A Privacy-Aware, Decentralized, End-to-End, CFG-based Regression Test Selection Framework for Web Services using only Local Information

Michael Ruth, Curtis Rayford, Jr.
Computer Science Department
Roosevelt University
Chicago, IL
USA
mruth@roosevelt.edu, crayford@mail.roosevelt.edu

ABSTRACT: *Web services are composable, interoperable, and autonomous which means that a single web service interaction could involve services written in several different languages provided by several different service providers. Such interactions hamper the development of RTS techniques because RTS techniques generally require some form of implementation details which service providers in separate autonomous systems are unlikely to expose. In this work, a privacy-aware, end-to-end regression testing framework employing a RTS technique using Control-Flow Graphs (CFGs) built using only locally available information at each service and a publish/subscribe mechanism will be presented. The proposed framework is unique because it does not require service providers to expose private and sensitive implementation details in order to participate in the regression testing framework. Also, a case study will be presented to highlight the approach and an empirical study is performed to evaluate the cost-effectiveness of the approach.*

Keywords: Web Services, Regression Test Selection, Regression Testing Frameworks, Control-Flow Graphs, End-to-End; Empirical Study

Received: 1 March 2011, Revised 18 April 2011, Accepted 24 April 2011

1. Introduction

Web services have become the de-facto standard for modeling inter- and intra-enterprise business processes since they have enabled business workflows to be extended beyond the boundaries of companies and organizations. These business workflows may be composed of several different Web services and may flow in or out of the enterprise. Since the business world often involves very rapid change to accommodate current market conditions, business processes inevitably require frequent adjustments along with their supporting Web services. These rapid adjustments, or modifications, must be supported by rapid verification to ensure some level of quality assurance. Every time an element within a system is modified, the system must ensure that the modification does not have an adverse effect on any other unmodified areas of the system. Typically, this is performed by running the same test cases on the various elements of the system before and after the modification to determine whether or not it had an adverse effect on the system, especially unmodified areas. This "retesting" before and after modifications is called regression testing.

One of the problems with regression testing in large systems is the large number of test cases that need to be executed repeatedly to ensure that the system is not made worse by each new modification and the key to solving this problem is regression test selection (RTS). The goal of RTS techniques is to reduce the number of tests which need to be executed upon each new

modification while still providing some level of confidence that the system was adequately tested. Although several RTS techniques have proven to be effective for traditional software applications [1], they suffer from limited applicability to Web service-based systems due to the implementation dependency of the RTS techniques themselves. The implementation dependency of the RTS techniques for traditional applications poses several problems for developers of RTS techniques for Web services, specifically dealing with the features that made Web services so popular, namely their composability, interoperability, and autonomy. Since Web services can be composed of other Web services, RTS techniques for Web services must support both atomic services, which are services not composed of other services, and composite services, which are services which are composed of other services (which could also be composite services).Additionally, RTS techniques for Web services must support the interoperability of Web services sinceWeb services are language independent implying that a single interaction could involve services written in several different languages. Finally, due to the inherent autonomy of Web service-based systems a service may be composed of services provided by several different service providers. Since RTS techniques often involve implementation details, developing an end-to-end RTS technique or Web services involves the sharing of these details, even across autonomous system borders. Since service providers in separate autonomous systems are unlikely to share implementation details with each other, this is a major concern for service providers since they need to maintain the privacy of those details. These features imply that any RTS technique designed for Web services must support interoperability and composability even across autonomous systems.

In general, service providers would never participate in a framework which performs a RTS technique that requires sharing sensitive information across autonomous borders due to the need to protect sensitive implementation details. Since service specifications, which are normally available, provide only interface specifications, most RTS techniques for Web services require the specifications be augmented with additional information, such as contractual information [2], to support the RTS selection process. In general, contractual information is considered specification and therefore not sensitive. For example, suppose a service were augmented with its effect, or obligation, of ordering a book. This additional information provides no sensitive implementation details and therefore would never be considered sensitive. However, if that additional information includes service dependencies it should be considered sensitive. For instance, using the same example, if the same service were augmented with its effect of ordering a book through a certain bookseller, this should be considered sensitive as it may present real problems for the service provider. If a competitor knows which bookseller the service uses, then the competitor may simply provide the same service using the same bookseller or advertise that their service is better since it uses a different bookseller. In either case, this exposure couldmean a loss of revenue for the service provider. In order to develop a RTS technique that service providers would participate in, the information which crosses autonomous boundaries must be carefully considered and necessarily limited.

In this paper, we propose a decentralized regression testing framework which employs a CFG-based RTS technique designed to performend-to-end regression testing for both intra- and inter-enterprise Web services, which is privacy aware as participation in the framework does not require service providers to expose sensitive implementation details. CFGs are an ideal medium for Web services as they address the interoperability and composability concerns which arise in frameworks designed for Web services. Normally, in an end-to-end CFG-based technique, the service providers participating in an interaction would normally share information to build information required for such a technique, namely CFGs, test cases, and coverage information, for their services, but service providers would never participate in such a technique because CFGs are implementation details and therefore very sensitive. This technique will therefore only use local information to build the three required elements of CFG-based techniques need, namely CFGs, test cases, and coverage information mapping the test cases to paths in the CFG. For atomic services which do not depend on other services, the CFG will be complete, but for composite services which depend on other services to perform their work, the framework will build only partial, or local, CFGs and perform all necessary testing activities at that service using only their partial CFGs. This is necessary as no service provider participating in the framework will ever need to share any sensitive implementation details such as CFGs. A publish/subscribe notification system is employed to ensure that the RTS and regression testing processes are performed in an end-to-end manner at each service in every service interaction they participate in. The main contribution of this work is the decentralized, privacy-aware, regression testing framework which employs a CFG-based RTS technique designed to perform end-to-end regression testing for both intra- and inter-enterprise Web services. Additionally, a case study and empirical study will be presented to highlight the application of the proposed approach and provide some early measure of effectiveness for at least the systems studied.

The remainder of this paper is organized as follows. Section 2 will present background information for Web services and regression testing. Section 3 will discuss the regression testing framework and the CFG-based RTS technique. The illustrative case study is presented in section 4, section 5 introduces and discusses the empirical study, related work will be discussed in section 6, and section 7 concludes.

2. Background

In this section, an introduction to Web services, along with background information on regression testing techniques including safe regression test selection techniques, will be provided in detail.

2.1 Web Services

Broadly, Web services refer to self-contained web applications that are loosely coupled, distributed, capable of performing business activities, and possessing the ability to engage other web applications in order to complete higher-order business transactions, all programmatically accessible through standard internet protocols, such as HTTP. More specifically, Web services are Web based applications built using a stack of emerging standards based on XML, namely SOAP, WSDL, and UDDI [3].

2.2 Regression Testing and RTS Techniques

Regression testing is the process of testing modified software to provide confidence that the unchanged parts have not been adversely affected by the modification and RTS techniques attempt to reduce the cost by selecting and executing only a subset of tests. Most regression test selection techniques use some information about the system under test (SUT) to determine the location of the modification in the system occurred in a process called impact analysis. The technique adopted for our approach uses CFGs, which are extracted from the underlying source code [1]. After initialization, when a modification occurs the technique follows three basic steps: 1) It Constructs a new CFG for the modified system. 2) It identifies the impact of the modification by comparing the two CFGs using a dual traversal. The algorithm performs an equality comparison to determine if an individual node, which contained the source code for that node, has been modified. The result of the comparison is a dangerous edge list, which is a list of nodes which may behave differently under at least one test case due to a difference between the original and modified system. 3) The algorithm uses the dangerous edge list and the coverage information to select all test cases which correlate to a dangerous edge. The technique provides a guarantee that any test case which does not cover a dangerous node will behave identically if run on both the original and the modified system, and thus cannot expose a new fault, which implies that the test case can be safely left unselected.

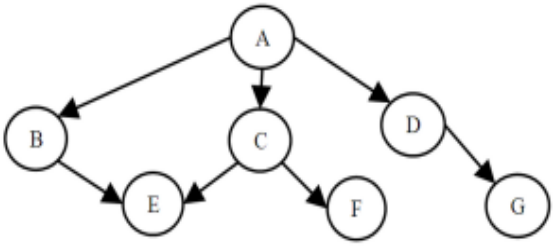

Figure 1. Example of a CFG of a software entity

For example, suppose that a program has the CFG shown in Figure 1 and that the following test cases exist: 1 covers the path A-B-E, 2 covers the path A-C-E, 3 covers the path A-C-F, and 4 covers the path A-D-G. If the code corresponding to node E was modified, the algorithm will producea resulting dangerous edge list that contains paths A-B-E and A-C-E. Using the coverage information, this implies that only test cases 1 and 2 need to be rerun. However, if the code corresponding to node C is modified, then the dangerous edge set will be A-C. Then test cases 2 and 3 will be selected to be rerun.

In [4],Rothermel and Harrold developed a cost model to determine if a RTS technique could be considered cost-effective, or beneficial, for a system which states: In order for a RTS technique to be considered cost-effective, the costs of performing the technique and executing only the selected tests must be less expensive than the costs of running all tests without performing a selection step. In the section outlining the empirical study, this cost model will be used to to provide a measure of its effectiveness for at least the systems under study.

3. Regression Test Selection Technique

In this section, the RTS technique that adapts the three step RTS technique outlined in section 2 will be presented. Since Web services can be modified independently and concurrently, the RT and RTS technique is decentralized and every service will

have its own agentwhich performs the RTS technique and the regression testing for that service automatically. The agents communicate, primarily to inform interested parties of modifications, through a publish/subscribe notification system. Together, the decentralized agents and the notification system ensure that after a modification occurs anywhere within the system all services which could possibly be affected perform RTS and regression testing at their endpoint to determine if their service was negatively impacted by the modification. The remainder of this section will present in natural order, the construction phase, which is designed to initialize the system, followed by the operational phase, which is designed to be executed upon notification of a modification.

3.1 Construction Phase

In order for the agents to participate in the framework, the construction phase must initialize the framework for each participating service by constructing the three elements required by the framework, namely CFGs, test cases, and coverage information mapping the test cases to the CFG. First, the CFG must be constructed and the process for performing this task depends on the nature of the service (atomic or composite) but is similar to the process of generating CFGs for monolithic applications since all requisite information is available for all services. For atomic services, generating the CFG is identical to the process of generating it for a monolithic application because all necessary code is present and therefore the result is a complete CFG. However, only partial CFGs can be generated for composite services since only partial information is available. Partial CFGs are those which contain "call" nodes, which are placeholder nodes in the CFG which represent a composed service. This is necessary since at the composite service level, the CFG of the composed service is unavailable.In order to create these "call" nodes, which contain the name of the operation and the URI of the service, the scanning of the composite service code must be able to recognize calling patterns. A calling pattern is a well-defined pattern used to call other services which are generally employed by frameworks, such as Apache Axis, to generalize the "glue" code created by the framework. In the process of recognizing the calling pattern and constructing these "call" nodes for the composed services, the agent registers with the agent of the composed service for later modification notifications. This registration process is the means by which agents inform composed services that they are "interested" in their modifications, so that when a composed service is modified, the service will perform regression testing and RTS processes at the composite level to determine if the composite service is impacted by the modification. The "call" nodes and registration process are important components of the proposed RT and RTS framework and will be discussed at various times throughout all phases of the approach. Once the CFG is constructed, the construction of the other two elements, namely test cases and coverage information, can proceed.

The next step in the construction phase involves generating, or creating, a test suite that effectively tests the services involved and this step may requirehuman intervention depending on whether the service is an atomic or a composite service. The process for generating test suites for atomic services is identical to generating test suites for traditional applications because all requisite information is available to the tester, but the process for generating test suites for composite services is less trivial since composed servicesare black boxes to the composite service. If the composed service exposes a test suite, the testers must obtain these test cases, but if not a number of test case generation tools exist [5]. In either case, these test cases are for the composed service and need to be made valid at the point of entry for the composite service which requires human intervention. In order to make these test cases valid at the point of entry for the composite service, the testers must determine what input must be delivered to the composite service so that at the point of entry of the composed service, the "call" node, the input from the test case is presented to the composed service.Once the test cases are generated, the construction of the final component, coverage information, may begin.

The final step in the construction phase is generating the coverage information which maps either the CFG for atomic services or the partial CFG for composite services to the generated test cases and is performed in much the same way that traditional applications generate the mapping of the test cases to the CFG. The service is augmented with monitors to determine which path each test case covers as it is being executed and the path each test case covers through the CFG is recorded. Once this construction process is complete for each of the participating agents, the agents have constructed all necessary components of the technique and are ready for the operational phase to begin.

3.2 Operational Phase

The operational phase begins when an agent is being notified of a modification and is designed to ensure that the RTS and regression testing processes are performed at each and every service which could have been impacted by the modification. The operational phase is automatic for services receiving modification notifications, but service providers need to start the process by modifying their service and then notifying the corresponding agent of the modification. After being notified that a local modification occurred, the service notifies all interested parties that a modification occurred using the publish/

subscribe system which then starts their RTS and regression testing processes and then the service begins its own RTS and regression testing processes. This ensures that every service that could have possibly been affected by the modification performs their RTS and regression testing processes which is essential for end-to-end testing.

After notification of interested parties, the RTS process begins by rebuilding the modified CFG which depends on the locality of the modification. If the modification occurred locally, the agent rebuilds the CFG exactly as described in the construction phase. However, if the modification occurred in a composed service, the agent rebuilds the partial CFG by copying the partial CFG and marking the composed service's "call" node as modified. This ensures that upon performing the RTS technique, all tests which pass through the "call" node are selected and executed which ensures that any modification which occurs in the composed service will be fully tested at this service. Once the modified CFG is generated, the process of the rest of this approach is identical to the approach outlined in the background in section 2. The original CFG is compared to the modified CFG to identify a set of dangerous edges. Once the dangerous edge list is computed, the tests which need to be rerun can be determined from using the test coverage information and the dangerous edge list, and executed.

3.3 Limitations of the RTS technique for Web Services

There are several important limitations to the technique which must be addressed. The technique is limited to acyclic service interactions. This is a minor limitation since services should model complete business functions in a SOA and are unlikely to interact in a loop. Finally, this approach is limited to static service interactions to ensure test case determinism.

4. Case Study: Purchase Order System

In this section, the proposed approach is applied to a purchase order processing system. The system was chosen as it is used in a variety of case studies [6], the system is very intuitive and indicative of real-world service interactions, and can be shown in full detail. This purchase order system is comprised of 4 services including: 1) S1, which is a composite service that accepts all orders for a given company. 2) S2, which is a hardware provider that accepts orders for hardware products. 3) S3, which is a software provider that accepts orders for software. 4) S4, which is an office supply provider that accepts orders for office supplies. Service S1 is composed of the other three services: S2, S3, and S4. The application of the approach, applied to a purchase order processing system, will be discussed in natural order: the construction phase followed by the operational phase.

In the construction phase, the three components (CFGs, coverage information, & test cases) of the framework must be constructed for each service. The agents corresponding to the three atomic services (S2, S3, and S4) perform this process as discussed in the construction phase section. First they construct their CFGs, augment them with test cases, and map those test cases to their CFGs, thus providing the coverage information. Their CFGs are complete since as atomic services they have the all of the requisite code available to them. The complete CFGs of S2, S3, and S4 are shown in Figure 2.

The agent corresponding to service S1 (the lone composite service) constructs a partial CFG which contains "call" nodes. The agent for service S1 can only develop a partial CFG since it only uses information locally available to it. The partial CFG is shown in Figure 2. During the construction of its partial CFG, when the agent constructs "call" nodes it contacts the agents corresponding to each "call" node found (in this case S2, S3, and S4) to register for updates and determine if test cases are available for the service. If test cases are available, the testers must query for the test cases and if not, the testers must generate test cases for each atomic service. In either case, the testers must create test cases for S1 which correspond to test cases for the composed service. For instance, suppose S3 provides the testers of S1 with a test case. The testers must determine a way for the test case to be called through S1 such that when the service S1 calls S3, the call uses the test case information provided by S3.Once finished, the agent augments the service with monitors and executes the test cases to record the paths each test case covers. Once this process is complete, the construction phase is complete and the operational phase can begin.

In the operational phase, the agents which correspond to each of the services wait for modification notifications. For illustrative purposes, suppose that the service provider responsible for S3 modifies the code corresponding to node 12 (shown in figure 2). The agent corresponding to service S3 would notify S1 of the modification and then rebuild its CFG with a modified node 12. After rebuilding its CFG, the agent corresponding to service S3 will begin the RTS and regression testing processes (selecting all test cases which cover node 12 and executing them). At the agent corresponding to service S1, once it receives the notification from the agent corresponding to service S3, it copies its partial CFG and then marks the "call" node

corresponding to service S3 to create its modified CFG. After rebuilding its CFG, the agent corresponding to service S1 will then begin the RTS and regression testing processes (selecting all test cases which cover the "call" node corresponding to service S3 and then executing them). A key point to note is that since no information is shared between S1 and S3, all test cases which cover all of S3 in S1 must be executed to ensure that any test cases which could cover the modification in S3 are executed.

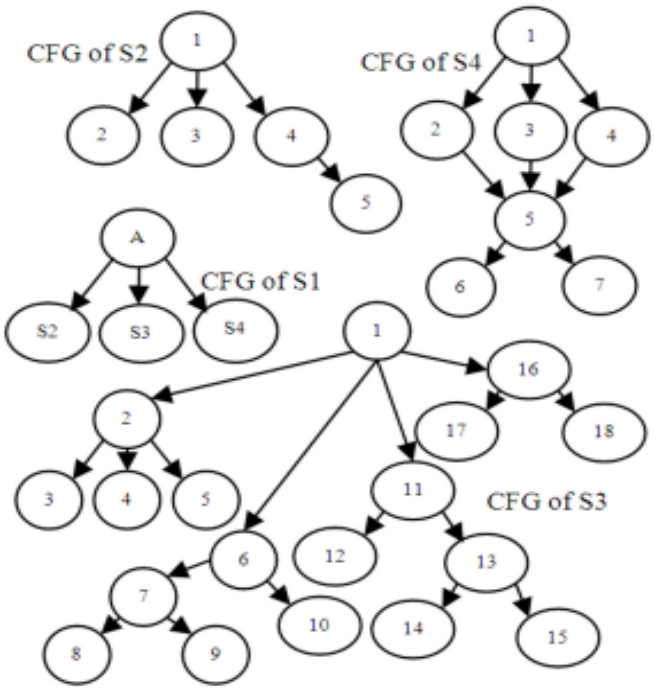

Figure 2. CFGs of the Purchase Order System

5. Empirical Study

As mentioned earlier, regression test selection techniques are only beneficial when the cost of running all tests is greater than the cost of performing the regression test selection process and the cost of running the tests selected by the process. This section will provide an empirical study of the outlined approach for applying anend-to-end, CFG-based RTS technique to Web services which was performed to investigate the feasibility of the approach, to determine if the approach can be beneficial, and to help guide future experimentation.

5.1 Subjects

In other empirical studies on the effectiveness of regression test selection techniques, the researchers used preexisting software systems which were used in other studies. Since no standard systems exist in the realm of Web services, developed these systems were developed for the purpose of studying the effectiveness of the approach. These systems are: a purchase order system (from the case study), a loan application system, and a supply chain management system. As described in the case study, it was chosen because it is very intuitive and indicative of real-world service interactions, and can be shown in full detail. The loan application system is a bank loan system which accepts and processes various types of loan applications. This system was chosen because it is frequently used in case studies to illustrate Web service systems [7] since they are indicative of the interactions of real-world Web systems. The supply chain management system manages supplies and processing of items for various manufacturing plants and this system was chosen because supply chain management systems are used in a variety of case studies related to Web services, including the system designed to illustrate the WS-I (Web Services Interoperability) standard [8]. Additionally, supply chain management is a real need for businesses today and remains an active topic for researchers. Table 1 shows some important statistics for each service that will take part in the empirical study including the number of services, the number of branches, and the number of test cases in each system.

5.2 Methodology

The framework must be constructed prior to operational phase which includes the construction of CFGs, test cases, and

coverage information that maps the test cases to the CFGs. Since the systemswere developed for this study, the test cases (and therefore coverage information) were generated to create code-coverage based test suites. Test cases were generated and then executed to determine which path in the CFG the test covered. If the path the test covered had less than 30 test cases, the test case was added to the test suite. Ensuring that each executable path in the system had 30 distinct test cases provides for an even distribution of test cases throughout each of the systems. Although this is not representative of real-world testing in which different branches have different priorities, an even distribution ensures equal consideration to every branch since testers may not be aware of the given priority of the branches. After generating the test cases for each service across each system, the entire test suite was executed to determine the cost of not performing the technique and executing all test cases everywhere in the system.

In this study, in each experiment the experimental systemswere modified by selecting an element of their CFG at random and then carrying out the modification. These experiments were carried out one thousand times to prevent selection bias from entering the results. In each system, all services were on the same machine to limit the influence of the network on the overall costs. The cost of the technique was determined by recording the time required to perform the operational phase at each service which includes rebuilding the CFG (partial for composite services and complete for atomic services), performing the impact analysis algorithm, selecting the test cases, and executing the selected test cases. In addition to recording overall costs, the costs of each of the individual components of the technique were recorded to provide insights into their relative contribution to the costs of the technique as a whole.

For each modification, the cost of performing the technique and executing only the selected test cases was compared to not performing the technique and executing all test cases.

5.3 Results

In this section, the results of the study will be presented and discussed. All of the graphs which show the results of the study, show the results of comparing the costs of performing the approach and running only the selected tests (steps 4a-4c) versus running all tests (step 3a). They show the percentage savings for each experiment which is obtained by dividing the cost of running all tests by the cost of performing the approach and running only the selected tests. Finally, the results of the experiments were sorted by percentage savings to more clearly show the results.

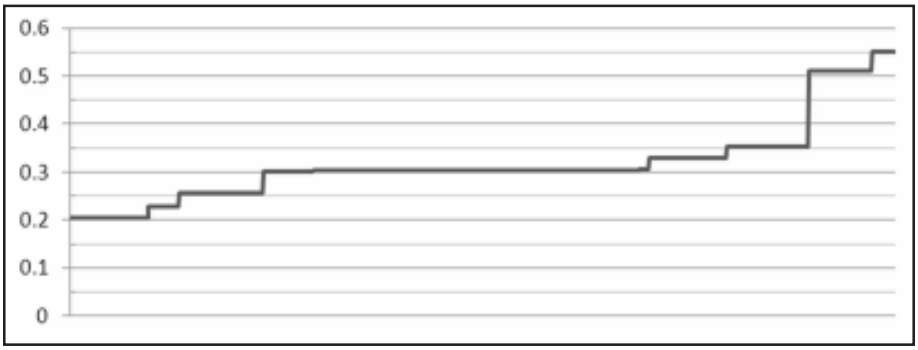

Figure 3. Results for the Purchase Order System

For example, the graph shown in Figure 3 is the result of performing the study on the POS.Along the y-axis is the percentage cost of performing the approach versus not performing the approach and along the x-axis are the experiments sorted by percentage. As shown in figure 3, the costs of performing the technique and executing only the selected test cases for the purchase order system was on average approximately 31% (with a standard deviation of approximately 8%) of not performing a selection step and executing all tests in the system with no cases reporting a higher cost for performing the technique as compared to not performing the technique. This indicates that the approach is clearly beneficial with an average performance gain of 69% for the purchase order system.

The result of the study performed on the loan application system is shown in Figure 4. The average cost for the loan application system was approximately 35% with a standard deviation of just under 9%. None of the experiments for this studyyielded a higher cost for performing the approach than running all tests, therefore the approach is clearly beneficial for the loan application system with an average 65% performance improvement over running all test cases in the system. The approach is clearly beneficial for the supply chain management system with an average cost savings of approximately 72% over running all test cases without a RTS selection step.

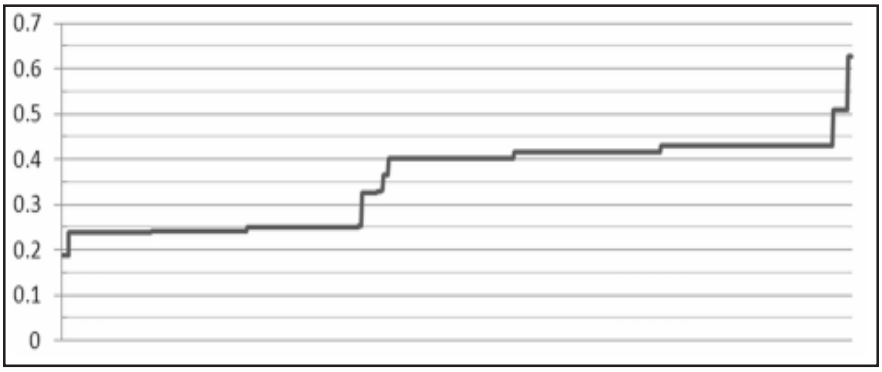

Figure 4. Results for the Loan Application System

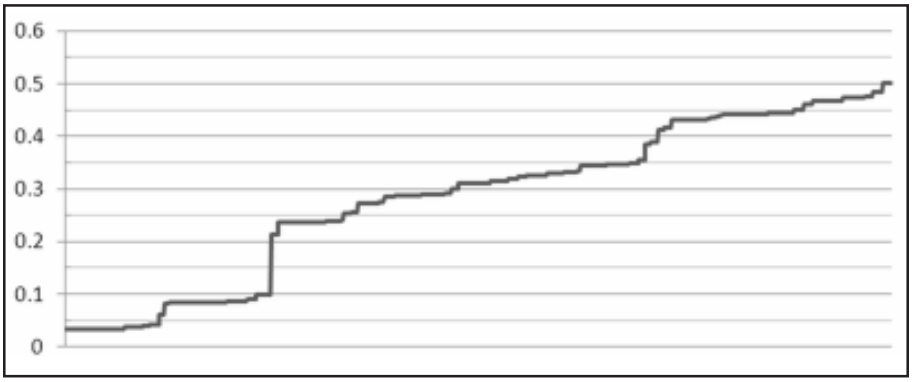

Figure 5. Results for the Supply Chain Management System

System	Services	Branches	Test Cases
Purchase Order System	4	42	1,260
Loan Application System	9	331	9,930
Supply Chain Management System	15	353	10,590

Table 1. Services, Branches, and Test Cases

System	Average Cost%	Average Savings %
Purchase Order System	31%	69%
Loan Application System	35%	65%
Supply Chain Management System	28%	72%

Table 2. Results of the Empirical Study for the Systems Studied

5.4 Discussion

As shown in the previous section, the results of the study were very promising. In fact, across the systems studied the costs of performing the technique and executing only the selected test cases on average was approximately 31% of not performing a selection step and executing all tests in the system. None of the experiments reported a higher cost for performing the technique as compared to not performing the technique.As described earlier, in order for a RTS technique to be considered cost-effective and therefore beneficial, the cost of performing the selection step (including all steps involved) and executing the selected tests must be less than the cost of not performing a selection step and executing all tests. Since, for all three systems studied the costs involved in performing the technique and executing only the selected test cases was much less than not performing the technique

and executing all test cases, the technique is clearly cost-effective and therefore beneficial for all of the systems studied.
.
One of the major reasons the technique is clearly cost-effective for this system is that the individual components of the technique contribute little to the overall costs in comparison to executing the test cases. One of the reasons that test cases for Web services carry such a high cost in comparison to traditional system is that test cases for Web services must be packaged in and unpackaged out of SOAP messages and transmitted across a network. In this empirical study, each system was on the same machine reducing, but not eliminating, the cost of transmission. Since the test cases are proportionately the largest contributor to the costs of the approach, the technique will be beneficial as long as the approach is selective and therefore removes some test cases.

Although the experimental study determined that our approach can be cost effective for the purchase order system, there are several important threats to validity which must be discussed. First, the inherent external validity which limits our ability to generalize the results of this experimental study to industrial practice which is a major concern since the study suffers from artifact representativeness since the subject program may not be representative of a real-world system. As described earlier, the purchase order system was chosen because it is used in a variety of Web service-based case studies, it is an intuitive system, its interactions are indicative of real-world service interactions, and the system is ideal for its purpose since the operation of the approach can be shown in full detail with complete CFGs. Although the approach was cost-effective for the system studied, additional empirical studies need to be performed to determine if the approach can be cost-effective for a wider variety of systems.

6. Related Work

The majority of RTS techniques designed for Web services augment service specifications with additional information to perform impact analysis [9-15]. Tsai, et al, presented a framework which augments WSDL with dependency information, function descriptions, and sequence specifications [2] designed to work with in a centralized UDDI service [9]. Later, they formalize the augmented information into contracts, including preconditions, post-conditions, obligations, benefits, and internal process information [10]. Another group of researchers proposed a model to perform a RTS technique for Web services using control-flow and data-flow analysis [11, 12]. In [11], service specifications are augmented with contract-based information such as precondition and effect which includes service dependencies. In [12], they augment service specifications with control-flow and data-flow information to build Guarded Finite-State Machines which can be then used in the same manner as their previous approach. Similarly, another group of researchers augment service specifications with graph transformation rules [13] designed to provide behavioral specifications which include data-flow information for composite services. Another approach focused on generating eXtensible BPEL flow graphs (XBFG) from BPEL specifications and using them to perform impact analysis [14]. Another group augments their specifications with a Labeled Transition Systems (LTS) to perform impact analysis in their framework designed to verify service choreographies [15].

There are several important differences between theirtechniques and ours including the handling of dependency information, RTS techniques employed, and the centralization of control. Several of these techniques require service providers to share dependency information with everyone in order for their technique to perform end-to-end impact analysis whereas our approach only shares this information with dependent services in the registration process. Our framework requires parties that are interested in modifications of any service, register at the service to be notified of any modifications in the service. The dependents of the service are registering interest and not dependency, so there is no guarantee that the service registering depends on the service. Also, since they are dependent services they have an interest in not disclosing this information because the composite service providers may use a different composed service if there is a risk that the dependency will be exposed. Also, the related techniques use specification-driven, or human-generated, information to perform impact analysis which may produce poor representations of the SUT which is considered unsafe [16]. Our technique only uses information derived directly from implementation so the technique can never suffer from inaccurate representations. These techniques also centralize the impact analysis which requires service providers to trust a third party to perform the impact analysis whereas our technique decentralizes the work thus ensuring no single point of trust.

Finally, in a previous work [17], we demonstrated a similar technique which also operates in a decentralized, end-to-end manner and employs a publish/subscribe mechanism to notify interested parties of modifications in dependent services. The key difference between our previous work and the current is our previous work focused on securing the shared artifacts using a

variety of privacy-preserving techniques,this work is focused on performing the RTS and regression testing processes without sharing any artifacts.

7. Conclusion & Future Work

Although RTS techniques are a critical component of any regression testing framework, service providers must be willing to participate in these frameworks. In general, all RTS frameworks are acceptable for use within an enterprise regardless of what information needs to be shared because service providers are willing to share sensitive information within an organization. However, outside of their organization, service providers are unwilling to expose sensitive information even if the RTS technique depends on this information.

Our CFG-based, decentralized regression test framework which employs a RTS technique is privacy-aware since it does not require the exposure of sensitive implementation details. It uses complete CFGs for atomic services and partial CFGs for composite services and a publish/subscribe notification system to ensure end-to-end automatic operation. Also, an empirical study was employed to study the cost effectiveness of the technique and the technique was clearly shown to be beneficial for the system studied. The future of this work will involve performing a series of additional experiments on more varied and more complex systems to further verify these early results.

References

[1] Rothermel, G., Harrold, M.J., A Safe. (1997). Efficient Regression Test Selection Technique, ACM Transactions on Software Engineering and Methodology, 6 (2) 173-210.

[2] Tsai, W.T., et al.(2002). Extending WSDL to Facilitate Web Services Testing, *In:* Proceedings of High Assurance Software Engineering (HASE) IEEE, p. 171- 172, Tokyo, Japan.

[3] W3C, Web service activities, http://www.w3.org/ws/

[4] Rothermel, G., Harold, M.J., (1998). Empirical Studies of a Safe Regression Test Selection Technique, *IEEE Transactions on Software Engineering,* 24 (6) 401-419.

[5] Offutt, J., Xu, W., (2004). Generating Test Cases for Web Services Using Data Perturbation, *Proceedings of Workshop on Testing, Analysis and Verification of Web Software*, SIGSOFT Software Engineering Notes, 29 (5) 1-10.

[6] Monson-Haefel, R., (2003). J2EE Web Services, 1st Edition, Addison-Wesley.

[7] Hohpe, G., (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley Professional.

[8] Ballinger, K., et al. (2003). Web Services Interoperability (WS-I) Basic Profile 1.3, http://www.ws-i.org/Profiles/BasicProfile-1.2.html.

[9] Tsai, W.T., et al. (2003). Verification of Web Services using an Enhanced UDDI Server, *Proceedings of Workshop on Object-Oriented Real-Time Dependable Systems* (WORDS), IEEE, p. 131- 138, Guadalajara, Mexico.

[10] Dai, G., et al. (2007). Contract-Based Testing for Web Services, *In:* Proceedings of Computer and Software Application Conference (COMPSAC), IEEE, p.517-526, Beijing, China.

[11] Li, L., Chou, W., Guo, W. (2008). Control Flow Analysis and Coverage Driven Testing for Web Services, *Proceedings of the International Conference on Web Services* (ICWS), IEEE, p. 473-480, Beijing, China.

[12] Li, L., Chou, W., (2009). An Abstract GFSM Model for Optimal and Incremental Conformance Testing of Web Services, Proceedings of the International Conference on Web Services (ICWS), IEEE, p. 205-212, Los Angeles, CA.

[13] Heckel, R., Mariani, L., (2005). Automatic Conformance Testing of Web Services, *Proceedings of Fundamental Approaches to. Software Engineering* (FASE), Springer Berlin / Heidelberg, LNCS 3442, p. 34-48, Edinburgh, Scotland.

[14] Wang, D., Li, B., Cai, J. (2008).Regression Testing of Composite Service: an XBFG Approach, *Proceedings of International Congress on Services*, Part II, IEEE, p. 112-119, Beijing, China.

[15] Mei, L., Chan, W.K., Tse, T.H. (2009). Data Flow Testing of Service Choreography, *In:* Proceedings of Fundamental Approaches to. Software Engineering (FASE), ACM, p. 151-160, Amsterdam, The Netherlands.

[16] Rothermel, G., Harrold, M.J., (1996). Analyzing Regression Test Selection Techniques,IEEE Transactions on Software Engineering, 22 (8) 529-551.

[17] Ruth, M., (2011). Employing Privacy-Preserving Techniques to Protect Control-Flow Graphs in a Decentralized, End-to-End Regression Test Selection Framework for Web Services, *In:* Proceedings of ICST Workshop on Regression Testing (Regression 2011), IEEE, Berlin, Germany.