Science in China Series F: Information Sciences

www.scichina.com
info.scichina.com
www.springerlink.com

Technical framework for Internetware: An architecture centric approach

YANG FuQing[1,2†], LÜ Jian[3,4] & MEI Hong[1,2]

[1] Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China;
[2] School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China;
[3] State Key Laboratory for Novel Software Technology, Nanjing 210093, China;
[4] Department of Computer Science and Technology, Nanjing University, Nanjing 210093, China

Being a new software paradigm evolved by the Internet, Internetware brings many challenges to the traditional software methods and techniques. Sponsored by the national basic research program (973), researchers in China have developed an architecture centric technical framework for the definition, incarnation and engineering of Internetware. First of all, a software model for Internetware is defined for what to be, including that Internetware entities should be packaged as components, behaving as agents, interoperating as services, collaborating in a structured and on demand manner, etc. Secondly, a middleware for Internetware is designed and implemented for how to be, including that Internetware entities are incarnated by runtime containers, structured collaborations are enabled by runtime software architecture, Internetware can be managed in a reflective and autonomic manner, etc. Thirdly, an engineering methodology for Internetware is proposed for how to do, including the way to develop Internetware entities and their collaborations by transforming and refining a set of software architectures which cover all the phases of software lifecycle, the way to identify and organize the disordered software assets by domain modeling, etc.

Internetware, component, software architecture, agent, middleware

1 Introduction

Software is essentially a computer program modeling the problem space of the real world and its solution. Software can perform various tasks including controlling hardware devices, computing, communicating, and so on. It always pursues a computing model that is both expressive and natural. A basic software model comprises entity elements and the interactions among them. The model has evolved from the initial machine language instruction and the sequence and jump rela-

Received December 24, 2007; accepted March 6, 2008
doi: 10.1007/s11432-008-0051-z
†Corresponding author (email: yang@pku.edu.cn)
Supported by the National Key Basic Research and Development Program of China (973) (Grant No. 2002CB312000)

tionship, to high-level language statement and the three control structures, procedures and the sub-procedures relationship, object and message passing, to the currently popular model of components and connectors. The evolution of computing model greatly facilitates software construction and maintenance. On the other hand, software also pursues better utilization of hardware capabilities. Hardware devices are controlled by software, without which it is impossible for them to work efficiently and flexibly. For instance, operating systems are developed from the earliest boot programs for starting the computers, the simple device management programs, and the multi-channel programs for efficiently utilizing the CPU and I/O, to resource management systems for efficiently utilizing both software and hardware. In that sense, the evolution of software technology and systems is mainly driven by the application domains and underlying hardware.

In the 21st century, Internet promotes the globalization widely and deeply and then brings new opportunities and challenges to the software applications. On the other hand, Internet is now growing up into a "ubiquitous computer" which consists of a great and growing number of computing devices. Internet provides much more powerful supports for problem solving than traditional computer systems. In order to cope with such evolutionary changes of application domains and underlying hardware, software systems need to be online evolvable, continually responsive and self-adaptive. Based on the object-oriented methods, software components and other techniques, software entities are distributed on the nodes of Internet as active and autonomic software services. These entities can collaborate with each other in various manners and thus form a software Web, which is similar to the World Wide Web, an information web. Such a new software paradigm evolved by Internet is called Internetware in this paper[1].

Generally speaking, Internetware is constructed by a set of autonomic software entities distributed over the Internet, and a set of connectors enabling the collaboration among these entities in various manners. The software entities are capable of perceiving the dynamic changes of its environment, and adapting to these changes through architectural evolution, including the adding, deleting and evolving of software entities and connectors, as well as the corresponding changes of the system topology. Through such context aware behaviors, the system can not only satisfy user requirements, but also improve their experiences[2]. The form of Internetware is very different from that of the traditional software. From the micro perspective, software entities collaborate with each other on demand, and from the macro perspective, the entities can organize themselves into an application domain. Accordingly, the development of Internetware can be seen as the composition of various "disordered" resources into "ordered" software systems. As the time elapses, changes of resources and environments may "disorder" the existing software systems again, which will become "ordered" by composition sooner or later. The iterative transformation between "ordered" and "disordered" entities implies a bottom-up, inside-out and spiral development process.

Since the traditional software engineering methods and techniques are originated from and more suited for a relatively static and closed environment, they are not appropriate for open, dynamic, ever-changing and decentralized Internetware. Basically, Internetware is still a natural evolution of traditional software over the Internet, and thus the characteristics, concepts, connotation and key techniques for Internetware can be seen as a general enough abstraction and specification of the software technology and methodology in the coming future. Aiming at Chinese software industry which should support modern service industries, researchers from Chinese universities and institutes (including Peking University, Nanjing University, Tsinghua University,

Institute of Software of the Chinese Academy Sciences, the Academy of Mathematics and Systems Science of Chinese Academy Sciences, East China Normal University, Southeast University, Dalian University of Technology, Shanghai Jiao Tong University, etc.) proposed the project "Research on Theory and Methodology of Agent-based Middleware on Internet Platform" in 2002, sponsored by the National Basic Research Program (973). Currently, the research and practice outputs can be concluded as an architecture centric technical framework for Internetware, including the following three major aspects: For Internetware entity model, we separate an entity from the environments and other entities, and introduce decision-making capabilities into an entity to make it more autonomous. For the collaboration between Internetware entities, we address the shortcomings of object-oriented methods such as the fixed message receiver, synchronized interaction and single implementation by the architecture-based programming for explicit, flexible and loose-coupled collaborations. For Internetware operating platform (i.e. the middleware), we use containers and runtime software architectures (RSA) to incarnate the Internetware entities and on-demand collaborations respectively, and use a series of key techniques, such as the componentized platform structure, completely reflective framework and autonomic management loop to achieve the self-organization and self-adaptation of Internetware. For Internetware development methodology, we propose the "software architecture across the whole lifecycle" to cope with the phenomenon that the main body of the development shifts from pre-software-release to post-software-release. We use an architecture centric composition method to support the development of Internetware entities and on-demand collaborations, and also a domain modeling method for organizing disordered resources into ordered ones.

2 Software model of Internetware

Software model is the core of software methodology, i.e. how a special methodology, along with its supporting techniques, abstracts the elements and behaviors of a target software system. Being an abstraction of software systems running in the open, dynamic and decentralized Internet, Internetware differentiates from classical software systems in constitution, operation, correctness, development, trustworthiness, lifecycle, etc.

- For the constitution, Internetware is open and reflective. Openness means that Internetware is constituted by a set of widely distributed, third parties provided, autonomous and service like software entities, which form a virtual organization by diverse connectors for the communication and interoperability. Reflection means that Internetware has two levels internally: the objective level and meta level. The objective level covers the scope of traditional software, as well as an explicit abstraction of the environment. And the meta level covers the context-aware mechanisms for external environment changes and evolution mechanisms for internal system behaviors.
- For the operation, Internetware can actively adapt to the context. The operation of the objective-level system is mainly to satisfy user requirements in the current period, and the operation of the meta-level system is to dynamically adapt the running system according to the environment changes so that the objective-level system can better satisfy user requirements in the next period. In that sense, the system in operation is evolving continuously. Furthermore, the software lifecycle of Internetware manifests as a dynamic model with emphasis on evolution.
- For the correctness, through the continuous evolution, Internetware seeks for variable and flexible user experiences instead of relatively strict and static functionalities.

- For the development, developing Internetware is a process of looking for the best service and the value-added services. During this process, the incremental development approach supported by the autonomous software entities and on-demand collaborations essentially contains the principle of software reuse based on component composition.
- For the trustworthiness, besides the classical techniques for security and dependability, Internetware emphasizes a flexible measurement, deduction and application of trustworthiness, based on the historical information.

Emerging from the static, closed and controllable environment, the object-oriented (OO) software model has some inborn characteristics such as poor autonomy, fixed encapsulation, simple interaction, tight coupling, offline evolution, etc. Although there are some progresses on such areas as distributed object computing and aspect oriented programming, these inborn characteristics of OO model have not been changed fundamentally, and therefore it still cannot support the development of Internetware sufficiently and efficiently. As a result, on the basis of OO model, we propose an agent based, software architecture centric software model, as shown in Figure 1 (technical details can be found in refs. [3—9]).

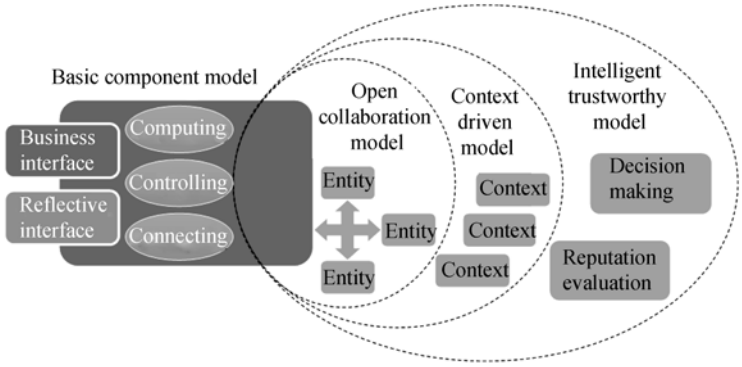

Figure 1 Software model for Internetware.

2.1 Basic component model

First of all, the entity of Internetware is a component so that it can be deployed, executed and evolved independently. Like traditional software components, Internetware entities have the business interfaces to provide the functions of computation, control and connection. But different from traditional software components, Internetware entities have reflective interfaces to provide the capabilities for monitoring, analyzing, planning and adapting the business functions, so that Internetware can be autonomous, evolutionary, cooperative, polymorphic, and context-aware. The heterogeneity of Internet determines that the Internetware entities do not have the same characteristics, and thus the Internetware component model has to be customizable and extensible. For example, software entities providing only business functions but none reflective capabilities can be seen as a basic type of Internetware entities. The software entities with the component meta model and reflective interfaces have the capability of self-awareness. The entities with the environment meta model and reflective interfaces have the capability of context-awareness. The entities with reflective interfaces for altering their states and behaviors have the capability of self-adaptation. The rationale and implementation of these Internetware characteristics are specified by the open collaboration model, context driven model, and intelligent trustworthy model.

2.2 Open collaboration model

This model aims to provide a new interpretation for the core of OO model, i.e. the object-message structure, according to the open Internet. For the structure, the separation between software entities and their collaborations makes the collaborating logic explicit and independent entities based on software architectures. For the development, the software entities and their collaborations can be developed independently by service providers and service integrators, respectively. Service providers can develop the Internetware entities under the support of the mature object-oriented technology, and encapsulate those that use existing techniques like Web Services. Service integrators can design the collaborations using the techniques like the runtime software architectures, multi-mode interactions and collaboration-oriented service compositions. For the adaptation, software entities can be evolved or replaced by newly developed ones, while the collaborations can cope with changes by flexible service composition and layered evolution.

2.3 Context driven model

This model aims to build up a model to capture the features and changing modes of the context of the open collaboration model. These two models can interact with each other and evolve together by reactive service computing. To establish a context driven system, we need the awareness and analysis to the ever-changing interaction mode between the user and the environment of an Internetware system, and then, on the basis of such analysis to derive the context driven behavior mode and also to achieve the context awareness and the self adaptation based on the above framework and self-adaptive software architecture. The basic implementation techniques include agent-based contextual information collection and process, intelligent software architecture evolution, and so on.

2.4 Intelligent trustworthy model

This model aims to solve the problems derived from the open environment, such as the trustworthiness, personalization, self-evolution, etc. by combining the trustworthy computing framework and artificial intelligence on the basis of the context driven model. Based on the classical trustworthy computing techniques, which usually focus on single attribute in a developer-directed and specification-dominant manner, we develop more appropriate ones for Internetware, including taking multiple attributed into account, empirical and flexible assessment, etc. The supporting techniques include trustworthiness description and measurement frameworks specific to Internetware, agent-based trust management, trusted (semi) automatic composition and evolution, and so on.

3 Middleware for Internetware

Middleware is a special kind of distributed software that mediates between the application software and system software, enabling the interactions between distributed application software while hiding the complexity and heterogeneity of the underlying system software. Along with the rapid development and wide application of Internet, middleware technology, products, applications and standards are proliferating, and as a result, the connotation of middleware becomes much wider and can be considered as a new type of system software on Internet. The development of middleware has the following trends: 1) encapsulating more and more capabilities belonging to the tradition distributed operating systems for providing stronger runtime support; 2)

becoming a ubiquitous operating platform for not only the traditional enterprise computing but also many new computing models, such as peer to peer computing, pervasive computing, grid computing and service oriented computing; 3) distilling more and more functions common to a special application domain besides those common to all applications; 4) putting more and more important impacts on the whole software lifecycle, including the operation, deployment, testing, implementation, and even design and analysis.

The proliferation and trends of middleware determine its role in Internetware technical framework, that is, an "operating system" for the deployment, operation and management of Internetware. Basically, the middleware for Internetware should have two capabilities. One is to support the incarnation of Internetware model, i.e. how to make the above-mentioned Internetware entities executable and control their collaborations according to software architectures. The other is to enable the autonomic management of Internetware systems, i.e. how to ensure the running Internetware system to satisfy the expected functionality, performance, reliability, security, etc. in the open, dynamic and decentralized Internet. We establish a model of middleware for Internetware as shown in Figure 2, and the technical details can be found in refs. [10—18].

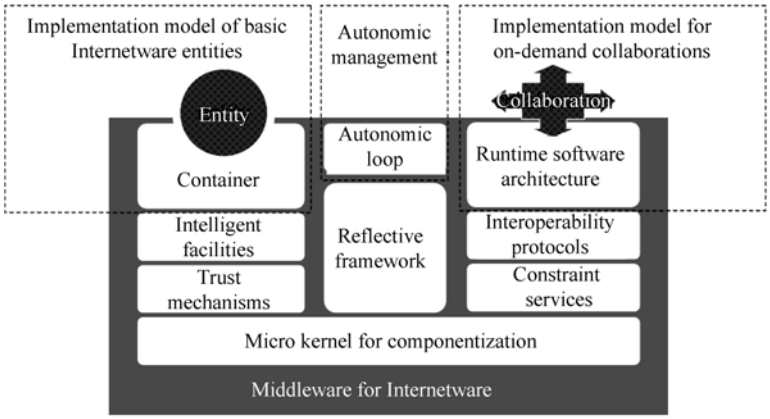

Figure 2 Model of middleware for Internetware.

3.1 The implementation model of basic Internetware entities

The container is the runtime space of an Internetware entity, which is responsible for managing the entity's lifecycle (e.g. the class loading, instantiation, caching, release, etc.), the entity's runtime context (e.g. the invocation context and database connections), and the binding between entity's business functions and quality policies (e.g. interoperability, security, transaction, persistence and rule reasoning). For a normal java class which implements some business functions and is deployed into a container, it can be bound with various quality policies to implement various advanced features, e.g. the interoperability protocols for allowing the invocation through RMI, HTTP or SOAP, the constraint services for the access control, data persistence and transactions, the intelligent facilities for self adjustment of prices based on rules, the trust computing mechanisms for the evaluation of users' reputation, and so on. It should be noted that the implementation of the business function, the quality policies and their bindings could be adjusted online.

3.2 The implementation model of on-demand collaborations

Runtime software architecture (RSA) is a running entity provided by middleware, which has a

causal connection with the running system based on the middleware's in-depth control of the whole system. Such a causal connection ensures that all the changes of RSA can immediately lead to the corresponding changes of the running system, and *vice versa*. Every Internetware entity can be abstracted as an RSA component, and the interactions between entities can be abstracted as RSA connectors. Thus the on-demand collaboration is incarnated by the RSA. From the macro perspective, the collaborations between all Internetware entities form an internet-scale RSA. And from the micro perspective, we can either adjust a part of RSA to explicitly drive the collaboration between entities, or allow the entities to collaborate with each other autonomously, which is still a part of RSA and then under the control of RSA.

There are mainly two ways to establish the causal connection between RSA and running systems: reflective programming model and reflective middleware, each of which has its own advantages and disadvantages. We use reflective middleware as a basis, leveraging the reflective programming model, to achieve a completely reflective framework, which improves the width and depth of the causal connection to support the on-demand collaborations.

3.3 The autonomic management of middleware

The proliferation of middleware and the rich features of Internetware bring two challenges to the management of middleware. On the one hand, as its capabilities become diverse and complex, middleware becomes too complicated to customization, extension and quality control. On the other hand, as middleware is playing a more and more important role in the whole system, the management of middleware determines the users' confidence for the Internetware system, and thus, it is necessary to manage middleware in a systematic perspective. The flexible management objectives and environment changes further increase the complexity of middleware management. Consequently, autonomic management becomes a necessary capability for middleware, that is, middleware can manage itself with less or none human intervention.

Aiming at the three fundamental issues of middleware autonomic management, including the scope, operability and trustworthiness, we establish an architecture centric autonomic management loop for middleware. We model the software architecture of the middleware and its application to define the scope and connotation of middleware management, monitor and control the running system on the basis of RSA, and ensure the correctness and effectiveness through software architecture analysis and evaluation. For the key challenge of autonomic computing, i.e. analyzing and decision making, we first use software architecture and the formal description of decisions to make the knowledge explicit and well formed. Here, we usually use architecture styles and patterns, domain-specific software architectures and application architectures to specify the common sense, domain-specific and application-specific knowledge, respectively. For the adaptation plan, we can either use the imperative descriptions based on dynamic software architecture or the declarative descriptions based on self-adaptive software architecture. That is to say, middleware can automatically derive the adaptation plan using the given knowledge, e.g. use the rules like <event, condition, action> to derive the proper actions under the give event and condition, refactor the partial software architecture by the knowledge derived from the description of bad patterns and the corresponding good patterns, and enhance the whole software architecture by merging a new style to the existing style.

3.4 Componentization of middleware

The capabilities mentioned above require that the middleware has a fine and clear structure. For

this reason, we make the middleware itself component-based by a micro kernel. The implementations for the middleware capabilities, such as containers, protocols, services, facilities, mechanisms and frameworks, are all encapsulated into platform components, and the micro kernel is for the customization, extension and autonomic management of these platform components. The micro kernel is mainly constituted by the following parts: registration and destruction interfaces for the dynamic insertion and removal of platform components, naming and searching interfaces for the selection of given platform components, bus mechanisms for the invocation of platform components through specific meta-programming interfaces, management mechanisms for the lifecycle of platform components, meta-data interfaces and meta-programming mechanisms. It should be noted that, unlike the micro kernels in operating systems, the micro kernel of middleware does not provide functions like memory management, inter process communication, I/O and interruption, for limiting the performance impacts.

4 Engineering approach for Internetware

Due to the open, dynamic, and decentralized Internet, as well as the various user preferences, Internetware keeps evolving after it has been developed and deployed. When an Internetware is published, it is capable of perceiving the dynamic changes of its environment and evolves according to its functionalities and qualities so that it not only satisfies user requirements but also improves user experiences. Besides, the variation of user preferences and return of investment usually lead to a long-lived and ever-changing Internetware. Inevitably, the engineering of such Internetware faces the challenges from the development process, methodology and techniques. For these challenges, we propose a methodology for the engineering of Internetware, as shown in Figure 3, and the technical details can be found in refs. [19—28].

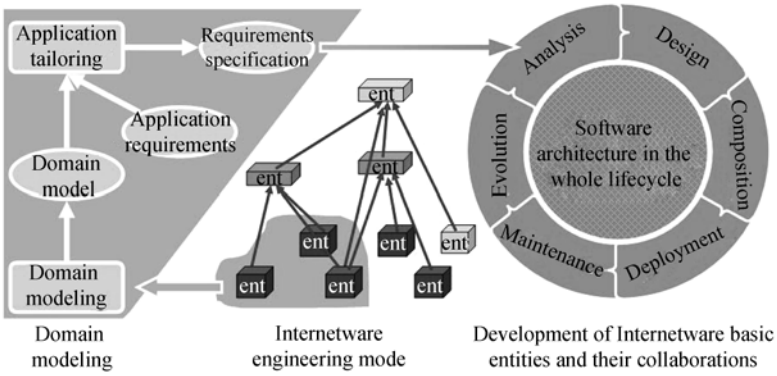

Figure 3 Internetware engineering approach.

4.1 Software architecture in the whole lifecycle

The classical software lifecycle emphasizes the development process from the requirements to the delivery. All efforts and challenges after software delivery are simplified as “software maintenance”. This is appropriate for engineering software systems under a relatively static, closed and controlled environment. However, it is not suitable for Internetware, because 1) the new software entities are usually assembled by the existing and reusable entities. Since all of these entities are relatively independent and there is no central control for them, it is hard to ensure that

the assembled entities satisfy the required functionalities and qualities unless it starts to run; 2) the open, dynamic, and ever-changing environment requires Internetware entities and their collaborations to face with many changes. No matter whether these changes can be predicted or not exactly, the running system has to adapt itself continuously. As a result, software maintenance would become a relatively important phase; 3) Internetware provides services for the worldwide users. Furthermore, an Internetware usually consists of software entities that are distributed over the Internet, and then has no chance to be shut down completely after it is deployed and starts to run. This implies that all maintenance activities, including debugging, optimizing and upgrading, have to perform online. All these activities also have the phases of analysis, design, implementation, testing and deployment, which cannot be controlled well by the concepts and techniques in the classical software maintenance. For this reason, we propose the concept of software architecture in the whole lifecycle; that is, software architecture acts as a blueprint and plays a central role in every stage of Internetware lifecycle.

- Analysis (using concept view of software architecture). To narrow the gap between the requirements and design, we should use a structural method in the stage of requirements analysis to organize the problem domain and user requirements. We do not restrict a specific form for the artifact in this stage, so long as the artifact can be transformed into software architecture conveniently, naturally, and directly. For example, the analysis result from object oriented analysis, like class diagrams for a problem domain, can be directly viewed as a conceptual software architecture; the result from feature oriented requirement analysis, the feature model, can be used to generate the conceptual software architecture semi-automatically through a way similar to functional decomposition. There are also some architecture oriented requirement analysis approaches that can be used directly to obtain the conceptual software architecture.
- Design (using design view of software architecture). From the requirements specification of a software system, designers can make overall design decisions, refine the components and connectors of the conceptual software architecture and finally construct the static and dynamic software architecture models, including type diagrams, instance diagrams and process diagrams. During this process, designers or the development tools can maintain the trace ability between the requirements specification and design model. It is worth noting that the philosophy of Internetware development is to reuse existing assets whenever possible. As a result, during this stage, designers should pay attention on the reusable assets, such as components, connectors, constraints, aspects, styles, patterns, etc.
- Composition (using implementation view of software architecture). This stage corresponds to the implementation stage in the traditional software development. But unlike the implementation of traditional software architecture, which is achieved through programming according to the design, the basic functional units of Internetware are the existing and running Internetware entities, and thus the focus of Internetware development is not programming but composition. Here the composition means selecting the entities adapted to the software architecture and making all these entities collaborate according the specification in the software architecture. When the entities or collaborations do not fit the software architecture, developers need to do adaptation, and if such adaptation failed, they need to develop new entities to completely implement the target system.
- Deployment (using deployment view of software architecture). Internetware is usually running on a specific middleware platform and it should be deployed into the platform. Deploy-

ment involves a great deal of various data items that usually require deployers to configure manually. But actually, most information involved in the deployment already exists during the design and composition stages, and it can be used for the deployment after some kinds of transformation. On the other hand, the introduction of new entities or new interactions may require changing the organization of existing entities, and how to implement such organization is also the main task of the deployment. For these reasons, we define the deployment view of software architecture, which contains a lot of information inherited from the design and implementation views and also provides a visualized way for deployers to append extra information. This deployment view can also visualize the resource and load of the target environment, and support online changing the organization of running entities.

- Maintenance and evolution (using runtime view of software architecture). In a sense, Internetware development can be viewed as a continuous and iterative refinement, mapping and transformation of software architectures of the target system. After each refinement and transformation, the syntax and semantics of software architecture become more accurate and complete. In the stage of maintenance and evolution, the runtime view reflects the actual runtime states of the system, and thus provides the most accurate and complete information of the system. This view is the runtime software architecture, which also supports the online maintenance and evolution without stopping the whole running system.

4.2 Development of Internetware basic entities and their collaborations

The two main tasks of Internetware development are the development of Internetware entities and the development of on-demand collaborations. Internetware entities are essentially the traditional stand-alone or intranet software systems, with extra characteristics like autonomous, evolutionary, cooperative, polymorphic, and context-aware. Therefore, the development of Internetware entities is actually to develop new traditional software systems with Internetware characteristics, as well as evolve existing traditional software systems to have Internetware characteristics. For supporting both types of the development of Internetware entities, we propose an architecture centric, component oriented software development method, called ABC (architecture-based component composition).

Take the self-adaptation for example, for the application-specific self-adaptation, ABC leverages existing software architecture (SA) techniques in a systematic manner: SA models are used to analyze expected qualities for locating the part of SA models to be adapted at first; dynamic SA records what should be done at runtime to achieve the desired qualities; finally, the designed adaptation is executed by runtime SA without stopping the running system. For the general enough adaptation, domain experts can use bad architecture patterns to define the ill structures that may lead to quality problems, and provide the corresponding good patterns. The runtime software architecture can automatically detect if the running system has bad structures, and dynamically refactor them into good structures. ABC also supports the introduction of rules into SA, and the entities governed by some rules can achieve rule-based self-adaptation by dynamically binding the containers with rule engines.

The on-demand collaborations of Internetware are controlled by runtime SA, that is, the development of on-demand collaborations is actually to develop the software architecture that controls Internetware entities. In that sense, ABC is still fit to the development of collaborations. But on the other hand, while the development of Internetware entities is still in the static, closed and controllable environment, the development of on-demand collaborations has to deal with the open,

dynamic and decentralized Internet environment. As a result, different from developing Internetware entities by ABC, developing on-demand collaborations by ABC should take the characteristics of entities, e.g. distribution, autonomy and heterogeneity, as well as the characteristics of entity interactions, e.g. diversity, complexity and changeability, into account.

4.3 Domain modeling for Internetware

Traditional software development processes are more suited for the relatively close, static and controllable environments. Most of them are top-down, that is, scoping the system border and using divide-and-conquer principles to make the whole process under control. However, the environment of Internetware has abundant resources and it is always opening, dynamic, and ever changing. The development over this environment can be seen as the composition of various "disordered" resources into "ordered" software systems. As time elapses, changes of resources and environments may "disorder" the existing software systems again, which will become "ordered" by the development sooner or later. The iterative transformation between "ordered" and "disordered" Internetware implies a bottom-up, inside-out and spiral development process. If only we can integrate these disordered entities during the development process and make them under "ordered" control, we can really achieve the development of entities and collaborations mentioned above.

As a systematic way to produce the reusable artifacts in a particular problem domain, domain engineering addresses the creation of domain models and architectures that abstract and represent a set of reusable assets within a domain through domain scoping, commonality and variability analysis and adaptable design construction based upon the study of the existing systems, knowledge from domain experts and emerging technology within a domain. Here, domain analysis refers to the process of identifying, collecting, organizing and representing the relevant information in a domain. In a sense, the domain analysis is a process in the bottom-up fashion, coinciding with the engineering of Internetware. Therefore, ABC adopts the methods and techniques of domain engineering to organize the underlying resources. We first construct the disordered resources over Internet as a domain model by domain scoping and analysis for representing high-level business goals of a set of Internetware systems; then generate the requirements specification for a new application by tailoring and extending the domain model; and finally implement the new entities and collaborations. When time elapses, the new application may be scattered somewhere on the Internet as a service and then becomes a new disorder resource. In turn, these new disordered resources can be added into the domain model by further domain analysis, and therefore the iterative process of disordered resources to ordered ones comes into being.

5 Conclusion

Internet brings new opportunities and challenges to the information technology, which therefore provides many new models and methodologies from different perspectives, such as grid computing, pervasive computing, service oriented computing, semantic web, web science, etc. Similarly, Internetware tries to investigate Internet from the perspective of software paradigm, that is, whether new or evolved software theory, methodology and technology for the open, dynamic, ever-changing and decentralized Internet are needed or not. This paper summarized the research and practice of Chinese scholars during the last five years, under the support of National Basic Research Program of China (973). Our achievements form an architecture centric technical

framework for Internetware, including a software model based autonomous entities and structured collaborations, a middleware implementing the software model and managing Internetware in an autonomic manner, and an engineering based on software architecture in the whole lifecycle.

Our future work will mainly focus on strengthening the current achievements in both depth and width. For the depth, we need to perfect the architecture centric technical framework, such as the intelligent trustworthy model, the autonomic management, the automation degree of engineering, etc. For the width, the future trend of integrating various networks will provide software a complex network environment, which consists of various heterogeneous networks like Internet, wireless networks, telecommunication networks, etc. Software running on such complex network environment is much more complex than that on Internet. However, we believe Internetware, though originated for Internet, and its technical framework provide necessary and efficient supports for such new environments.

1 Yang F Q, Lü J, Mei H, et al. Some discussion on the development of software technology. Acta Elect Sin (in Chinese), 2003, 26(9): 1104－1115

2 Lü J, Ma X X, Tao X P, et al. Research and progress of Internetware. Sci China Ser F-Inf Sci (in Chinese), 2006, 36(10): 1037－1080

3 Lü J, Tao X P, Ma X X, et al. Research on agent-based model for Internetware. Sci China Ser F-Inf Sci (in Chinese), 2005, 35(12): 1233－1253

4 Jiao W P, Mei H. Automated adaptations to dynamic software architectures by using autonomous agents. Eng Appl Art Intell, 2004, 17: 749－770

5 Cao J N, Feng X Y, Lü J, et al. Reliable message delivery for mobile agents: push or pull? IEEE Trans Syst Man Cyber, Part A: Systems and Humans, 2004, 34(5): 577－587

6 Wu W G, Cao J N, Yang J, et al. Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. IEEE Trans Comp, 2007, 56(8): 1055－1070

7 Cao J N, Ma X X, Chan A T S, et al. Architecting and implementing distributed web applications using the graph-oriented approach. Software - Prac Exp, 2003, 33: 799－820

8 Liu J, He J F, Liu Z M. A strategy for service realization in service-oriented design. Sci China Ser F-Inf Sci , 2006, 49(6): 1009－2757

9 Jiao W P, Mei H. Supporting high interoperability of components by adopting an agent-based approach. Software Qual J, 2007, 15(3): 283－307

10 Huang G, Wang Q X, Cao D G, et al. PKUAS: A domain-oriented component operating platform. Acta Elect Sin (in Chinese), 2002, 30(12Z): 39－43

11 Huang G, Mei H, Yang F Q. Runtime software architecture based on reflective middleware. Sci China Ser F-Inf Sci, 2004, 47(5): 555－576

12 Huang T, Chen J N, Wei J, et al. OnceAS/Q: A QoS-enabled web application server. J Software (in Chinese), 2004, 15(12): 1787－1799

13 Huang G, Liu T C, Mei H, et al. Towards autonomic computing middleware via reflection. In: Proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC). Hongkong, China, September 28-30, 2004. 122－127

14 Shen J R, Sun X, Huang G, et al. Towards a unified formal model for supporting mechanisms of dynamic component update. In: The Fifth Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE'05), 2005, 80－89

15 Mei H, Cao D G. ABC-S2C: Enabling separation of crosscutting concerns in component-based software development. J Software, 2005, 28(12): 2036－2044

16 Wang Q X, Shen J R, Wang X P, et al. A component-based approach to online software evolution. J Software Main Evol: Res Prac, 2006, 18(3): 181－205

17 Huang T, Ding X N, Wei J. An application-semantics-based relaxed transaction model for Internetware. Sci China Ser F- Inf Sci, 2006, 49(6): 774－791
18 Huang G, Liu X Z, Mei H. An online approach to feature interaction problems in middleware based systems. Sci China Ser F-Inf Sci, to appear in 2008
19 Mei H, Chang J C, Yang F Q. Software component composition based on ADL and middleware. Sci China Ser F-Inf Sci, 2001, 44(2): 136－151
20 Zhang W, Mei H. A feature-oriented domain model and its modeling process. J Software (in Chinese), 14(8): 2003, 1345—1356
21 Jin Z, Lu R Q. Automated requirements modeling and analysis: an ontology-based approach. Sci China Ser E (in Chinese), 2003, 33(4): 297－312
22 Pan Y, Wang L, Zhang L, et al. Relevancy based semantic interoperation of reuse repositories. In: Proceedings of the 12th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE2004), 2004, 211－220
23 Mei H, Zhang W, Zhao H. A metamodel for modeling system features and their refinement, constraint and interaction relationships. Software Syst Mod, 2006, 5(2): 172－186
24 Hou L S, Jin Z, Wu B D. Modeling and verifying web services driven by requirements: An ontology-based approach. Sci China Ser F-Inf Sci, 2006, 49(6): 792－820
25 Zhao W, Zhang L, Liu Y, et al. SNIAFL: Towards a static non-interactive approach to feature location. ACM Trans SoftWare Engin Meth, 2006, 15(2): 195－226
26 Jiang S J, Xu B W, Shi L. An approach to analyzing recursive programs with exception handling. SIGPLAN Notices, 2006, 4: 30－35
27 Tan G Z, Li C X, Liu H, et al. Research and application of traffic grid. J Comp Res Devel, 2004, 41(12): 2066—2072
28 Mei H, Huang G, Zhao H Y, et al. An architecture centric engineering approach to Internetware. Sci China Ser F-Inf Sci, 2006, 49(6): 702－730