10th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), Sedona, February 2005, pp. 139 - 147.

Specification-Based Verification and Validation of Web Services and Service-Oriented Operating Systems

Wei-Tek Tsai, Yinong Chen, Ray Paul*
Department of Computer Science and Engineering, Arizona State University
Tempe, AZ 85287-8809, U.S.A.
*Department of Defense, Washington DC, U.S.A.

Abstract

Service-Oriented Architecture (SOA) and Web Services (WS) have received significant attention recently. Even though WS are based on open standards and support software interoperability, but the trustworthy issues of WS has actually limited the growth of WS applications as organizations do not trust those WS developed by other vendors and at the same time they do not have access to the source code. This paper addressed this issue by proposing several solutions including specification-based verification and validation, collaborative testing, and group testing. The key concept is that it is possible to provide a comprehensive evaluation of WS even if their source code is not available.

Keywords: Web services, service composition, collaborative testing, group testing, verification.

1. Introduction

Web Services (WS) have received significant attention recently as it is based on a simple yet powerful Service-Oriented Architecture (SOA) that supports easy software interoperability via standard WS protocols such as SOAP, UDDI, WSDL, as well as other standard protocols such as OWL-S, CDL, BPEL. Many production and research projects have been initiated and carried out, aiming at finding a new way of developing and using software, by major computer and software corporations, such as IBM, Microsoft, Oracle, Sun Microsystems, and SAP, as well as by universities, research laboratories, and government agencies such as NCES, GIG-ES, JBMC2, FORCEnet projects within U.S. DoD.

In theory, with this elegant way of WS interoperability via standard protocols, software productivity will be improved significantly because interoperability promotes software reusability. While it is true that numerous WS products are now available for public use online, the corporations have not made use of each other WS or integrate WS developed by others into their WS, especially, when the WS to be developed is mission- or business-critical. The main issue is the trust:

Can the WS developed by one organization be used in a trustworthy application in another organization?

The answer is unfortunately "no" at this time. Even companies that push for interoperability of WS and SOA technology refuse to use WS developed by another organization if the WS source code is not available. The consequence is staggering and is affecting the current WS development and applications.

Companies that support and promote WS and SOA technologies are willing to integrate the WS developed by other companies if the source code is available. However, the WS developers may not be willing give the source code away, and thus the technology designed to promote interoperability may not reach its original goal due to reasons not related to technology.

If the current trend persists, we will have several major clusters of WS, WS within each cluster will be able to interoperate with each other and work in a seamlessly manner. However, WS from different clusters may not be able to interoperate in an integrated application with each other in spite of the fact all the WS in each cluster use the same WS and SOA standard protocols such as WSDL, OWL-S, SOAP and UDDI.

This is a serious challenge for the original goal of WS and SOA, i.e., software interoperability via standard protocols. This issue arises due to the trustworthiness, which includes not only security, but also reliability and safety issues.

Computer users have used software without having access to their source code for years. People use the software because of the reputation, the authorized distribution channels, and availability of objective evaluation by the third parties. The problem with WS is different: WS can be offered by unknown providers through internet, which can be searched, bounded, and executed at runtime. Based on our research on WS testing [26-32], we propose several methods to address the trustworthiness of WS offered over the internet:

- Specification-based Verification and Validation (V&V): Proposed in this paper, V&V will be based on WS specifications only, and thus bypass the need of the WS source code. Techniques include Completeness and Consistency (C&C) analysis, model checking, and test case generation based on WS specifications only.
- Collaborative V&V or CV&C: Instead of solely depending on WS developers, WS V&V should depend on all parties involved: WS clients, brokers, providers, regulators. For example, all of them can contribute test cases for WS testing. Moreover, it is desirable to have a neutral 3rd-party organization to perform objective evaluation of WS using the contributed test cases. The role of the organization is similar to the Consumer Reports for consumer products and NHTSA (National Highway Traffic Safety Administration) for rating auto safety, but this organization will test and evaluate WS and publish the evaluation and ranking of WS evaluated.
- Group testing: It is expensive to test a large number of candidate WS for applications, and group testing techniques, originally proposed for blood testing, can be used for WS testing [27]. The key advantage of this approach is that the test oracle can be established and intelligent algorithms are available to test a large number of WS rapidly.

Furthermore, once the trustworthiness issue is addressed, SOA should not stop at the application level. We propose in this paper to extend SOA to the entire software system, all way through from the kernel of the operating system to the application layer.

The remainder of the paper is organized as follows. Section 2 outlines the WS development process. Section 3 discusses existing WS testing techniques, identifies the missing pieces in current WS testing techniques, and presents three new WS testing techniques. Section 4 proposes a full SOA that completely reorganize the entire software system into a single layer of services. Finally, section 5 concludes the paper.

2. Developing WS applications

WS development process is significantly different from that of traditional software that consists of specification, design, implementation, and testing. New WS can be composed at runtime using WS found over the Internet. For example, a new travel agency service can be composed at runtime from existing travel agency services, airline services, rental car services, and hotel services.

When planning a new WS application, traditional development steps such as specification, design, implementation (coding), and testing are not sufficient. In the WS applications, development steps could be supplemented by searching, discovery, matching, dynamic composition, remote invocation, remote verification, and remote monitoring. Existing WS can be searched, located, remotely invoked to deliver the required functionality or a part of the functionality. A new service thus can be composed dynamically and at runtime, completely or partially using existing WS available over the Internet.

It is a business decision, when planning new WS, whether to use (purchase) existing WS to compose the new WS or to pay developers to implement new WS from scratch and sell the WS on the Internet for profit. The former approach has the advantage of rapid development while the latter approach could be more cost effective in long term. As a result, a large number of alternative WS may exist for any given WS specification.

Figure 1 illustrates a typical development process of a WS application. First, the WS specification is written in a WS specification language such as OWL-S, DMAL-S, WSDL, and WS-CDL, etc. A decision needs to be made in the first place if to decompose a Web Service into multiple modules. If no decomposition is necessary, the WS will consist of only one module, otherwise, it will consist of multiple modules. For each module, WS search will be performed and the found WS will be tested. Other factors such as the costs of using existing WS and the costs of writing new WS may be taken into consideration. After unit testing, the required WS will be composed from the modules. The composed WS will be tested for their interoperability and overall functionality among the modules. For the functionality, the composed WS is tested as a unit and techniques for unit testing can be applied.

The challenges imposed on testing WS include

- Test WS developed by unknown developers without the source code. Often only the WS developer has the access to the source code, while the other parties only need to know the quality of the WS.
- Test WS at runtime: One major challenging in testing WS is that the unpredictable environment that the WS may run on. This issue is especially serious in WS orchestration, which involves multiple organizations rather than one. It is difficult to

estimate how many clients will access the WS simultaneously, how they invoke them, or whether an unauthorized client or broker invokes the WS. WS testing includes all of the performance, scalability, reliability, availability, security, and stress/load testing aspects for traditional software, but the specialty and distributed property of WS also make WS testing difficult and complicated, and the entire V&V of WS also becomes critical for practical applications.

- Test large number of available WS as potentially a WS consumer may need to choose a WS from hundreds of candidate WS available on the Web for application.

.

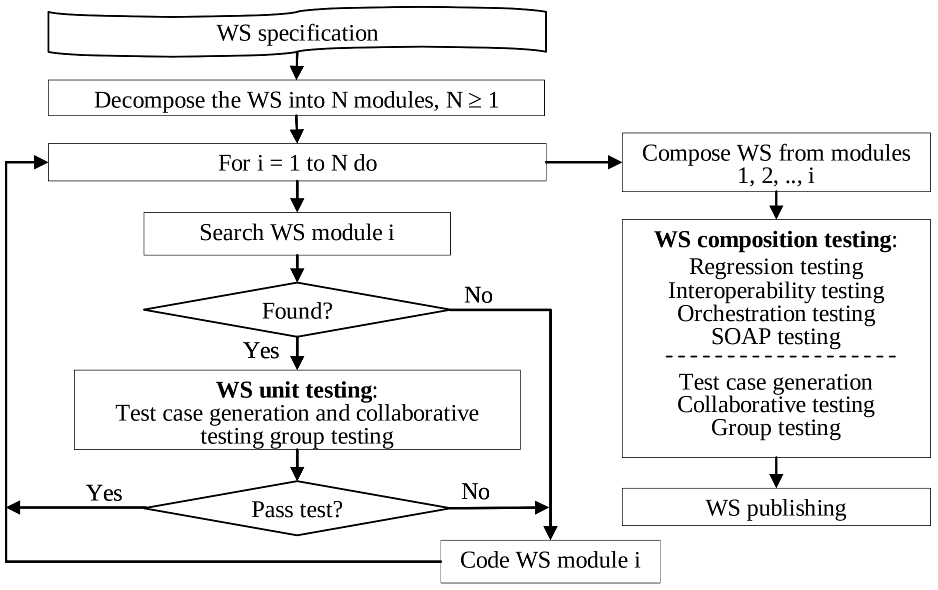

Figure 1. Development process of Web Services

3. Specification-based WS testing

Current WS testing techniques assume component WS have been tested properly by the WS providers and thus focus on integration testing of composite WS. This assumption is not acceptable if the composed WS need to be trustworthy. In a trustworthy system, every component must be verified before being used in a composite WS. We thus propose to perform rigorous unit test on the components of WS to be used in trustworthy composite WS.

3.1 Current techniques and tools

As a software system, WS can be tested by traditional software testing techniques, such as functional testing, regression testing, performance testing, security testing, stress/load testing, availability testing, safety testing, versioning testing, compatibility testing etc. Versioning testing and compatibility testing are especially important in WS composition. Due to its Web environment, certain extensions to traditional software testing are necessary, including UDDI testing, including WS publishing, querying, and binding capabilities; SOAP message testing which including intermediary capabilities; XML formatting; Service-Oriented Architecture testing, including the configuration, reconfiguration, composition, and re-composition capabilities. Configuration and reconfiguration deal with faults and failures that prevent a part of WS from working correctly, while composition and re-composition deal with changing environments and evolving new functionalities into the existing WS.

Current WS testing techniques focus on the composition and the interoperability among WS, including functionality testing, interface (WSDL file) testing, SOAP message testing, stress testing, and performance testing. A common solution is the model-based approach [15][17][19]. In [13] the verification problem for WS specifications is studied, aiming at guiding the construction of composite WS to guarantee desired properties, such as deadlock avoidance, bounds on resource usage, and response times. Several formal languages have been developed for modeling and verification. BPEL [2], which combines WSFL and

WSCI with Microsoft's XLANG specification, accompanied with WS-Coordination protocol and WS-Transaction protocol is one of the widely used languages. Web Ontology Language (OWL) and OWL-Services (OWL-S) are widely used in specifying and developing WS. There are other formal methods. Process Algebra and Petri Nets [14] have been used in verifying software correctness. They could be applied in verifying WS. However, it is more difficult because WS are often associated with business and enterprise environments that involve non-numerical transactions. None of these approaches provides good scalability for WS testing, nor they support automatic composition [20] or provide testing mechanism or matrix to choose the most qualified WS from multiple candidate WS that claim to perform the same function.

WS testing tools are developed to enable automated testing. Most tools support testing a single service by simulating thousands of clients accessing the service simultaneously. Such examples includes SilkPerformer [25] by Segue Software, ANTS [24] by Red Gate Software, e-Test Suite (FirstAct2.0 previously) [11] by Empirix, Astra LoadTest [18] by Mercury Interactive, DevPartner Studio [8] by Compuware, MQTester [7] by Rational Software etc. Xmlspy 2005 [1] by Altova focuses on SOAP messaging testing. SOAPtest [23] by Parasoft can perform functional testing, stress testing, security testing, and interoperability testing by simulating both client side and server side.

Some WS testing tools are also development tools that are often platform dependent, e.g. DevPartner Studio is a development suite for Microsoft .NET platform-based WS application, while MQTester [7] is a test studio plug-in for IBM's WebSphere MQ system. For interoperability testing, WS-I provides a testing tool to ensure the WS under development conform to the Basic Profile 1.1 Compliance [3].

3.2 Specification-based test case generation

Because the source code of existing WS may not be available, it is important to be able to generate test cases from WS specifications only. WS specifications often contain conditions and constraints the WS operate, which can be used for test case generation. Figure 2 shows the overall process of our specification-based test case generation process. Assume the given WS specifications are written in OWL-S. The specifications written in other specification languages such as WSDL, and WS-CDL may be translated into OWL-S first. OWL-S facilitates WS specification with a core set of markup language constructs for describing the properties and capabilities of WS in an unambiguous and computer-interpretable form. OWL-S supports the automation of various tasks including automated WS discovery, invocation, interoperation, composition, and execution monitoring. The first step is to perform specification V&V. We developed three different specification V&V techniques: Completeness and Consistency (C&C) analysis [30], model checking technique based on BLAST [4][9], and verification patterns [31]. If the specification fails the check, i.e., it does not meet the published specification, the service fails the test.

Once the specification passes the test, Boolean expression analysis method is used to extract the full scenario coverage of Boolean expressions [30], which are then applied as the input the Swiss Cheese Automated Test Case Generation Tool [29][32], which, in turn, generates both positive and negative test cases. Positive test cases are used to test if the WS output meets the specification for the legitimate inputs, while negative test cases are used to test the robustness, i.e., the behavior of the WS if unexpected inputs are applied. Negative test is particularly important for WS because the source code may not be available. Negative testing will be able to verify that a service does not add unspecified features into the code. Failing to perform negative testing may compromise the security and safety of the other WS that communicate with this WS.

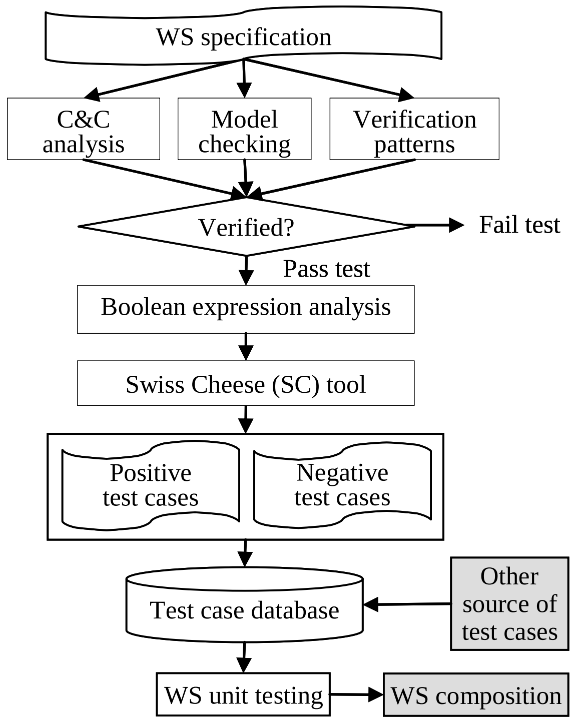

Figure 2. WS unit testing

Finally, the test cases are stored in the test case database before actual application. The key technologies applied in WS unit test include:

- Completeness and consistency (C&C) analysis of the WS specification: This checks whether all conditions in OWL-S specification are consistent and whether all conditions have been covered and handled properly by the specification.
- Model checking: Model checking has been proposed recently to facilitate software testing following the idea that model checking verifies the model while testing validates the correspondence between the model and the system. One of the most promising approaches was proposed at University of California at Berkeley using BLAST [4][9]. The BLAST model checker is capable to check safety temporal properties, predicate-bound properties (in the form to assert that at a location l, a predicate p is true or false), and identify dead code. BLAST abstracts each execution path as a set of predicates (or conditions) and then these predicates are used to generate test cases to verify programs. This approach is attractive because it deals with code directly rather than the state model in traditional model checking [6][16]. Thus, the BLAST approach is better suited for software verification than traditional model checking. However, this approach is not directly applicable to WS verification because WS providers may not provide the source code. Furthermore, the test cases generated in the BLAST approach are targeted mainly on the positive aspects of testing. Negative aspects such as near misses are not handled. In our approach, we extend the BLAST approach to adapt to WS testing in the following ways: (1) Instead of using the source code to drive model checking, we use the OWL-S for model checking. The control flow automata used by BLAST resembles the workflow model derived by the control constructs in the process ontology of OWL-S. (2) We rely on the conditional or unconditional output, effect, and precondition of each atomic / primitive WS to construct their essential inner control logic.
- Test case generation: Test case generation techniques can be greatly enhanced by this comprehensive formal C&C analysis followed by test case generation based on Boolean expressions [30]. An important distinction of this approach is that test case generation is based on topological structure of Boolean expressions and quantitative Hamming Distance (HD). Previous approaches including MC/DC and MUMCUT [5] explores the Boolean expressions but did not consider the topological structure. Exploring the topological structure of Boolean expressions can easily reveal the faults not discoverable by previous approaches. Furthermore, these test case generation mechanisms can be automated and thus saving significant effort and time. Both positive and negative test cases can be generated. Because the topological structure of Boolean expressions may be similar to the topological structure of Swiss Cheese, this test case generation is called Swiss Cheese (SC) approach. The SC approach identifies those (positive and negative) test cases that are most likely to fail in the source code. The SC approach first maps the Boolean expressions into a multi-dimension Karnaugh map called polyhedron. It then iteratively identifies all boundary cells of the polyhedron and selects most fault-sensitive test cases among all boundary cells. The more neighboring negative test cases (degree of vertex -- DoV) a boundary cell has, the more error-sensitive it is. The last step is post-checking, trying to identify critical negative test cases within the polyhedron. For each negative test case, HD is used to define the minimum different Boolean digits between it and any boundary cells. The HD of all boundary cells is 0, while the one next to it has HD of 1, and so on. A positive HD means that the cell is outside of the Boolean expression specified, and a negative HD means that the cell is inside the Boolean expression specified. By selecting cells at corners and major intersections, one can select the most potent test cases including positive and negative test cases.

3.3 Collaborative testing

WS offers a more competitive market than the traditional software market. All countries can easily provide their software products as services on the web. It is even more competitive than "traditional" e-commerce because the decision may be now made at runtime rather than at "think" time. Decisions are based on testing results in addition to other criteria such as brand and profiles. Only the best services can survive in such a competitive environment.

Traditional software is tested using Independent V&V (IV&V), in which testing is done by an independent team different from the development team. This is a good practice to avoid common mode errors. However it is not sufficient for WS testing. A WS broker or provider can compose a new service using available WS from different providers without knowing the implementation details. WS need to be tested collaboratively by clients, WS brokers, WS providers, and other independent organizations such as research institutions, standard organizations, and regulators. Since the collaborators in testing also include the competing WS providers, it is necessary to balance the WS providers' business secrets (implementation details) and trustworthiness of the composite WS. Thus, WS testing needs a new model: Collaborative V&V (CV&V). Under this model, the WS providers must still perform IV&V during the development of WS, but when a service registers at a WS broker, the WS provider must provide a set of sample test cases. Then all the parties including clients, WS brokers, and WS providers collaborate to perform CV&V. Table 1 summarizes the similarities and differences between traditional IV&V and CV&V for WS. As can be seen, traditional IV&V is often performed statically and off-line. Although traditional V&V techniques can be applied to WS, a significant portion of WS testing must be done dynamically at runtime. Such dynamic and just-in-time nature challenges the traditional IV&V theory and practice.

This CV&V can be centrally performed by a neural 3rd-party. All WS participants including clients, providers and brokers can submit WS, WS specifications, test cases, model checker to this neutral party, and this neutral party can perform various test and evaluation on submitted WS using submitted test cases and model checkers. Figure 3 shows a potential design for this neutral party.

Table 1. IV&V versus CV&V

	Traditional IV&V	Service-Oriented CV&V
Approach	The test team is independent of the development team to ensure objectivity and completeness. Test is done by software providers.	Test by collaboration among WS providers, clients, and independent WS brokers. The emphases are on real time and just-in-time testing, and evaluation using data dynamically collected at runtime.
Testing location	Centralized multi-phase testing.	Distributed, remote, multi-agent and multi-phase testing.
Operational testing	Off-line field testing or simulation.	On-line just-in-time testing in application environment.
Regression testing	Off-line regression testing.	On-line regression testing using data dynamically collected.
Integration testing	Static configurations and systems must be linked before integration testing	Dynamic configuration and systems are linked at runtime and verified at runtime
Testing coverage	Input domain, structural (white-box), or functional (black-box) coverage.	WS providers can have traditional coverage, brokers and clients may have black-box (e.g., WSDL) coverage only.
Test case profiling	Static profiling	Dynamic profiling with data collected by distributed agents.
Reliability modeling	Input domain-based and reliability growth models	Reliability models based on dynamic profiles and group testing
Certification	Static certification center.	Dynamic certification based on service history.
Test case repository	Statically maintained repository.	Dynamically expanding repository.
Model Checking	Model checking on the code or state model.	Just-in-time dynamic model checking on the specification and WSDL/DAML-S/OWL-S.
Simulation	Specification and models can be simulated to verify the design and code	Just-in-time on-the-fly simulation for composite WS.

3.4 Group testing

A group testing technique, originally developed for testing a large number of blood samples and later for software regression testing [10], is an attractive solution to address the problem of testing large number of available WS.

Assume the new WS to be composed consists of n component WS: WS_1, WS_2, ..., WS_n. For component WS_i, there are k_i alternative WS available. Thus, there are totally $k_1 * k_2 * ... * k_n$ different possibilities to compose the new WS. Since k_i is a large number it is obviously not efficient to test all the combinations. A group testing technique is proposed to test a large number of WS efficiently [27]. The main idea is to test the large number of WS in two phases. In phase 1, following steps are executed:

1. Select a subset of WS randomly from the set of all WS to be tested.

2. Group testing: Apply each test case to all the WS in the selected WS simultaneously.
3. Voting: For each test input, the outputs from the WS under test are voted by a stochastic voting mechanism based on majority and deviation voting principles.
4. Failure detection: Compare the majority output with the individual outputs. A disagreement indicates a component failure.
5. Oracle establishment: If a clear majority output is found, the output can be used to form the oracle of the test case that generates the output.
6. Test case ranking: Test cases will be ranked according to their fault detection capacity, which is proportion to the number failures a test case detects. In the phase 2, the higher ranked test cases will be applied first to eliminate the WS that failed to pass the test.
7. WS ranking: The best WS found will be selected for new WS composition.

In phase 2, the remaining WS will be tested and a shortlist of best WS will be generated using the oracles and the most powerful test cases found in phase 1, so that the overall testing time can be reduced. In phase 2, following steps will be executed:

8. Apply the most powerful test case that has not been used in phase 2 to group test the remaining WS;
9. Update the oracle if necessary;
10. Update WS ranking if necessary;
11. Update the test case ranking;
12. Eliminate the WS that cannot enter the shortlist;
13. return to step 8 if there are still unused test cases;
14. Output the shortlist of the best WS.

The two-phase group testing technique is name ASTRAR for Adaptive Service Testing and Ranking with Automated oracle generation and test case Ranking. Extensive experiments have been conducted to verify the effectiveness of the group testing techniques. The data show that this approach save significant time and effort, and at the same time achieving the same test results as compare to exhaustive testing.

4. Full service-oriented operating systems and the challenges

Current computer system architecture for both software and hardware resembles an onion in which each layer provides a set of functions depending only on the immediate layer (ring) within it. The CPU and memory of a computer system form the innermost ring or the core of the system. The input/output devices form a ring surrounding the core. For software, the innermost ring is the Operating System (OS) kernel. The next ring is the OS device mangers and drivers. A layer of middleware (or agent) that supports specific types of applications could form a ring outside the manager and driver ring. The outermost layer of an OS is the shell or a window that interfaces the OS functions to the programmers. The application layer forms another ring surrounding the OS. The advantages of such an onion architecture are:

- Each component communicates with a limited number of components only and thus the interface is relatively simple.
- V&V of the OS design are relatively easy because of the hierarchical structure and limited interactions.
- A new system can be developed based on or from the layer needed thus saving development cycle.

However, the current architecture does not support the important features offered by the SOA architecture, including

- Interoperability: Components in SOA are services and are implemented strictly following interfacing standards so that each component can easily communicate with other components with the same interface.
- Dynamic re-composition: The overall function of the system can be recomposed by adding new components, removing, replacing, and re-organizing existing components. Re-composition is more generic than reconfiguration, where failed component can be replaced by a backup component.
- Searching and remote invocation: Due to the standard interface, an SOA system can search, find, and remotely invoke components to perform required services. In other words, the interoperability applies not only to the components within a system. It also applies to the components among heterogeneous systems, which ultimately solves the data representation and interpretation problems among different systems.

Current implementation of SOA is only applied at the application layer or in a middleware layer which is created to support service-oriented computing. The rest of the system, i.e., the operating system, networks, and hardware components are still implemented in the traditional way. Thus, the benefits of SOA are applicable at the application layer only.

In this paper we propose to apply SOA to the entire system, including the inner layers, that is, to design a Service-Oriented Operating System (SOOS). Figure 3 shows the structures of a traditional OS and an SOOA. The former has an onion of layers while the latter has only one layer: all services are equally placed in the same layer.

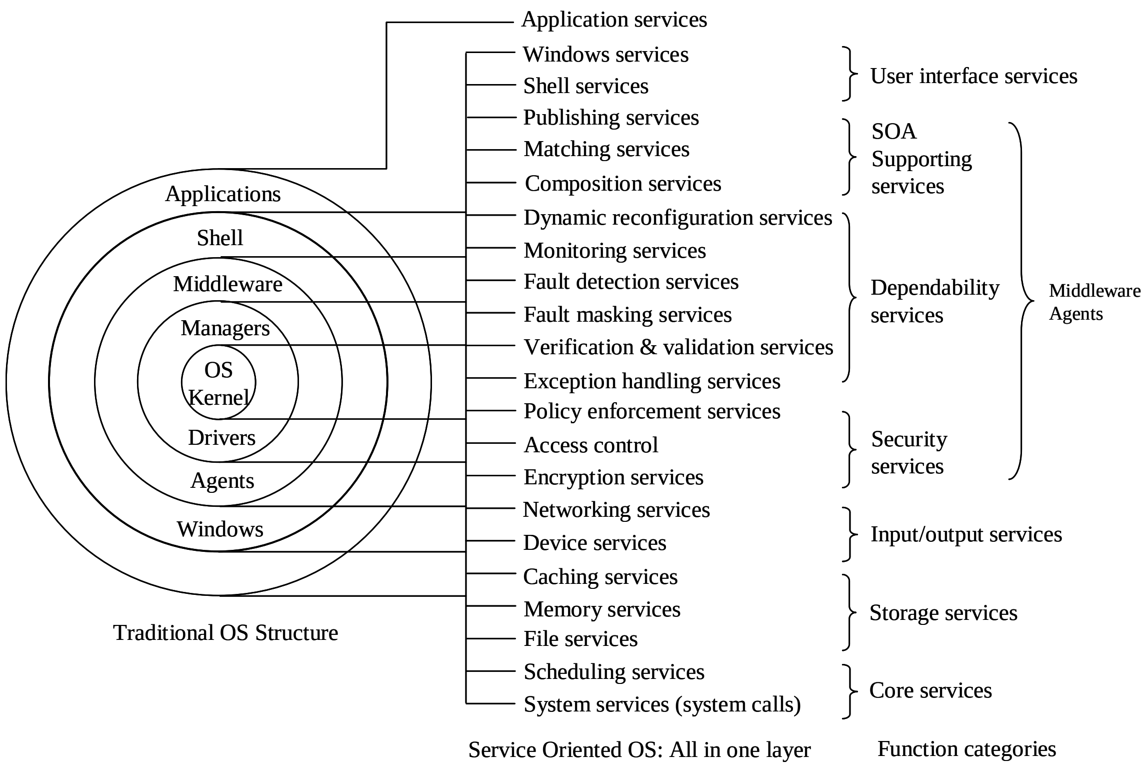

Figure 3. A full service-oriented operating system architecture

What is the significant difference between an SOOS and a traditional OS? The most significant difference is the interoperability among heterogeneous systems and the compatibility of the applications on these systems. We will no longer have Windows applications, Unix applications, and Mac applications. All applications will be compatible on the SOOS. Another major benefit is that all the components on OS are now services that can be added, removed, and replaced, including the system calls and the scheduler in OS kernel. This feature will allow the OS of a computer system to be upgraded and gradually replaced without stopping the operations or causing incompatible problems for the existing applications. The current practice is, every a few years a new OS version is released and all computer users will have to stop all applications, install the new OS, restart the computer, and cross their fingers to wish all the existing applications to work on the new OS version.

What are the major challenges to the SOOA design? V&V of SOOA will be much more challenging because of the all-in-one layer architecture with its composability and interoperability. The service-oriented testing techniques presented in this paper, the specification-based testing, dynamic model checking, CV&V, and group testing, can be applied to test SOOA. However, due to the complexity and the open platform design of such a system, more research is needed before the SOOA can gain the level of trustworthiness that a traditional OS possesses. We urge the research community to pay attention to the V&V issues while researching for the functionality of full SOA systems.

5. Summary and conclusion

Current WS testing techniques assume that unit testing has been adequately performed by the WS providers. This assumption does not hold if trustworthy services need to be composed based on the searched and found WS over the Internet. Limiting the sources of WS providers breaches the idea of the open platform on which WS are based upon. This paper examined the importance of dynamic WS unit test, which is significantly different from software unit testing due to the unavailability of source code and the runtime feature. Then this paper proposed three techniques to perform unit testing: specification-based test case generation, collaborative testing, and group testing, which can enforce the trustworthiness of the WS components before they are integrated into the new composite WS.

Finally, this paper outlined the layout of a full SOA-based OS, or SOOS, in which all components from the kernel of a traditional OS through the application layer are implemented as services. Such an architecture will allow the OS to be upgraded and replaced without stopping the current operations, introduce full interoperability among heterogeneous systems, and ultimately create platform (OS) dependent applications. The challenges to implement full SOA-based OS are the development of adequate V&V techniques to ensure the trustworthiness of the SOA-based OS and the computer system.

References

[1] Altova, "Altova XMLSpy® 2005", http://www.altova.com/products_ide.html

[2] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana, "Specification: Business Process Execution Language for Web Services Version 1.1", http://www-128.ibm.com/developerworks/library/ws-bpel/

[3] Keith Ballinger, David Ehnebuske, Christopher Ferris, Martin Gudgin, Canyang Kevin Liu, Mark Nottingham, Prasad Yendluri, "Basic Profile Version 1.1", http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[4] D. Beyer, A. Chlipala, T. Henzinger, R. Jhala, and R. Majumdar, "Generating Tests from Counterexamples", Proceedings of the 26th International Conference on Software Engineering (ICSE'04), Scotland, UK, May 2004, pp. 326 – 335.

[5] T. Y. Chen and M. F. Lau, "Test Cases Selection Strategies Based on Boolean Specifications", Software Testing, Verification and Reliability, Vol. 11, No. 3, Sep. 2001, pp.165-180.

[6] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*, MIT Press, 2002.

[7] CommerceQuest, "CQ MQTester", http://www.commercequest.com/products_utilities_mqtester.asp#

[8] Compuware, "DevPartner Studio Professional Edition", http://www.compuware.com/products/devpartner/studio.htm

[9] N. Davidson, *Testing Web Services*, at http://www.webservices.org/, in October 2002.

[10] D. Z. Du and F. Hwang, *Combinatorial Group Testing And Its Applications*, World Scientific, 2nd edition, 2000.

[11] Empirix, "e-Test suite", http://www.empirix.com/www/resources/media/pdf/brochures/br_eTESTsuite.pdf

[12] R. Fikes, A. Farquhar, "Distributed Repositories of Highly Expressive Reusable Ontologies", IEEE Intelligent Systems, 14(2): 74-79, 1999.

[13] X. Fu, T. Bultan, and J. Su, "Formal Verification of E-Services and Workflows," Proc. Workshop on Web Services,E-Business, and the Semantic Web (WES), LNCS 2512, Springer-Verlag, 2002, pp. 188–202.

[14] R. Hamadi and B. Benatallah, "A Petri-Net-Based Model for Web Service Composition," Proc. 14th Australasian Database Conf. Database Technologies, ACM Press, 2003, pp. 191–200.

[15] Howard Foster, Sebastian Uchitel, Jeff Magee, Jeff Kramer, "Model-based Verification of Web Service Compositions", Proceeding of the 18th IEEE International Conference on Automated Software Engineering (ASE'03)

[16] G. Holtzman, "The Spin Model Checker," IEEE Transactions on Software Engineering, Vol. 23, No. 5, May 1997, pp. 279-295.

[17] Daniel A. Menascé, "Composing Web Services: A QoS View", IEEE Internet Computing, IEEE Computer Society, November-December 2004.

[18] Mercury Interactive, "Mercury Interactive Astra LoadTest",http://www.b2net.co.uk/mercuryinteractive/mercury_interactive_astra_loadtest.htm

[19] Sin Nakajima, "Verification of Web Services Flow with Model-Checking Techniques", Proceedings of the First International Symposium on Cyber Worlds (CW'02), November 2002, pp. 378 - 385.

[20] S. Nara Narayanan and S.McIlraith, "Simulation, Verification and Automated Composition of Web Services," Proc. Int'l World Wide Web Conf. (WWW2002), 2002, pp. 77–88.

[21] B. Sleeper, "The five missing pieces of SOA", www.infoworld.com/article/04/09/10/37FEwebservmiddle_1.html, September 2004.

[22] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout, "Enabling Technology For Knowledge Sharing", AI Magazine, Volume 12, No. 3, Fall, 1991.

[23] Parasoft, "SOAPTest Data Sheet", http://www.parasoft.com/jsp/products/quick_facts.jsp?product=SOAP

[24] Red-gate, "Load Testing for .NET Developers and Testers", http://www.red-gate.com/dotnet/load_testing.htm

[25] Segue,"SilkPerformer", http://www.segue.com/pdf/silkperformer.pdf

[26] W. T. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao, "Verification of Web Services Using an Enhanced UDDI Server", Proc. of IEEE WORDS, 2003, pp. 131-138.

[27] W. T. Tsai, Y. Chen, R. Paul N. Liao, and H. Huang, "Cooperative and Group Testing in Verification of Dynamic Composite Web Services", in Workshop on Quality Assurance and Testing of Web-Based Applications, in conjunction with COMPSAC, September 2004, pp.170-173.

[28] W. T. Tsai, D. Zhang, Y. Chen, H. Huang, R. Paul, N. Liao, "A software reliability model for Web Services", 8th IASTED International Conference on Software Engineering and Applications, Cambridge, MA, November 2004, 144-149.

[29] W. T. Tsai, X. Wei, Y. Chen, B. Xiao, R. Paul, and H. Huang, "Developing and Assuring Trustworthy Web Services", 7th International Symposium on Autonomous Decentralized Systems (ISADS), April 2005, pp. 91-98.

[30] W.T. Tsai, Lian Yu, Feng Zhu and Ray J. Paul, "Rapid Verification of Embedded Systems Using Patterns", COMPSAC 2003: 466-471.

[31] W. T. Tsai, R. Paul, L. Yu, X . Wei, and F. Zhu, "Rapid Pattern-Oriented Scenario-Based Testing for Embedded Systems", to appear in book *Software Evolution with UML and XML*, edited by H. Yang, 2004.

[32] W. T. Tsai, X. Wei, L. Yu, R. Paul, and H. Huang, "Condition-Event Combination Covering Analysis for High-Assurance System Requirements", to appear in *Computer Systems Science & Engineering (CSSE) journal.*