues are still not well addressed in the existing middleware solutions and need more future research efforts. In the next section, we discuss some potential future directions to address these issues.

5 Future Work

In this paper, we view middleware systems in cloud computing from a service perspective. We analyze the requirements and challenges for middleware support for Internetware in the context of could computing. We also review the state-of-the-art solutions to address these requirements and challenges in existing work. We found that although many existing middleware systems have been proposed and developed, they are still not enough to address the requirements needed for middleware services. To address the remaining issues, more research efforts are needed.

One possible way to address the coordination issue is to extend the content-based publish/subscribe middleware to support more flexible service collaboration. Publish/subscribe middleware decouples message senders and receivers and supports a many-to-many message delivery pattern. This feature is useful for flexible service coordination wherein the number of services is not fixed and services can join or leave from time to time. However, a publish/subscribe middleware in such scenarios should not be limited to delivering messages only. Services can be advertised to others through the publish/subscribe middleware. Services can also subscribe to other services based on their subscriptions. This requires the middleware should be able to route services based on the matching of service behavior against the service subscriptions. In addition, another extension is to support flexible scope-based message routing (e.g., only the participants with authority can send or receive messages due to privacy or security concerns, or a service is advertised to only the friends of the publisher based on the social network information).

Another possible direction is to enrich both the middleware services and high-level application services with more informative interfaces. These additional service interfaces can be used to check the interoperability issue among different services. For example, a middleware service may provide a resource interface in addition to its traditional service interface. The resource interface can be used to describe the resource usage policies and strategies inside the middleware service. Based on the resource interface, we can verify whether two collaborating middleware services are compatible in terms of resource consumption policy. Other possible interfaces may include context interface to exchange contexts, testing and debugging interfaces to test and debug services, and so on.

We also plan to explore and expose additional internal state changes inside services (middleware services as well as high-level applications) as events without revealing the implementation details of services, and make use of the additional exposed information to help develop reliable and dependable services. For example, we plan to develop a methodology to expose certain internal state changes inside services during their execution as events[85]. Such internal state changes may reveal some useful information about the errors inside a service. These events may be useful in testing and debugging service compositions running on public middleware services in cloud computing.

References

[1] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M. A View of Cloud Computing. Commun. ACM., 2010, 53(4): 50–58.

[2] Amazon. Amazon Web Services. http://aws.amazon.com/.

[3] Apache. Apache ActiveMQ. http://activemq.apache.org/.

[4] Apache. Apache Orchestration Director Engine. http://ode.apache.org/

[5] Boldrini C, Conti M, Delmastro F. Context- and Social-aware Middleware for Opportunistic Networks. J. Netw. Comput. Appl, 2010, 33(5): 525–541.

[6] Bellavista P, Corradi A, Montanari R, Stefanelli C. Context-Aware Middleware for Resource Management in the Wireless Internet. IEEE Trans. Softw. Eng, 2003, 29(12): 1086–1099.

[7] Bellavista P, Corradi A, Montanari R, Stefanelli C. A Mobile Computing Middleware for Location- and Context-aware Internet Data Services. ACM Trans. Internet Technol. 2006, 6(4): 356–380.

[8] Bellifemine F, Caire G, Poggi A, Rimassa G. JADE: A White Paper. EXP in Search of Innovation, 2003, 3(3): 6–19.

[9] Chan A T S, Chuang S N. MobiPADS: A Reflective Middleware for Context-Aware Mobile Computing. IEEE Trans. Softw. Eng., 2003, 29(12): 1072–1085.

[10] Chuang S N, Chan ATS. Dynamic QoS Adaptation for Mobile Middleware. IEEE Trans. Softw. Eng., 2008, 34(6): 738–752.

[11] Capra L, Emmerich W, Mascolo C. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. IEEE Trans. Softw. Eng., 2003, 29(10): 929–945.

[12] Cabri G, Leonardi L, Zambonelli F. Engineering Mobile Agent Applications via Context-Dependent Coordination. IEEE Trans. Softw. Eng., November 2002, 28: 1039–1055, .

[13] Comuzzi M, Pernici B. A Framework for QoS-based Web Service Contracting. ACM Trans. Web, 2009, 3(3): 1–52.

[14] Claessens J, Preneel B, Vandewalle J. How can Mobile Agents Do Secure Electronic Transactions on Untrusted Hosts? A Survey of the Security Issues and the Current Solutions. ACM Trans. Internet Technol., February 2003, 3: 28–48.

[15] Roman G, Julien C, Payton J. A Formal Treatment of Context-awareness. FASE'04. Springer. 2004. 12–36.

[16] Carzaniga A, Rosenblum DS, Wolf AL. Design and Evaluation of A Wide-area Event Notification Service. ACM Trans. Comput. Syst., 2001, 19(3): 332–383.

[17] Anind K. Dey and Gregory D. Abowd and Daniel Salber. A Context-Based Infrastructure for Smart Environments, 1999.

[18] Di Stefano A, Santoro C. Locating Mobile Agents in a Wide Distributed Environment. IEEE Trans. Parallel Distrib. Syst., August 2002, 13: 844–864.

[19] Emmerich W, Aoyama M, Sventek J. The Impact of Research on the Development of Middleware Technology. ACM Trans. Softw. Eng. Methodol., 2008, 17(4): 1–48.

[20] Eugster PT, Felber PA, Guerraoui R, Kermarrec AM. The Many Faces of Publish/subscribe. ACM Comput. Surv., 2003 , 35(2): 114–131.

[21] Emmerich W. Software Engineering and Middleware: a Roadmap. ICSE'00. 2000. 117–129.

[22] Erradi A, Maheshwari P, Tosic V. Policy-driven Middleware for Self-adaptation of Web Services Compositions. Middleware'06. Springer-Verlag New York, Inc. 2006. 62–80.

[23] Fabret F, Jacobsen HA, Llirbat F, Pereira J, Ross KA, Shasha D. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. SIGMOD'01. 2001. 115–126.

[24] Fujii K, Suda T. Semantics-based Context-aware Dynamic Service Composition. ACM Trans. Auton. Adapt. Syst., 2009, 4(2): 1–31.

[25] Foster H, Uchitel S, Magee J, Kramer J. Model-based Verification of Web Service Compositions. ASE '03. 2003. 152–161.

[26] Gu L, Cheung SC. Constructing and Testing Privacy-aware Services in a Cloud Computing Environment: Challenges and Opportunities. Internetware'09. 2009, 2: 1–2:10.

[27] Grimm R, Davis J, Lemar E, Macbeth A, Swanson S, Anderson T, Bershad B, Borriello G, Gribble S, Wetherall D. System Support for Pervasive Applications. ACM Trans. Comput. Syst., 2004, 22(4): 421–486.

[28] Girdley M, Emerson SL, Woollen R. J2EE Applications and BEA WebLogic Servers. Prentice Hall PTR. 2001.
[29] Gallagher B, Jacobs D, Langen A. A High-performance, Transactional Filestore for Application Servers. SIGMOD'05. 2005. 868–872.
[30] Google. Google APP Engine. http://code.google.com/intl/en/appengine/.
[31] Gray RS. Agent Tcl: a Flexible and Secure Mobile-agent System. TCLTK'96. USENIX Association. 1996. 2–2.
[32] Gilman L, Schreiber R. Distributed Computing with IBM MQSeries. John Wiley & Sons, Inc., 1996.
[33] Hapner M, Burridge R, sharma R. Java Message Service Specification. Technical report, Sun Microsystems, 1999. http://java.sun.com/products/jms.
[34] Halima RB, Drira K, Jmaiel M. A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services. ICWS'08. 2008. 104–111.
[35] Herness EN, High RH, McGee JR. WebSphere Application Server: a Foundation for on Demand Computing. IBM Syst. J, 2004, 43(2): 213–237.
[36] Han Q, Venkatasubramanian N. Timeliness-Accuracy Balanced Collection of Dynamic Context Data. IEEE Trans. Parallel Distrib. Syst., 2007, 18(2): 158–171.
[37] Hu Y, Wong J, Iszlai G, Litoiu M. Resource Provisioning for Cloud Computing. CASCON'09. 2009. 101–111.
[38] Hassan M, Zhao W, Yang J. Provisioning Web Services from Resource Constrained Mobile Devices. CLOUD'10. 2010. 490–497.
[39] Jung G, Joshi K R, Hiltunen MA, Schlichting RD, Pu C. A Cost-sensitive Adaptation Engine for Server Consolidation of Multitier Applications. Middleware'09. Springer-Verlag, 2009. 163–183.
[40] Jin J, Nahrstedt K. QoS-Aware Service Management for Component-based Distributed Applications. ACM Trans. Internet Technol, 2008, 8(3): 1–31.
[41] Julien C, Roman GC. EgoSpaces: Facilitating Rapid Development of Context-Aware Mobile Applications. IEEE Trans. Softw. Eng, 2006, 32(5): 281–298.
[42] Kazemzadeh RS, Jacobsen HA. Reliable and Highly Available Distributed Publish/Subscribe Service. SRDS'09. 2009. 41–50.
[43] Kistijantoro AI, Morgan G, Shrivastava SK, Little MC. Enhancing an Application Server to Support Available Components. IEEE Trans. Softw. Eng., July 2008, 34: 531–545.
[44] Laumay P, Bruneton E, Palma ND, Krakowiak S. Preserving Causality in a Scalable Message-Oriented Middleware. Middleware'01. Springer-Verlag. 2001. 311–328.
[45] Li G, Muthusamy V, Jacobsen HA. A Distributed Service-oriented Architecture for Business Process Execution. ACM Trans. Web, 2010, 4(1): 1–33.
[46] Lodi G, Panzieri F, Rossi D, Turrini E. SLA-Driven Clustering of QoS-Aware Application Servers. IEEE Trans. Softw. Eng., March 2007, 33: 186–197.
[47] Liu H, Wee S. Web Server Farm in the Cloud: Performance Evaluation and Dynamic Architecture. CloudCom. 2009 369-380.
[48] Meier R, Cahill V. On Event-Based Middleware for Location-Aware Mobile Applications. IEEE Trans. Softw. Eng, 2010, 36(3): 409–430.
[49] Munch-Ellingsen A, Eriksen DP, Andersen A. Argos, an Extensible Personal Application Server. MIDDLEWARE'07. Springer-Verlag, Berlin, Heidelberg, 2007. 21–40.
[50] Microsoft. Microsoft Azure platform. http://www.microsoft.com/windowsazure/.
[51] Microsoft. .Net platform. http://www.microsoft.com/net/.
[52] Mei Y, Liu L, Pu X, Sivathanu S. Performance Measurements and Analysis of Network I/O Applications in Virtualized Cloud. CLOUD'10. 2010. 59–66,
[53] Malek S, Mikic-Rakic M, Medvidovic N. A Style-aware Architectural Middleware for Resource-constrained, Distributed Systems. Software Engineering. IEEE Trans. on, March 2005, 31(3): 256–272.
[54] Murphy AL, Picco GP, Roman GC. LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents. ACM Trans. Softw. Eng. Methodol. 2006, 15(3): 279–328.
[55] Maheshwari P, Tang H, Liang R. Enhancing Web Services with Message-Oriented Middleware. ICWS'04. 2004. 524–.

[56] Mamei M, Zambonelli F. Programming Pervasive and Mobile Computing Applications: The TOTA Approach. ACM Trans. Softw. Eng. Methodol., 2009, 18(4): 1–56.
[57] Naghshineh M, Ratnaparkhi R, Dillenberger D, Doran J R, Dorai C, Anderson L, Pacifici G, Snowdon J L, Azagury A, VanderWiele M, Wolfsthal Y. IBM Research Division Cloud Computing Initiative. IBM J. Res. Dev., July 2009, 53: 499–508.
[58] Pietzuch P R, Bhola S. Congestion Control in a Reliable Scalable Message-oriented Middleware. Middleware'03. Springer-Verlag, New York, Inc. 2003. 202–221.
[59] Perera S, Herath C, Ekanayake J, Chinthaka E, Ranabahu A, Jayasinghe D, Weerawarana S, Daniels G. Axis2, Middleware for Next Generation Web Services. ICWS'06. IEEE Computer Society. 2006. 833–840.
[60] Peine H, Stolpmann T. The Architecture of the Ara Platform for Mobile Agents. MA'97. Springer-Verlag, 1997. 50–61.
[61] Pleisch S, Schiper A. Fault-Tolerant Mobile Agent Execution. IEEE Trans. Comput. February 2003. 52: 209–222.
[62] Pearson S, Shen Y, Mowbray M. A Privacy Manager for Cloud Computing. CloudCom'09. Springer-Verlag, 2009. 90–106.
[63] Papastavrou S, Samaras G, Pitoura E. Mobile Agents for World Wide Web Distributed Database Access. IEEE Trans. on Knowl. and Data Eng., September 2000, 12: 802–820.
[64] Michael PP, Paolo T, Istituto R, Scientifica T. Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer, 2007, 40: 2007.
[65] Quaglia F, Romano P. Ensuring e-Transaction with Asynchronous and Uncoordinated Application Server Replicas. IEEE Trans. Parallel Distrib. Syst, March 2007, 18: 364–378.
[66] Ramakrishnan R. Data Management in the Cloud. ICDE'09. 2009. 5–.
[67] Binoy R. Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language, and Middleware. IEEE Trans. Softw. Eng., 2002, 28(1): 30–57,
[68] Ranganathan A, Campbell RH. A Middleware for Context-aware Agents in Ubiquitous Computing Environments. Middleware'03. Springer-Verlag New York, Inc., 2003. 143–161.
[69] Shannon B. Java 2 Platform Enterprise Edition Specification. Technical report, Sun Microsystems. 2003. http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf.
[70] Sum J, Shen H, Leung C, Young G. Analysis on a Mobile Agent-Based Algorithm for Network Routing and Management. IEEE Trans. Parallel Distrib. Syst., March 2003, 14: 193–202.
[71] Talia D, Yahyapour R, Ziegler W. Grid Middleware and Services: Challenges and Solutions. Springer Publishing Company. Incorporated. 2008.
[72] van Steen M, Tanenbaum AS, Kuz I, Sips HJ. A Scalable Middleware Solution for Advanced Wide-area Web Services. Middleware'98. Springer-Verlag, 1998. 37–53.
[73] Willenborg R, Brown K, Cuomo G. Designing WebSphere Application Server for Performance: An Evolutionary Approach. IBM Syst. J., April 2004. 43: 327–350.
[74] Wong EYC, Chan ATS, Leong HV. Xstream: A Middleware for Streaming XML Contents over Wireless Environments. IEEE Trans. Softw. Eng., 2004, 30(12): 918–935.
[75] Wohlstadter E, Li P, Cannon B. Web Service Mashup Middleware with Partitioning of XML Pipelines. ICWS'09. 2009. 91–98.
[76] Wong D, Paciorek N, Walsh T, DiCelie J, Young M, Peet B. Concordia: An Infrastructure for Collaborating Mobile Agents. MA'97. Springer-Verlag, 1997. 86–97.
[77] Wohlstadter E, Tai S, Mikalsen T, Diament J, Rouvellou I. A Service-oriented Middleware for Runtime Web Services Interoperability. ICWS'06. 2006. 393–400.
[78] Wang C, Wang Q, Ren K, Lou WJ. Privacy-preserving Public Auditing for Data Storage Security in Cloud Computing. INFOCOM'10. 2010. 525–533.
[79] Xi BW, Liu Z, Raghavachari M, Xia CH, Zhang L. A Smart Hill-climbing Algorithm for Application Server Configuration. WWW'04. 2004. 287–296.
[80] Ye CY, Cheung SC, Chan WK. Publishing and Composition of Atomicity-equivalent Services for B2B Collaboration. ICSE'06. 2006. 351–360.
[81] Ye CY, Cheung SC, Chan WK, Xu C. Local Analysis of Atomicity Sphere for B2B Collaboration. SIGSOFT'06/FSE-14. 2006. 186–196.
[82] Ye CY, Cheung SC, Chan WK, Xu C. Detection and Resolution of Atomicity Violation in

Service Composition. ESEC/FSE'07. 2007. 235–244.
[83] Ye CY, Cheung SC, Chan WK, Xu C. Atomicity Analysis of Service Composition across Organizations. IEEE Trans. Softw. Eng, 2009, 35(1): 2–28.
[84] Ye CY, Cheung SC, Wei J, Zhong H, Huang T. A Study on the Replaceability of Context-aware Middleware. Internetware'09. 2009. 1–10.
[85] Ye C Y, Jacobsen H. Event Exposure for Web Services: A Grey-Box Approach to Compose and Evolve Web Services. The Smart Internet, LNCS6400, Springer, 2010. 197–215.
[86] Ye XF, Shen YL. A Middleware for Replicated Web Services. ICWS'05. 2005. 631–638.
[87] Yau SS, Wang Y, Karim F. Development of Situation-Aware Application Software for Ubiquitous Computing Environment. COMPSAC'02. 2002. 233–238.
[88] You C, Zhou MH, Xiao Z, Mei H. Towards a Dynamic and Adaptable Application Server. Internetware'09. 2009. 13:1–13:5.
[89] Zeng LZ, Benatallah B, H.H.Ngu A, Dumas M, Kalagnanam J, Chang H. QoS-Aware Middleware for Web Services Composition. IEEE Trans. Softw. Eng., 2004, 30(5): 311–327.
[90] Zheng YJ, Chan ATS. MobiGATE: A Mobile Computing Middleware for the Active Deployment of Transport Services. IEEE Trans. Softw. Eng., 2006, 32(1): 35–50.
[91] Zulkernine F, Martin P, Craddock C, Wilson K. A Policy-Based Middleware for Web Services SLA Negotiation. ICWS'09. 2009. 1043–1050.