Int J Software Informatics, Vol.3, No.4, December 2009, pp. 415–438 E-mail: ijsi@iscas.ac.cn
International Journal of Software and Informatics ISSN 1673-7288 http://www.ijsi.org

Internetware Computing: Issues and Perspective*

Wei-Tek Tsai[1,3], Zhi Jin[2], and Xiaoying Bai[3]

[1] (Department of Computer Science and Engineering, Arizona State University, USA)

[2] (Key Laboratory of High Confidence Software Technologies, Peking University, China)

[3] (Department of Computer Science and Technology, Tsinghua University, China)

Abstract The Internetware addresses the unique challenges of software development and maintenance in the open and dynamic Internet environment. The project identifies the four major features as the vision of future Internetware including autonomous services, dynamic collaboration, environment-aware evolution and adaptation, and verifiable and justifiable trustworthiness. The paper discusses four key enabling techniques to achieve the above Internetware capabilities. 1) The lifecycle model: it proposes the model-driven and reuse-centric adaptive lifecycle of service software and the mashup approach for composition-based application development. 2) The ontology system: it discusses a wide range of software development ontology systems that can be used at various abstraction levels throughout all the stages of software lifecycle, and the issues with ontology systems such as consistency and completeness, dependency analysis, merging and change management. 3) Modeling and simulation: it identifies the necessary characteristics of the modeling language in the new paradigm such as the modeling of environment , system and environment interactions, the environment-system co-engineering process, and the ontology support for modeling and simulation. 4) Social ranking: it points out that social network will play an important role in Internetware development framework. Traditional software activities such as requirements solicitation and testing can be improved following this social approach. From these above four perspectives, the paper gives an outlook to the emerging techniques and their potential power in Internetware software engineering.

Key words: Internetware

Tsai WT, Jin Z, Bai XY. Internetware computing: Issues and perspective *Int J Software Informatics*, **2009, 3(4): 415–438. http://www.ijsi.org/1673-7288/3/415.htm**

1 Introduction

The Web is becoming the computing platform as claimed by the number one principle of Web 2.0 "Web is the platform"[25]. This new Web platform will replace traditional computing platforms such as PCs and mainframes. If the Web is indeed the new platform and PCs or mobile devices are simply connection devices, software development on this Web platform will be different from software development in traditional platforms.

* This effort is sponsored by US Department of Education FIPSE project, Korean ETRI project, National Natural Science Fund for Distinguished Young Scholars of China under Grant No. 60625204, the Key Project of National Natural Science Foundation of China under Grant No. 90818026 and the National Basic Research and Development 973 Program under Grant No. 2009CB320701.
Corresponding author: Wei-Tek Tsai, Email: wtsai@asu.edu
Manuscript received 2009-08-09; revised 2009-11-18; accepted 2009-11-30; published online 2009-12-30.

In fact, the difference is already witnessed by the emerging techniques such as Web services, Service-Oriented Computing (SOC), Service-Oriented Architecture (SOA), and cloud computing[7]. Specifically, in cloud computing, software will be treated as a service (Software as a Service or SaaS) that runs on top of a virtualized environment of large number of resources. Cloud computing may involve virtualized systems including network operating systems (such as the new Chrome OS just announced in November 2009), scalable databases such as Google's BigTable, and applications that will run on top of significant computing capacity. With such vast infrastructure resources, one needs new software design and development strategies.

Many organizations have initiated software development on the Web, including those SaaS approaches. Some have even succeeded commercially, e.g., Salesforce.com is an outstanding example. Google is another example as it provides desktop applications on the Web like Google Docs, Google Code and Google App Engine. Microsoft also launched the Azure project to migrate their desktop applications like .NET and its operating systems to the Web.

Researchers in China name such a software development and execution framework Internetware[44]. The Internetware project studies, establishes, and applies scientific and quantifiable methods for software development, execution, and maintenance on the Web.

Essentially, Internetware emphasizes the open nature of Internet computing including the openness of software as well as software development. In fact, one important lesson learned from Google is that software needs to be open to gain market share[12], and if software development is indeed going to be open on the Web, related software technologies and tools will be open, which is likely to include modeling, architecture, design, code, test scripts/cases, and T&E (Test and Evaluation) results. This may mean that not only software but also software development techniques and tools will be available on the Web as services in the future. And they can be identified, discovered and composed to form new applications in a service-oriented manner.

Internetware faces four challenges[44]: 1) the autonomy of software components; 2) the dynamics of collaboration among software components; 3) the environment-aware evolution and adaptation; and 4) verifiable and justifiable system trustworthiness.

Autonomous Software Components: Modeling individual software components with their features forms the basis for the application development framework. In traditional systems, software components are generally modeled from two perspectives: 1) the *implementation* model that represents the structure and behavior of each component so that each component can be constructed, deployed, executed and evolved independently; and 2) the *interface* model that represents the computational functionalities and capabilities the component provides to others so that it can interoperate and collaborate with others.

However, for Internetware, as software components transform to services, they need to collaborate with each other dynamically in an open environment. Hence, autonomy becomes an important capability for loosely coupled yet dynamic composed services, with a variety of self-healing and self-adaptive capabilities. Internetware needs to sense the environment, accumulate knowledge via learning and reasoning, and adapt behavior to the changes either reactively or proactively. Autonomy enhances

the flexibility of cooperation so that it allows Internetware to build collaborations on-the-fly in an agile and cost-effective approach.

Dynamic Collaboration: In the Internetware framework, software components and their collaborations can be modeled separately and they can be developed independently. Software components can be developed using traditional techniques and wrapped into autonomous components with standard interfaces exposed for public access. Collaborations can be established on-demand by reusing components available over the Internet. Requirements-driven on-demand collaboration and collaboration-oriented coalition are two important issues.

Furthermore, as the software components are autonomous, they may change their decisions along with the environment changes. Autonomous components make decisions on-the-fly to join in or withdraw from the coalition and accept or refuse the collaboration with a specific partner in the coalition. Hence, the collaboration model changes along with the changing decisions of these autonomous software components. In summary, the dynamic environment triggers changes in components' decisions. As a result, the coalition changes over time, including the participants and their collaborations in the coalition.

Environment-Aware Evolution and Adaptation: Internetware are situated in an open environment. Changes can come all the time from the physical environment like unstable network connection, as well as business requirements like diversified user preferences and expectations. Hence, both software components and application systems need to evolve and adapt their behavior in accordance to environment changes, either reactively or proactively.

For this purpose, the context/environment of the systems should be modeled separately as well, in addition to the system model. The environment model can facilitate the abstraction of the environment features and the detection of environment changes. The Internetware can evolve and adapt with environment awareness by analyzing the environment models together with the system model. Context/environment awareness and self-adaptation are two important issues.

Verifiable and Justifiable Trustworthiness: Software trustworthiness has been recognized in safety-critical systems and fault-tolerant computing for a long time. It has been gained attentions recently. The system trustworthiness should be verifiable and justifiable.

Verifiable means that the system's trustworthiness is measurable and testable. Trustworthiness consists of many attributes such as correctness, performance, safety, security, and privacy. The overall trustworthiness of the application cannot be verified unless all the individual attributes are measurable and testable. However, isolated investigation of individual attributes is far from sufficient for judging the system trustworthiness. A holistic approach may be required.

Justifiable means that the system's trustworthiness matches the application requirements well. Internetware applications range over all kinds of applications such as e-banking, online shopping, e-healthcare and e-government applications. These applications demand a certain confidence level that the applications will function as intended. The failures of these application systems may cause significant damages. On the other hand, as higher confidence normally needs more efforts in system development, for those non safety-critical applications such as on-line games, the economic

choice is to have a reasonable confidence instead to reduce the development effort.

A good example of Internetware is the recently announced Chrome OS, a network OS developed from Linux, to be released in 2010. According to Ref.[24], Chrome OS is a revolutionary software as it has the following key features:

- **The Web browser Chrome is the OS**: Using a Web browser as the network operating system is not new as Netscape tried to do so about 15 years ago. However, Netscape did not deliver, but Google delivered it this year.
- **It is adaptive**: Chrome OS has many self-adaptive features especially security features. Applications run on Chrome OS are placed in "security sandboxes," like Jave security mechanism, and Chrome software checks the integrity of its code, and if the code is compromised, it will reboot to fix the problems.
- **All applications are Web applications without any installations**: All applications will be Web applications, and thus no installation, and this includes most popular applications such as Microsoft Office, TweetDeck, or Digsby and even Google's own Android.

While the claims of the Chrome OS are yet to be validated until the system is released in 2010, but the direction of Chrome OS represents a radical departure from the current software design:

- **Autonomy**: It treats the Web as the platform as no application software is installed, and every application need to be treated as a service, It has limited autonomy capabilities as the system keeps on checking the integrity of its code to ensure that nothing is compromised, and if compromised, it takes actions in a pro-active manner to reboot the system.
- **Dynamic collaboration**: Currently, Chrome OS still follows the traditional SOA framework and treat each software as a service. This is the simplest form of dynamic collaboration.
- **Environment-aware evolution and adaption**: Its environment is the Web, and it depends on search engines to identify the needed services. As the Web is an open and changing environment, the system is thus open and dynamic as one can view the system as "Chrome OS + the Web" instead just Chrome OS only.
- **Verifiable trustworthiness**: The Chrome OS verifies the digital signatures of application services all the time to ensure trustworthiness.

To address these issues, this paper presents four aspects of Internetware:

1. **Lifecycle model for Internetware**: Lifecycle is a fundamental issue that will affect the related technologies such as modeling, architecture, design, code generation, execution, monitoring, and evaluation. This will be discussed in Section 2.
2. **Ontology**: As Internetware uses ontology extensively[17,42,43], but originally ontology was designed for knowledge representation, sharing, classification, and

reasoning. To use an ontology system for software development, it needs to address software engineering issues such as correctness, cross referencing, architecture, design, test, verification, and change management. These issues will be dealt with in Section 3.

3. **Modeling and simulation**: As the Internet environment is open and dynamic, modeling and simulation will be an important issue as both the system and its environment[18,40] will be modeled using formal or semi-formal modeling languages for analyses and simulation. Environment modeling is important as the environment provides the context to evaluate the application objectively, and as the environment evolves, the system may need to be evolved accordingly. Furthermore, environment-system interaction needs to be modeled, evaluated, and evolved. Section 4 will address these issues.

4. **Social ranking for software evaluation**: Social behavior can be useful for many software development activities. This paper explores the possibility of using social ranking to rank software services, and the social ranking can be used either independently or together with physical evaluation and ranking. Section 4 will discuss this issue.

2 Internetware Lifecycle Models

What is an appropriate software development lifecycle for Internetware? While one may continue to practice traditional lifecycle models such as Waterfall, Spiral, rapid prototyping, and agile methods, but other new models need to be explored. Traditional lifecycle models often heavily depend on face-to-face meeting (such as agile methods) and documentation (such as Waterfall model). The differences between these traditional processes are sequencing and work breakdowns of these activities. For example, the Spiral process divides the software development into spirals, while agile methods allow flexible sequencing. While the Web provides face-to-face meetings via webcast and online documentation, it is not clear that they can be as effective as before if one uses traditional processes such as agile methods for developing Internetware.

IBM's SOA lifecycle model[10] is an interesting lifecycle model. This lifecycle, showed in Figure 1, has a linear feedback loop of *modeling*, *assembling*, *deploying*, and *managing* phases. The lifecycle is layered on a backdrop of governance and processes which enforce compliance and operational policies, and ensure change occurs in a controlled fashion with appropriate authority.

IBM SOA lifecycle begins with modeling: capturing business design from requirements and objectives, translating that into a specification of business processes, goals and assumptions, and creating a model. In the assemble phase, business design derived from model is converted to a set of business process definitions and activities. Designs of services are derived from activity definitions. Existing asset inventories which are artifacts to be assembled should be considered during design and implementation of business processes and services. The deploy phase of the lifecycle includes a combination of creating the hosting environment for applications and the actual deployment of applications. This includes resolving the application's resource

dependencies, operational conditions, capacity requirements, and integrity and access constraints. The manage phase of the lifecycle maintains the operational environment and enforces policies. This includes monitoring performance of service, maintaining problem logs, detecting and localizing failures, routing and recovering work, correcting problems, and restoring the operational state of the system. The manage phase also includes managing the business model, that is tuning operational environment to meet objectives expressed in business design, measuring success or failure to meet those objectives, ensuring policies that express the operational requirements of the business services and processes of the business design, and relating issues with that enforcement back to the business design.

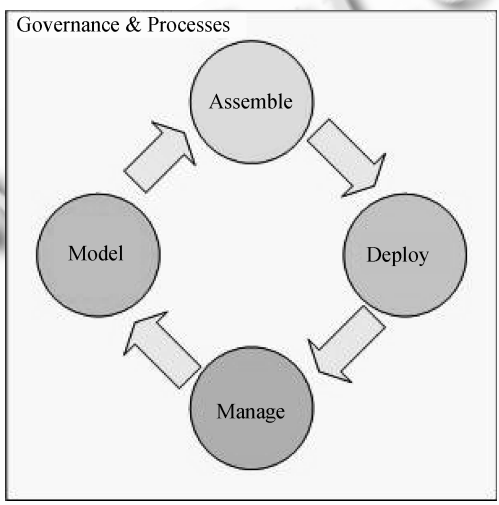

Figure 1. IBM SOA lifecycle

Another lifecycle model is Tsai's Service-Oriented Computing lifecycle as shown in Fig. 2[5]. In this lifecycle, reusability covers the entire lifecycle from identification to modeling, from modeling to simulation, from architecture to components, from automated code generation using code templates to reuse software services, from testing to verification, from monitoring to policy enforcement. For example, when a service is submitted for publication, it can be evaluated by various parties including service consumers, brokers and providers, using perhaps published evaluation mechanisms. The results of evaluation will be published too to provide guidance for all parties: service consumers may select better services, service providers can improve their services, and service brokers may publish better ranking and evaluation mechanisms for others to use. With appropriate support such as publishing, search, discovery, and analysis, these reusable artifacts can greatly reduce the time and effort in constructing software, and various lifecycle models can be developed by reusing these artifacts in software development.

Another lifecycle model is the current Web-based software development approach. One popular web method is *mashup*[5] where people uses a service and/or data provided by a service provider, and apply it in another application. A popular application is the real estate listing, and this is done by mashing up the real estate listing service

with the Google Map service. This is a form of SOC composition. In this approach, requirement analysis is relatively simple as users are familiar with this application from the beginning, and there is no need for new specification or the specification is already available with the associated specification analyses such as completeness and consistency checking, simulation and model checking performed already. The design is straightforward as it is a composition of two well-known services with a reliable data source, and thus there is no need of new design specifications. Few new code will be developed as the application is simply composed from two well-known services. There is little need of code verification as the mashup code and service code may have been verified and ranked by users before. However, users still need to be involved in validation as they should examine the system output carefully to ensures that all the houses in the listing are properly displayed, and whether the map provided by the Google map accurately reflect the location of the properties listed.

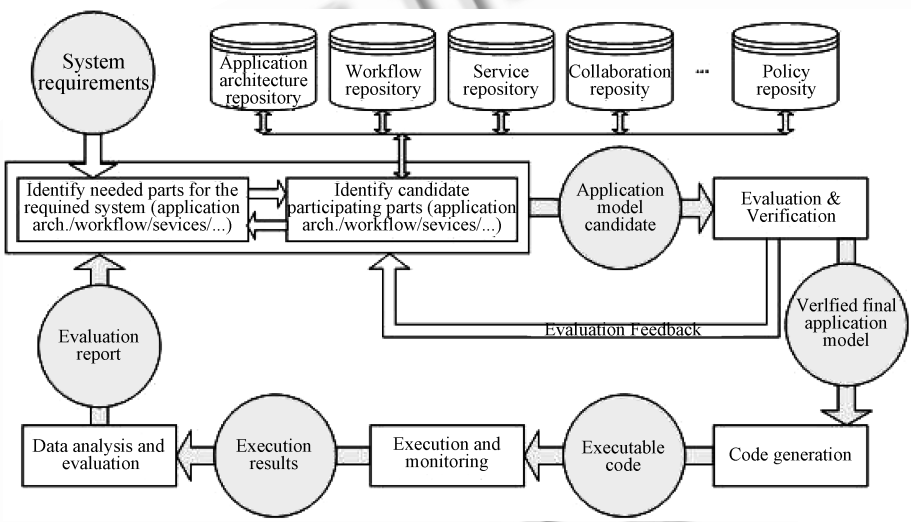

Figure 2. Tsai's service-oriented computing lifecycle

This mashup development process illustrates that a Web-based software development process can be different from traditional lifecycle models. Some key steps in traditional processes may be skipped, changed or even re-arranged.

First, knowledge, search, and discovery of existing reusable artifacts including architecture, services, workflows and test scripts become important. This leads to **search-based software engineering**[26]. This can be applied to requirements, design, services, tests, test results, oracles, monitors, and policies. For example, one can search a requirement template in a given requirement repository to specify a given problem. Once a requirement template is found, a search can be conducted to find design templates, design patterns, and design architecture in design repositories. Then search can be conducted to find suitable code templates and associated test scripts to implement and test the given software.

Second, once reusable items are identified, ability to reuse and modify these reusable items to create applications is important. This leads to **composition-based** (instead of construction-based) and **modification-based** software engineering. This

may require the need of composition algebra (interprocedural analysis, Petri net, data abstraction, design patterns have their composition rules). Table 1 shows the differences.

Table 1 Traditional software development vs. Web-based software development

	Traditional development processes	Web-based development processes
Activities	• Constructing all the artifacts. • Performing all the software process activities, e.g. requirement, design, coding, and testing.	• Discovering existing reusable artifacts. • Reusing already provided by artifacts. • Composing new applications from these existing artifacts.
Knowledge of existing artifacts	• Important in domain-specific product-line development processes. • Mainly used to improve the software development productivity and quality.	• Important in most Web-based application development. • Search and discovery capabilities are based on the knowledge about the existing artifacts.
Specifications	• A variety of specification languages such as UML are available. • These language can be used in most existing development processes.	• Ontology may be used to specify various artifact properties including both functional and nonfunctional attributes. • Different kinds of specifications languages are available to address different aspects in software development.
Verification and validation (V&V)	• A variety of V&V techniques are available, e.g. testing, simulation, model checking, theorem proving, and symbolic execution. • The emphasis is on independent verification & validation.	• Most traditional V&V techniques can be used. • The published artifacts may have been pre-verified and ranked before and after publication. • Newly composed artifacts may need on-demand integration with on-demand V&V. • The emphasis is more like collaborative verification and validation as multiple parties need to collaborate to complete V&V.
Data provenance	• Important for applications where data play a key role, e.g. data warehouse or data center.	• Important for most Web-based applications as artifacts will be shared among multiple parties and thus correctness and accuracy of application will depend on the data input.

What will be an appropriate lifecycle model for Internetware? Many web-based applications will be developed in the traditional model and then the developed software will be ported to the Web. This is the current approach. However, the following factors will gradually move traditional software development towards web-based development:

- More software artifacts will be available on the Web, and these artifacts will become candidates for reuse in composition. Moreover, many software development tools such as ontology editors, modeling tools, compilers, composition tools, simulators, model checkers, testers, C&C checkers, and policy enforcement tools, will be moved to the Web, and Web-based software development will become common. Microsoft takes this approach as it is porting most of her software development tools on the Web in the Azure project. The Azure will have .NET services (with access control, service bus, and workflows), live services, and SQL services.

- The mashup process is designed for end users who may not computer experts, and this kind of development processes will become dominant as there will be many end users who are not computer experts, but will use a variety of software services provided on the Web for applications.

The evolution from platform-based software development to Web-based software development or Internetware will take time, but as more software services and software development services will be available in the future, Internetware will become popular.

Some trends can be derived for Internetware lifecycle model:

- An important trend in Internetware lifecycle model will be reusability. This can be evidenced by the recent emphasis of publishing, search, discovery and mining in software development, and they are all related to software reusability.

- The reuse in Internetware may take place at different abstraction levels of the software models, e.g. the application framework, the business process model, the collaboration model, and in the different stages of the software development, e.g. assembling, testing, and evaluation. Publishing only makes reusable assets available, but specific items selected for reuse still need to be searched and discovered possibly by using data mining techniques. To facilitate the selection, not only software services can be published and reused, many other software artifacts can also be published and reused. In other words, reusability potentially can cover the entire lifecycle[33]. Another important issue in the lifecycle is the on-demand and adaptive nature of Internetware. An application may be dynamically composed to meet the changing needs and environment. The application may be re-composed fresh or can be developed by changing an existing application composition. These may require changes in the current computing infrastructure, and require the newly composed application to be tested and evaluated immediately during or after composition. Google's Chrome OS bypasses the test and evaluation by insisting on rebooting the system if it is compromised, as the original system is believed to be trustworthy, the rebooted system is thus trustworthy.

- As in this lifecycle model, everything is based on the model developed in the modeling phase, it is highly desirable that the modeling language to be in sync with the code. As such, the assembling phase will immediately follow the modeling phase and it will be followed by the deploying phase. If the model is not in sync with the code, when the development process enters the feedback loop, the model at hand may be rather different from the existing code, and thus making the next modeling phase difficult. Requiring model and code synchronization has been an issue as some has observed that UML model is often not in sync with the code. For example, Keith Short of Microsoft said[27]: *"But for whatever reasons, the existence of UML and UML-based tools, has not significantly changed the way developers build applications. Nor has it significantly contributed to developer productivity. Since Microsoft ships one of the most-used UML tools–those based on Visio in Visual Studio Enterprise Architect–we anonymously survey developers (not just our customers) on tool usage. We have discovered that it's a very small population who claim to use UML tools in*

support of their tasks, and most usage clusters around class diagrams and use case diagrams. When we drill into those who claim to use class diagrams, it's a tiny fraction that actually uses them to generate code. In fact, at Microsoft generally, we use UML for many purposes–mostly documentation or sharing of conceptual ideas–but almost never for any purpose where those documents relate to actual software development artifacts."

- It is important that the software engineering community develops a modeling language that is compatible with modern web-based software development or Internetware, and the models specified will be in sync with the code so that code can be automatically generated from the model.

3 Ontology Systems

Most Web-based technologies use ontology systems. They provide the definitions and vocabulary for a given domain. This kind of vocabulary is important as the artifacts on the Web are provided by different providers and used by different consumers. Many ontology languages are available with associated tools such as RDF, RDFs, OWL (including OWL-lite, OWL-DL and OWL-Full), OWL-S, SWSL, WSMO, and PSML-O.

When talking about the ontology-based approaches, one may ask questions, i.e. ontology of what (what kinds of knowledge or information will be specified), when it can be used (in what lifecycle process one can use the information specified in ontology), and how to organize it (what is the best way to represent the complex and dynamic information in the system), and how to apply it (how can a person use the information in ontology to create or understand software). A common way of applying ontology in current Web-based technologies is for service discovery for application composition. That is, knowledge about a service can be specified using the terms defined in an ontology system, and a service consumer can search the needed services using the terms specified in the ontology. The knowledge specified may include interface (including IOPE, or input, output, preconditions, and effects), internal process description, grounding rules, external use scenarios[30], and nonfunctional attributes.

However, only languages as well as the ontology about services cannot provide all the reusable artifacts in Web-based software development. While, what kinds of ontology systems are needed in Web-based software development for meeting the needs mentioned in last section? Our previous work[14,42,43] demonstrates that the ontology languages plus the software services alone are not enough for supporting the Internetware framework. Ontology for Web-based software development may need to conform to the following properties:

- First, ontology in Internetware is used not only for knowledge representation, sharing, classification and reasoning, but for *software development*. Software development often involves application modeling, architecting, code generation, test and evaluation, and software modification and maintenance. Thus these need to be addressed by ontology systems, as ontology will be used to produce software.

- Second, in accordance with the features of Internetware lifecycle model, the ontology for Internetware may be organized according to either software development processes or artifacts generated during the process. Software development often involves application modeling, test and evaluation (T&E), architecture, code generation, policy enforcement. Thus, one may classify the ontology into the following systems:

  – **Application ontology (AO)**: This specifies and classifies the software applications and will be directly reused or be adapted in the stage of application modeling;

  – **Collaboration ontology (CO)**: This defines various collaboration templates among different parties of an application and will facilitate the modeling of the collaboration models;

  – **Workflow ontology (WO)**: This specifies and classifies information related to workflows used in an application. In WO, specific workflows from different application domains are classified and relations are also specified facilitating collaboration. A domain-specific WO can be developed to facilitate rapid workflow identification and reuse;

  – **Service ontology (SO)**: This specifies and classifies information related to software services including its interfaces, and functionality description. This ontology helps the identification and discovery of the software services.

  – **Test ontology (TO)**: This defines concepts and relationships of test scripts and cases. It will help to conduct the ontology-based test case generation[1] for testing the generated application system;

  – **Policy ontology (PO)**: This defines and classifies policies used in a given application domain. It can be applied when deploying the generated application systems.

  The application will be refined from the application framework level, the collaboration models, the workflow models, to the service models and then be automated generated by discovering and composing the available software services.

- Third, instead of depending on keywords or feature matching when using an ontology system, it is possible to use the cross references or the context of specific items. For example, a given application in AO may reference several collaboration templates in CO, and each collaboration template may reference several services and workflows in SO and WO. In this way, a user may find the context of a given service or workflow by tracing from SO and WO to CO, and eventually to AO. Different from the current solution for service discovery, if one uses the ontology systems above with items specified can cross reference to each other, one can identify not only individual services, but also a cluster of services together with related workflows. This extends the current service discovery to cluster discovery where a cluster is a set of services, workflows, and related items such as policies that can work together as a sub-system of an application. This will expand service discovery algorithms from keywords

matching, semantic matching, and planning to include template matching, policy matching, workflow or algorithm matching.

- Finally, ontology systems may also be domain specific, i.e., the knowledge specified will be related to specific domains. The advantages of domain-specific ontology are that more in-depth analysis and reasoning can be developed for these ontology systems. For example, in Ref.[35], a set of ontology systems for smart-home application include domain ontology information that includes appliance devices, building facilities, and personal preference.

Furthermore, most of the above ontology systems are related to functional aspects, except TO and PO. These two may be related to constraints and verification aspects and deal with both functional and nonfunctional aspects. However, not only functional aspects can be specified and defined in ontology, nonfunctional properties can also be specified using ontology. For example, a nonfunctional ontology system[28] has been proposed to specify nonfunctional properties that is shown in Table 2:

Table 2 A list of non-functional properties

Name of Ontology	Content of Ontology
Locative ontology	address aspects of a service
Temporal ontology	the temporal (timing) aspects of a service
Availability ontology	information about service availability
Obligation ontology	information about contracts and obligation of services
Price ontology	information about pricing of services
Payment ontology	information about customer payment
Discounts ontology	information about customer discount
Right ontology	rights and royalty of intellectual properties
Trust ontology	various trusts of services
QoS ontology	quality of service including benchmark and ranking schemes
Security ontology	security properties about services and parties
IP ontology	information about intellectual properties
Rewards ontology	information about rewards of using services
Provider ontology	information about service providers
Measures ontology	information about measurement units
Currency ontology	information about currency for payment

The list is by no means comprehensive or exhaustive. But this indicates that nonfunctional aspects may have more things involved than functional aspects.

However, multiple ontology systems may arise some other issues such as consistency, traceability, and dependency. It is highly desirable that these ontology systems are specified using the same ontology languages so that consistency, traceability, and dependency among elements and relationships specified in these ontology systems can be maintained.

Ontology completeness and consistency: Is the concerned ontology complete and consistent, both internally with respect to knowledge specified in the ontology, and externally with respect to knowledge in other ontology systems? What are the completeness and consistency algorithms for the semantic information specified in these ontology systems? Note that many ontology systems will be used in Internetware including all the functional and nonfunctional ontology systems mentioned in

this section. If these ontology systems use different ontology languages and/or tools, the completeness and consistency issue may become problematic.

Dependency analysis: What are the dependency relationships among elements in the concerned ontology as well as in related ontology systems? Again, Internetware may involve many ontology systems, and they may depend on each other as they cross reference each other. This is critical for supporting rapid software development.

Ontology merging: While ontology has been proposed to the heterogeneity of different systems with different database design and concerns. As the Web will have many diverse ontology systems developed by different groups, ontology merging will become an important issue otherwise one cannot take advantages of those ontology systems that are available.

Ontology change management: Ontology updating is an issue as ontology systems need to be updated to include new knowledge and data, if the system is fully cross referenced, indexed, and analyzed, adding any new items into the system will require the system to re-analyze, re-reference, and re-index the updated ontology system. Note as ontology systems are used in software development, any changes may propagate to all the software programs developed using the changed ontology system. One way to address this problem is that ontology entries cannot be changed or deleted, and only new items can be added. This will minimize the change propagation. But is this a practical solution? Can an ontology system not allow any update but allow addition only in this open and dynamic Internet environment? If an ontology system allows changes in its existing entries, various change propagation mechanisms should be supported.

4 Modeling and Simulation

Note that the ontology systems are not application models or environment (or context) models for applications. It just provides definitions and vocabulary that can be used by software engineers to model applications and to mode environment of applications. Because of the on-demand and adaptive nature of Internetware, the development process in Internetware framework will be model driven and integrate both development and execution processes. Thus, modeling applications is the key issue in Internetware framework.

In fact, the service-oriented computing development lifecycle models, e.g., IBM SOA lifecycle in Fig.1, are also centered by the modeling. However, some existing SaaS applications do not follow this process yet. Instead, they follow the traditional engineering process, except that they design a software evolution scheme for the software to evolve after publication. In this way, the software design and implementation processes is slightly more involved than the current software development, but the kinds of evolution allowed will be decided at the design time.

To support a model driven and integrated development-execution process, one important issue is the selection or design of a modeling language. Is UML a suitable language for Internetware? Many UML modeling diagrams are mostly neutral to any software design or architecture, e.g., system sequence diagrams or system component diagrams can be considered architecture-neutral as they can be applied to OO application as well as SOC applications. However, some UML diagrams are designed mainly for OO modeling only such as class diagrams. Many UML extensions

are available to specify SOC applications[3,9,13,23,39,41]. For example, OMG proposes the SoaML (SOA Modeling Language)[23] to facilitate services modeling. SoaML is a standard extension to UML 2, and it provides a standard way to architect and model SOA solutions. It offers specification for the UML profile and metamodel for services, and is integrated with OMG Business Motivation Model (BMM). For example, the MUSIC approach[9] incorporates ontology and semantic web with UML to specify requirements in a ubiquitous computing environment. Wirsing and others proposed a semantic-based development of service-oriented systems, which includes service-oriented extensions to the UML, a mathematical basis formed by a family of process calculi, a language for expressing context-dependent soft constraints and preferences, qualitative and quantitative analyses, and model transformations from UML to process calculi[41].

However, it is important that the modeling language in Internetware framework needs to support the whole Internetware lifecycle model, specifically a model-driven process from modeling to assembling, from assembling to deploying, and from deploying to managing. Furthermore, the corresponding web-based infrastructure needs to be developed to support ontology, modeling, publishing, discovery, composition, testing, simulation, deployment, model checking, and evaluation based on the modeling language chosen.

Moreover, to support these features, a modeling language need to have the following characteristics:

- **Modeling Environments or Context**: Modeling an environment can be more difficult than modeling an application due to validation issues. About 10 years ago, an air-traffic control system falsely signaled an immediate danger as numerous aircrafts appeared in a fly zone. It turns out that those aircrafts are actually birds flying over the watched area, and the system mistakenly identified those birds as aircrafts. This is an example of the black swan phenomenon[28] as there was no record of that many flying birds can be detected by radar as aircrafts, and no one has anticipated this context during system development and even operation earlier. Thus, this example illustrates that environment modeling is inherently difficult. Furthermore, a large system may have a complex environment that includes external systems, operators, users, and intruders, and each of these are actually autonomous agents with their own goals, processes, and data. While significant progress has been made in modeling autonomous agents such as AML (Agent Modeling Language), ADEM (Agent-Oriented Development Methodology), and agent computing infrastructure including meta information, data storage, and execution engine, however, in general it is difficult to model every external agents' behavior, particularly those new and unexpected behaviors.

  In fact, any system can be considered as an environment for another system. For example, with respect to a heart simulator, a pacemaker can be considered as its environment, and vice versa. Thus, an environment needs to be modeled, analyzed, evaluated, and simulated like a system. As the system design now moves into the Web, the environment design also moves into the Web. In other words, it is possible to publish an environment system or agent about its interface (such as the traditional SOC interface IOPE), its internal process logic,

and its interaction with external systems. Once published, an environment can be discovered and used in composition to form a large composite environment just like a composite system can be composed using other subsystems as services. In fact, the entire SOC development processes and activities can be applied to develop an environment.

- **Formalizing Environment-System Interaction in a Service-Oriented Manner**: The interactions between the environment and the system need to be carefully designed, and evaluated during the environment design, monitoring, and policy enforcement during execution. The system-environment interaction can be formally specified, analyzed, simulated and even published, so that new environment system or agents can be developed to meet the specification of published interaction. Formal methods can also be used to model interactions to ensure safe system operation. Several system-environment interaction schemes have been proposed[30,40], and they can be further extended. These interactions, once formalized, can also be published in CCSOA (Consumer-Centric SOA[33]). For example, a user may want to publish a specification of system-environment interaction, and request software developers to develop software to meet the specification.

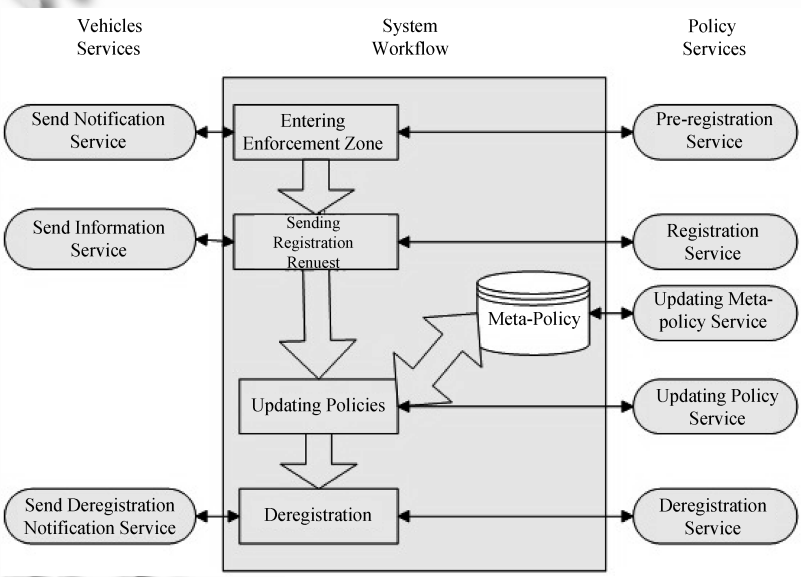

Figure 3. Environment and system

For example, as shown in Fig.3, the Vehicles system exposes its interactions with its external environment as services, such as notification service and information service. The Policy system, also exposed as services, captures the events in the interactions between the Vehicle system and its environment, detects the anomalies, and triggers the enforcement of the governance rules. The system workflow integrates the Vehicle system with the Policy system in specific business processes. The service-based approach enables dynamic configuration,

composition and evolution of the Vehicle system and Policy system. The servicetized Vehicle functionalities can be bound to various Policy services with different monitored events, detected fault models, regulation rules and enforcement strategies. Once a change occurs in the Vehicle system, the policies can be re-composed according to trace the changed interactions between the Vehicle system and its environment.

- **Integrated Environment-System Engineering Processes**: An important element of an environment is human. A human user may operate on the software according to the specified rules, and the person may use a completely different process. Fault-tolerant computing has learned that human operators are one of the major sources for system failures, and thus recently the human operation process and even the cognitive processes are modeled as a part of system modeling, and the human operation process can be treated as a part of integrated system engineering process. For example, MIT Engineering Systems Division now offers a track on human-systems engineering[21], and US government also started a human-centered system engineering process[22] where human operation processes are considered as an integral part of system processes during system development. In fact, if one considers other external systems as an integral part of the system, one can also propose an integrated environment-system system engineering processes where environment processes including autonomous human operation processes are considered. This system engineering process is different from traditional human factors or human engineering. Human factor engineering focuses on designing a system interface that will be easy to use and operate; however, human-system engineering is an engineering process where human processes are considered an integral part of the system processes including those fault-tolerant human operation processes and human cognitive aspects, and these processes are specified and analyzed during system requirement, design, coding and testing phases of system development. Thus, human-system engineering is a much broader process as it goes beyond user interface design.

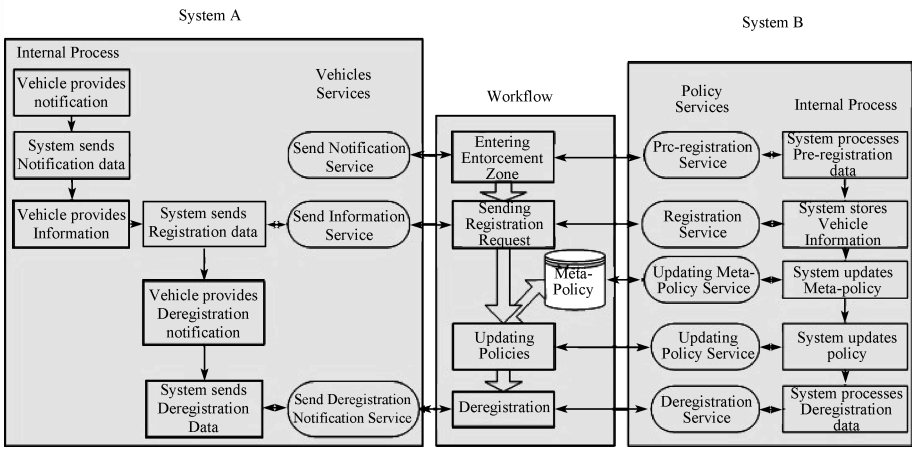

Figure 4. Environment-System co-engineering

For example, as shown in Fig.4, the Vehicle system is modeled as system A while its environment is modeled as system B. Each system can have its internal processes which can be built, and evolved as well, independently and in parallel. The workflow system serves as a glue between the two through their exposed service interfaces. In this way, each system is the environment of the other and each system can be built with awareness of the other. Intelligent mechanism can be built into each system to extend the functionalities so that the system can recieve/detect the events from its environment system and react to the events.

- **Environment-Based Security and Safety Issues**: For Internetware, the environment will be complex as it is an open system as new users may join after system operation, and system may be open too as new services may be available to be integrated into the system after deployment. Thus, it may be not possible to predict who will be involved and the kind of processes will be used during operation. The open Internet environment, together with new business and organizational practices, has increased the complexity of security and safety considerations dramatically. In such a setting, a system can potentially be interacting and sharing information with a large number of other systems, often on ad-hoc and dynamically negotiated configurations. Traditional models and techniques for characterizing and analyzing security and privacy may not be sufficient. Ref.[16] proposes an approach for quantitatively evaluating the security requirements based on an environment model.

  In general, any system modeling languages can be used to model an environment, e.g., some used state-transition diagrams[40,43], agents and agencies, UML, rules, and logic. In general, if the environment is modeled (possibly using vocabulary from a domain ontology), the environment model needs to be verified and validated. In some sense, any faults in the environment model and the mis-matching in the environment-system interaction may arise the safety and security issues[15].

- **Integrated System-Environment Engineering with Ontology**: Once system components and environment components can be formalized and published, they can also be composed to form larger systems and/or environments. Before a service can be published, it must be rigorously evaluated by service providers, and once published, it may be extensively evaluated and ranked by users, and evaluation may be based on test or evaluation ontology. Also, the evaluation methods and results can be published[31]. In this way, system and environment evaluation mechanisms and artifacts can be published (and composed like a system) using evaluation and test ontology systems. For example, individual test scripts for services can be composed to form an integration test script for a composed application.

The design of such a modeling language as well as corresponding modeling approach are an important issue for Internetware. Furthermore, Internetware will face additional issues: simulation and policy enforcement. Once a system and its environment have been modeled, they need to be evaluated including static and dynamic analysis. Two important dynamic analysis are simulation and policy enforcement.

If Internetware needs to be carried out rapidly, a simulation infrastructure needs to be developed and such simulation infrastructure is still at the beginning stage of development[33,34].

Table 3 summarizes the issues concerning the modeling and simulation.

Table 3 Modeling and simulation issues

	Activities and Issues
Modeling	• System modeling using original system requirements, ontology, and various published items such as services, workflows, and application templates. • Design of modeling languages that can specify services, workflows, system architecture, and is compatible with ontology and software development.
Assembling	• Software is designed by composition using discovered services, workflows, collaboration templates, and application templates. • Various software evaluations need to be conducted including simulation, model checking, completeness and consistency checking, and system integration testing. • As system may be just composed, these activities need to be performed on-demand immediately after composition.
Deploying	• Software deployment to an execution environment on the Web, so the execution environment needs to allow new software to be integrated in existing running software.
Managing	• Software execution will be monitored, so the modeling language needs to allow application execution to be observed. • various runtime policies will be enforced at runtime, so the execution environment needs to support police enforcement at runtime.

5 Social Ranking for Software Evaluation

As Internetware assets will be published in Internet and these assets will be reused in the future software development. Social networking will play a role in Internetware software development framework, social networking can be used in developing requirements, specifications, workflows, software services, test cases, test scripts, inspection and policies in a wiki manner. In this way, people collaborate using a wiki program to develop a software artifact over the Internet. That will become a typical scenario in Internetware software development framework.

An experiment has been done to the ranking based software testing in service-oriented manner[37]. It assumes that there are plenty of available test services and test scripts distributed over Internet. These test services and test scripts have been ranked for helping the selection for reusing. The ranking can be based on physical ranking, e.g., the number of failure found, or social ranking, e.g., the review by the community participants.

On the one hand, before a test service or test script is published and checked out from the repository, it should be verified to ensure its quality. This can be done by a service broker, and it can verify any service or test scripts submitted. Most of these ranking are based on test results (for example, those potent test cases will

be ranked high), coverage (certain coverage criteria may be ranked high for certain testing applications), or component tested (for example, the test cases that execute a modified section will have a high priority during regression testing). In other words, these are often based on physical items such as the test cases, test results, or software components. That is the physical ranking.

Suppose that there is a group of services $S = \{s_i\}$ of the the same type. We can use the Bayesian ranking method[1] to evaluate the relative ranking of a service s in the group as follows:

$$WR(s) = \frac{(\frac{1}{n}\sum N_i) \times (\frac{1}{n}\sum R_i) + N_s \times R_s}{(\frac{1}{n}\sum N_i) + N_s} \tag{5.1}$$

Where $WR(s)$ is the ranking of service s; N is the total number of test runs of each service s_i in the group; R_i is the pass rate of each service s_i; N_s is the test runs of the evaluated service s; R_s is the pass rate of s; and n is the total number of services in the group, that is the size of S. Taking P as the number of passes of a service, it can be simplified as follows:

$$WR(s) = \frac{(\frac{1}{n}\sum P_i) + P_s}{(\frac{1}{n}\sum N_i) + N_s} \tag{5.2}$$

In the formula above, all of the test results are treated equally. However, many factors may affect the pass rate of a service, such as the goodness of test cases and the skill of testers. For example, an experienced tester may select more suitable test cases with higher credibility to validate the service functional and non-functional properties, compared with the inexperienced testers. Hence, we can introduce "credibility" attribute to each test execution. In the formula below, the pass number of each service is revised with a weight k to indicate the credibility of each test run and the confidence of each test result[45].

$$WR(s) = \frac{(\frac{1}{n}\sum k_j \times P_{i,j}) + \sum k_j \times P_{s,j}}{(\frac{1}{n}\sum k_j \times N_{i,j}) + \sum k_j \times N_{s,j}} \tag{5.3}$$

In addition, one can also use a social process, i.e., opinions and comments by peers and/or participants. In fact, while physical T&E has been the principal approach for software development organizations, end users often use software due to social evaluation and ranking. For example, few people test office software before application, and they use the software because it has been ranked high socially. Similarly, people seldom perform extensive T&E before they purchase a car, and they often buy a car due to social ranking (for example, they read user feedback and ranking on the Web).

Thirteen factors are identified in social ranking[38], the following list show those related to testing services:

- **Author rank**: An item produced by a high-ranking author will have a high rank.
- **Usage rank**: An item used extensively will have a high rank, and the usage data will be contributed by participants.
- **Application rank**: An item used in critical applications will have a high rank.

- **User rank**: A service used by important organizations or groups will have a high rank. For example, a service used by a government agency will have a higher rank than a service used by an unknown organization.

- **User feedback rank**: A service can be directly ranked by users. Note that a user is also ranked, and thus a highly ranked user will have more weight.

- **Release time rank**: Recently released software may serve current needs, and thus it may be placed in a high priority list for discovery if it has other attributes such as developed by highly ranked authors or organizations.

All the physical and social items will be ranked to ensure that the best assets will be used first. Physical ranking and social ranking on the same assets have different meanings and both of them may contribute to the final ranking. While social ranking does not provide physical evaluation, it reflects the consumer sentiment. Thus, social ranking may represent extensive experience or practice. For example, a software program is ranked because it has been used extensive by a large number of users for an extended period of time, and thus it is likely that it will be a quality product. However, no software with low physical or social ranking can be used. For example, a software program with high physical ranking but low social ranking may indicate issues not addressed by common T&E activities such as low usability. Table 4 summarizes the discussion.

Also, rankings can be continuously updated as more data are collected. In this way, Internet software will be evaluated continuously updated.

Table 4 Physical and social ranking

Physical ranking	Social ranking	Recommendation
Not available	High	Use only for non-critical applications.
Low	High	Not recommended.
Low or not available	Low or not available	Not recommended.
High	High	Good for application.
High	Not available	Recommended, but it will be better if it is also socially ranked.
High	Low	Not recommended.

Another experiment has been done to the trust ranking of the software services[46]. This experiment assumes that the published software services are service agents. These agents have goals and intentions and can perform actions to supply services for achieving the goals. Furthermore, they show sociality by making commitment so that the service selectors can choose them and delegate tasks to them. When a service selector (normally the software developers) tries to select an agent to delegate it a task, the service selector need to inference the trust value based on its knowledge on this agent.

This experiment also takes Castelfranchi's view[4] on social trust beliefs for conceptualizing the trust-related information. It combines also the objective criteria and the subjective or social criteria. It classifies the information that can be used in trust measurement into six dimensions: the competence belief, the intention belief, the integrity belief, the persistence belief, the predictable belief and the reputation belief.

Among them, the first five are physical criteria that can be inferred from the description of the agent (the competence belief and the intention belief) or the history of the services the agent has offered (the integrity belief, the persistence belief and the predictable belief). While, the sixth criterion i.e. the reputation belief, is the social ranking of the service supplied by the agent.

Efforts are taken on relating those physical trust beliefs with the capability description of agents and the historic information about the service. Different from the physical ranking in the first experiment, the physical trust beliefs are inferred results by using some inference rules rather than by some physical measurement. These rules can be described informally as follows.

- Competence rule 1: If an agent knows how to perform an action and it has the resource for performing the action, then it can perform the action.
- Competence rule 2: If an agent knows that supplying a service needs performing a set of actions and it can perorm all the actions, then it can supply the service.
- Intention rule 1: If an agent wants a consequence and it knows that performing an action can result in the consequence, then it intends to perform the action.
- Intention rule 2: If an agent commits to supply a service and the service means a set of actions, then the agent commits to perform all of the actions and intends to perform all of the actions.
- Integrity rule 1: If an agent commits to supply a service and the service needs a set of actions but the agent is not able to perform one of the actions, then the agent is not integrity.
- Persistence rule 1: If an agent can supply a service and it commits to supply the service and the service means a set of actions but the agent does not perform one of the actions, then the agent is not persistent.
- Predictability rule 1: If an agent has goals and it supplies service for achieving some of the goals but it does not supply services for achieving another goals, then the agent is not predictable.

The second effort is scoring the integrity, the persistence, the predictability and the reputation of a service agent. These scores change along the interaction between the agent and the service selector. For the integrity, the persistence and the predictability, the scores increase by 1 for one interaction that meets the corresponding condition and decrease by 1 for one interaction that does not meet the corresponding condition. For the reputation value, we argue that the strategy should make the reputation score to increase slowly and decrease fast for encouraging agents to act successfully and avoid failure. Thus, for an candidate service agent a, after an interaction:

$$Integrity(a) = \begin{cases} Integrity(a) + 1 & \text{if it is integrity in the current interaction} \\ Integrity(a) - 1 & \text{otherwise} \end{cases} \tag{5.4}$$

$$Persistence(a) = \begin{cases} Persistence(a) + 1 & \text{if it is persistent in the current interaction} \\ Persistence(a) - 1 & \text{otherwise} \end{cases} \tag{5.5}$$

$$Predictability(a) = \begin{cases} Predictability(a) + 1 & \text{if it is predictable in the current interaction} \\ Predictability(a) - 1 & \text{otherwise} \end{cases} \tag{5.6}$$

$$Reputation(a) = Reputation(a) - (1 + \lceil Reputation(a) \times \alpha + \beta \rceil) \tag{5.7}$$

Here, $a \in (0,1), b \in [0,1)$ are two factors. With these rules, when selecting an agent for delegating a task, the service selector can judge the trustworthiness of the candidate service agent as follows:

$$Trust(a) = \begin{cases} \frac{1}{n} \times (p1 \times Integrity(a) + p2 \times Persistence(a) + \\ p3 \times Predictability(a)) \otimes Reputation(a) \\ \qquad\qquad \text{if } a \text{ is competent and has intention} \\ -\infty \qquad\qquad \text{otherwise} \end{cases} \tag{5.8}$$

Here, $p1, p2, p3, p4$ are the service selector's preference on the four trust-related dimensions, $p1, p2, p3, p4 \in [0,1]$ and $p1 + p2 + p3 + p4 = 1$. $\otimes$ is a symbol that represents that trust value consists of two parts.

Different from the global social ranking in the first experiment, this experiment emphasizes the individual judgement on particular software assets. Each service selector may have its own interaction history with the software assets and may have its own preference on different trust dimensions. Hence, each service selector in Web-based software development has its own trust value to any particular software asset. The trust value is derived by its own inference engine based on the known history and current interaction information about the assets. As such, the autonomy of the service selectors can be captured to a certain extent.

6 Conclusion

Internetware presents many new issues not faced by traditional software development. The new issues mainly from the open, dynamic, and distributed nature of the Internet environment. While many new techniques such as ontology, SOC, environment modeling, social ranking, cloud computing, and adaptive control mechanisms, are being developed to address these issues, these new techniques still need to be evaluated in the Internet environment to demonstrate their feasibility and effectiveness.

References

[1] Allen C, Appelcline S. 'Collective Choice: Rating Systems'. http://www.lifewithalacrity.com/2005/12/collective_choi.html.

[2] Bai X, Lee S, Tsai WT, Chen Y. Ontology-Based Test Modeling and Partition Testing of Web Services. Proc. of IEEE Web Services, 2008. 39–46.

[3] Berkem B. From BMM to SOA. Journal of Object Technology, 2008, 7(8): 57–70.

[4] Castelfranchi C, Falcone R. Social trust: cognitive anatomy, social importance, quantification and dynamics. Autonomous Agents 1998 Workshop on Deception, Fraud and Trust in Agent Societes, 1998. 35–49.

[5] Chen Y, Tsai WT. Distributed Service-Oriented Software Development. Kendall Hunt, 2008.

[6] The Chicago Manual of Style. University of Chicago Press, Chicago 60637, USA, 1982.

[7] Cloud Computing: Wikipeida entry. http://en.wikipedia.org/wiki/Cloud_computing, 2009.
[8] Forgaard R. A Program for Generating and Analyzing Term Rewriting Systems[Master's Thesis]. MIT Lab. for Computer Science, 1984.
[9] Geihs K, Khan MU, Reichle R, Wagner M. Modelling Notation for Adaptive Applications in Ubiquitous Computing Environments. Project Report. http://www.ist-music.eu/MUSIC/results/musicdeliverables/techreportreference.2008-02-21.3918111699, 2008.
[10] IBM Developers Works: IBM's SOA Foundation: An Architectural Introduction and Overview. http://www.ibm.com/developerworks/webservices/library/ws-soa-whitepaper, 2005.
[11] Adams H. Mashup Business Scenarios and patterns. http://www.ibm.com/developer-works/lotus/library/mashups-patterns-pt1, 2009
[12] Jarvis J. What Would Google Do? Collins Business, 2009.
[13] Johnson SK, Brown AW. A Model-Driven Development Approach to Creating Service-Oriented Solutions. http://www.ibm.com/developerworks/webservices/library/ar-soaspl/, 2006.
[14] Liu Z. Requirement Elicitation and Modeling for Service-Oriented Requirements Engineering[Master Thesis]. Arizona State University, 2008.
[15] Liu C, Wang Y, Jin Z. Elicit the Requirements on Software Dependability: A Knowledge-Based Approach. APSEC, 2009.
[16] Long T, Liu L, Yu Y, Jin Z. AVT Vector: A Quantitative Security Requirements Evaluation Approach based on Assets, Vulnerabilities and Trustworthiness of Environment. Proc. of IEEE International Conference on Requirements Engingeering, 2009: 377–378.
[17] Lv J, Ma X, Tao X, Xu F, Hu H. Research and progress of Internetware. Science in China Series F: Information Sciences, 2006, 36(10): 610–622.
[18] Lv J, Ma X, Tao X, Cao C, Huang Y, Yu P. On Environment-Driven Software Model for Internetware. Sciences in China F: Information Sciences, 2008: 683–721.
[19] Mahout A. http://lucene.apache.org/mahout/, 2009.
[20] Mei H, Huang G, Lan L, Li J. A Software Architecture-Centric Self-Adpation Approach for Internetware. Science in China F: Information Sciences, 2008: 722–742.
[21] Human-Systems Engineering (HSE). http://esd.mit.edu/hse/.
[22] Human-Centered Systems Eniginering Process Guidance. www.hf.faa.gov/docs/508/docs/Human_System_Engineering_(NSWC).pdf
[23] OMG: Service Oriented Architecture Modeling Language (SoaML) -Specification for the UML Profile and Metamodel for Services (UPMS). http://www.omg.org/docs/ad/08-08-04.pdf, Aug. 2008.
[24] Parr B. With Chrome OS, Google Intends to Destroy the Desktop and Microsoft. http://mashable.com/2009/11/19/impact-of-chrome-os/, Nov. 2009.
[25] T. O'Reilly: What is Web 2.0?. Available at http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html, 2005.
[26] Search-based Software Engineering, Wikipedia. http://en.wikipedia.org/wiki/Sear-ch_Based_Software_Engineering.
[27] Short K. UML and DSLs Again. http://blogs.msdn.com/keith_short/archive/2004/04/16/114960.aspx
[28] Toma I, Foxvog D, De Paoli F, Comerio M, Palmonari M, Maurine A. Non-Functional Properties in Web Services. CMS WG Report, 2008, D3 v. 0.1.
[29] Taleb NN. The Black Swan: The Impact of Highly Improbable. Random House, 2007.
[30] Tsai WT, Song W, Paul R, Cao Z, Huang H. Service-Oriented dynamic reconfiguration framework for dependable distributed computing. Proc. of the 28th annual International Computer Software and Applications Conference (COMPSAC), 2004. 28–30.
[31] Tsai WT, Chen Y, Paul R, Huang H, Zhou X, Wei X. Adaptive Testing, Oracle Generation, and Test Case Ranking for Web Services. COMPSAC, 2005(1): 101–106.
[32] Tsai WT. Service-Oriented System Engineering: A New Paradigm. Proc. of IEEE International Workshop on Service-Oriented System Engineering (SOSE), 2005. 3–8.
[33] Tsai WT, Xiao B, Paul R, Chen Y. Consumer-Centric Service-Oriented Architecture: a New Approach. Proc. of IEEE WCCIA, 2006. 175–180.
[34] Tsai WT, Fan C, Chen Y. DDSOS: A Dynamic Distributed Service-Oriented Simulation Framework. The 39th Annual Simulation Symposium (ANSS), 2006. 160–167.
[35] Tsai WT, Huang Q, Sun X, Chen Y. Dynamic Collaboration Simulation Framework in Service-

Oriented Computing Paradigm. Proc. of ANSS, 2007. 41–48.
[36] Tsai WT, Cao Z, Wei X, Paul R, Huang Q, Sun X. Modeling and Simulation in Service-Oriented Software Development. Simulation, 2007, 83(1): 7–32.
[37] Tsai WT, Zhou X, Chen Y, Bai X. On Testing and Evaluating Service-Oriented Software. IEEE Computer, 2008, 41(8): 40–46.
[38] Tsai WT, Zhong P, Bai X, Alston J. Role-Based Trust Model for Community of Interest. Proc. of IEEE SOCA, 2009.
[39] UN/CEFACT: UN/CEFACTs Modeling Methodology (UMM), UMM Meta Model - Foundation Module. Technical Specification V1.0. http://www.unece.org/cefact/umm/UMM Foundation Module.pdf, 2006.
[40] Wang P, Jin Z, Liu L, Cai G. Building Towards Capability Specifications of Web Services Based on an Environment Ontology. IEEE Transactions on Knowledge and Data Engineering, 2008, 20(4): 547–561.
[41] Wirsing M, et al. Semantic-Based Development of Service-Oriented Systems. Proc. International Conf. Formal Techniques for Networked and Distributed Systems, 2006. 24–45.
[42] Wu B. Service-Oriented Modeling: A Reuse-based Approach[Ph.D. Thesis]. Beijing: Academy of Mathematics and System Science, Chinese Academy of Science, 2009.
[43] Wu B, Jin Z, Tsai WT, Liu Z. Service-Oriented Modeling: A Reusability Approach IEEE Transactions on Service Computing, 2009. (to appear)
[44] Yang F, Lv J, Mei H. Technical Framework for Internetware: An Architecture-Centric Approach. Science in China Series F: Information Sciences, 2008. 610–622.
[45] Zhang Y, Bai X, Jiang C. A Technique for Evaluating Services Based on Improved Bayesian Voting Algorithm. J of Computer Science, 2008, 35(04): 255–259.
[46] Zhu M, Jin Z. An Agent-Based Trust Model for Service-Oriented Systems. R. Falcone, K. Suzanne Barber, J. Sabater-Mir, Singh MP(Eds.): Trust in Agent Societies, 11th International Workshop, TRUST 2008, Estoril, Portugal, May 12-13, 2008. Lecture Notes in Computer Science 5396: 162-181 Springer 2008.