A Capability Granularity Analysis on Web Service Invocations

Twittie Senivongse, Nattawud Phacharintanakul, Choltida Ngamnitiporn, and Matee Tangtrongchit

Abstract—**Web services technology embraces the use of standardized service interfaces, which serve as a service contract between service providers and service clients. A service generally has a service definition describing its capabilities, i.e. operations, which service clients can invoke. Service granularity is a key decision point during service design since the amount of capabilities offered by a service leads to issues in service usage such as reusability, composability, and performance. For a fine-grained Web service, more service invocations, i.e. more network roundtrips, are required for service clients to fulfill certain tasks. In this paper, we look at invocations of service clients to a particular Web service in order to determine if any pattern of calls exists for a set of the service operations. This can be a hint of fine-grained capabilities which have to be invoked together to fulfill a task. Our capability granularity analysis framework monitors Web service invocations and analyzes them using association rules and the Apriori algorithm to discover relations between the invoked operations. The analysis result can suggest the service provider a possibility to combine certain operations to reduce invocation costs.**

Index Terms—**Apriori, association rules, capability granularity, Web services.**

I. Introduction

Web services technology embraces the use of standardized service interfaces, which serve as a service contract between service providers and service clients. A Web service generally has an XML-based WSDL definition describing its offered capabilities in terms of service operations, data that are exchanged with the service, and a concrete detail about how to invoke service operations [1]. The service contract is of paramount importance as it is a mechanism to create understanding and enable interaction between service clients and service providers. Service contract design principles then are dedicated to standardized creation of the service contract.

Granularity is one key decision point during service design. According to Haesen *et al*. [2], granularity generally refers to the size of a service, and a fundamental design principle supports creation of services of large-sized or coarse-grained. This is due to the fact that business people are not interested in fine-grained concepts but prefer using automated chunks of functionality or services that correspond to units of work they usually handle. Different granularity types are defined at the level of the service interface from the point of view of a service client. Haesen *et al.* [2] propose three types of granularity: (1) *functionality granularity* which refers to how much functionality is offered by a service, (2) *data granularity* which reflects the amount of data that are exchanged with a service, and (3) *business value granularity* which indicates to which extent a service provides added business value. Erl *et al.* [3] classify the term granularity into (1) *service granularity* which represents the functional scope of the overall service context, (2) *capability granularity* which represents the functional scope of individual service capabilities (or operations), (3) *constraint granularity* which measures the level of validation logic a given capability will have, and (4) *data granularity* which represents the quantity of data processed. In this paper, we are interested in Erl *et al.*'s capability granularity (which corresponds to Haesen *et al.*'s functionality granularity).

The trend to design Web service capabilities that are coarse-grained has been encouraged as a means to overcome some of the performance challenges. When a capability is coarse, it abstracts a larger chunk of functionality within a single interaction. A Web service with finer-grained capabilities incurs more network roundtrips of service invocations in order for service clients to fulfill a certain task or (part of) a business process. Some fine-grained operations, processing fine-grained data in a traditional RPC style, may have a performance issue associated with XML-based processing. However, the coarser the capability granularity, the less reuse the service may be able to offer. If multiple functions are bundled into a single operation, it may be undesirable for service clients who only require the use of one of those functions. Also, operations with smaller functional scope generally require uncomplicated data and are more easily composed. Finding the right granularity is a matter of balancing between these multiple criteria.

Although service-related granularity is usually determined prior to service contract design, a particular service may vary over time in search of an appropriate granularity level for the corresponding vertical industry [4]. Even with a careful consideration at service design, after deployment a pattern of service usage may reveal inappropriate granularity which motivates the service provider to consider redesign of the service. In this paper, we are interested in a sign of fine-grained capabilities. We look at invocations of service clients to a particular Web service in order to determine if any pattern of calls exists for a set of the service operations. This can be a hint of fine-grained capabilities which have to be invoked together to provide certain functionality to a client

Manuscript received July 16, 2010.

T. Senivongse is with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand (phone: +66 2 2186996; fax: +66 2 2186955; e-mail: twittie.s@chula.ac.th).

N. Phacharintanakul, C. Ngamnitiporn, and M. Tangtrongchit are also with the Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok 10330 Thailand.

community, and hence cause a lot of high-latency service interactions. A capability granularity analysis framework is developed to monitor service invocations and analyze them. The analysis is by the use of association rules and the Apriori algorithm to discover relations between the invoked operations of the Web service and suggests the service provider a possibility to combine them to reduce invocation costs.

The organization of this paper is as follows. Section II discusses related work. Section III gives an overview of association rules and the Apriori algorithm and Section IV introduces a capability granularity analysis framework which applies them to service invocations to discover associations between service operations. Section V gives the evaluation results. A discussion and conclusion is found in Section VI.

II. Related Work

A number of literatures have addressed the importance of different types of granularity and their impact on service architecture. Nevertheless there is neither a concrete solution to how to get the right granularity nor a way to quantitatively determine if the granularity is right. Only guidelines are available for tackling this issue. Erl *et al.* [5] suggest to assign coarse-grained interfaces to services designated as solution endpoints and allow fine-grained interfaces for services confined to predefined boundaries so that interoperability is promoted in coarse-grained services and reusability is more fostered in finer-grained services. Schmelzer [6] addresses that there is no single measure for fine or coarse granularity since the measure applies in relation to the services available and the number of interactions required to accomplish a specific goal. Foody [7] gives a guideline when looking at granularity. On performance and size, rules of thumb are given to combine small operations or break apart bigger operations based on their response time, and limit the size of the messages exchanged with the operations. On transactionality and state, an operation should be self-contained and avoid keeping transient state across operations but at the same time should not do too much since failure in part may cause the whole operation to fail. On business suitability, understanding the big picture of the business is necessary for realizing what granularity makes sense.

An attempt by Shim *et al.* [8] defines metrics for an SOA system to measure the degree of service granularity and parameter granularity (i.e. data granularity). Service granularity takes into account the number of offered operations within the system as well as similarity between the operations which is determined by the number of similar messages that are exchanged with these operations. Parameter granularity considers the number of operations with coarse-grained parameters within the system. This quantitative approach determines granularity (i.e. size) by the number of explicit characteristics of the services but still requires judgment of the service provider on coarseness of parameters. In an attempt to identify a granularity level, we use a different approach which analyzes service usage to find a trace of multiple related invocations that may be combined for coarser-grained capabilities. We also focus on capability granularity of a single Web service.

III. Overview of Association Rules and Apriori

In this section, we give an overview of association rules and the Apriori algorithm.

A. *Association Rules*

Association rules [9] are often used to discover interesting relations between variables in large databases. The data on which association rules are applied are usually in the form of a database of transactions. For each transaction, the database contains the list of items that occur. The objective is to underline groups of items that typically occur together in a set of transactions. An example application is a market basket analysis that measures associations between products purchased at a supermarket.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items. Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of m transactions in a database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. For example, if I = {milk, cheese, coffee, biscuit}, a transaction t_1 = {milk, cheese} is a purchase of milk and cheese of a customer on a visit to the supermarket. With a large number of transactions in a database, rules that represent buying behavior of customers can be drawn. A rule is in the implication form $X \rightarrow Y$ where $X,Y \subseteq I$ and $X \cap Y = \varnothing$. X and Y are sets of items (or itemsets) and X is called an antecedent and Y a consequent. An example rule is {milk, cheese} $\rightarrow$ {biscuit} meaning that {milk, cheese} and {biscuit} occur together. In other words, if milk and cheese is bought, customers also buy biscuit.

Since there are many possible rules, minimum thresholds on the following constraints can be used to select interesting rules:

- support(X) is a measure of an itemset's frequency within the database. It is obtained by dividing the number of transactions that contain all items in X by the total number of transactions. For example, support({milk, cheese}) is 3/5 (i.e. 60%) if there are 3 transactions out of 5 transactions which contain {milk, cheese}.
- confidence($X \rightarrow Y$) is a measure of a rule's strength and indicates the frequency (or probability) of occurrence of Y, conditional on X being true. It is obtained by dividing the number of transactions that contain the items in $X \cup Y$ by the number of transactions that contain the items in X (i.e. support($X \cup Y$)/support(X)). For example, confidence({milk, cheese} $\rightarrow$ {biscuit}) is 2/3 (i.e. 66.67%) if support({milk, cheese} $\cup$ {biscuit}) is 2/5 and support({milk, cheese}) is 3/5.

The process to find interesting rules then comprises two steps: (1) Define a minimum support threshold and discover all itemsets for which their support passes this threshold. Such itemsets are called frequent itemsets. (2) Define a minimum confidence threshold and generate rules from the frequent itemsets. The rules for which their confidence passes this threshold will be selected. We use the Apriori algorithm to assist in these two steps.

B. *Apriori Algorithm*

Apriori is an algorithm for discovering frequent itemsets and association rules [10]. Its property guarantees that for a frequent itemset, also all its subsets are frequent, and thus for an infrequent itemset, all its supersets must be infrequent. The algorithm iteratively finds frequent itemsets of size 1 to k (k-itemset) as in the following pseudocode in Listing I.

LISTING I. APRIORI ALGORITHM.

```
C_k: {candidate itemsets of size k}
L_k: {frequent itemsets of size k}

L_1 = {frequent itemsets of size 1};
for (k = 2; L_{k-1} != ∅; k++) do
  begin
    C_k = {candidate itemsets generated from L_{k-1}};
    for each transaction t in the database do
      increment the support count of all candidate
        itemsets in C_k whose items are found in t;
    L_k = {candidate itemsets in C_k whose support
      count ≥ minimum support threshold;}
  end
return L = ∪_k L_k
```

To generate C_k in the algorithm above, the join step and prune step are performed respectively. In the join step, itemsets of size k are generated by joining L_{k-1} with itself. Then in the prune step, those itemsets of size k with any of its k-1-item subsets $\notin L_{k-1}$ (i.e. infrequent) will be discarded from C_k.

The procedure to generate association rules from L is in Listing II.

LISTING II. GENERATING ASSOCIATION RULES.

```
for each frequent itemset I where I ∈ L do
  generate all non-empty proper subsets s of I (s
    ≠ ∅ and s ≠ I);
for each non-empty proper subset s of I do
  output the rule s → I-s if support(I)/support(s)
    ≥ minimum confidence threshold;
```

IV. CAPABILITY GRANULARITY ANALYSIS

We propose a capability granularity analysis framework to analyze capability granularity of Web service operations as in Fig. 1. The major components of the framework are (1) Web service invocation monitor which intercepts service invocations from Web service clients, (2) Web service invocation log which records invocation data in a database, and (3) Web service invocation analyzer which performs an analysis on logged invocation data using association rules and Apriori. These components can be installed in the host environment of the Web services without affecting the hosted Web services and their clients.

A. *Web Service Invocation Monitor*

When there is a service invocation from a client, the Web service invocation monitor will intercept the message and record invocation data into the Web service invocation log. Then it will forward the request to the Web service and receive the reply before sending it back to the client.

The Web service invocation monitor supports JAX-WS Web services which is a Servlet in a container (or a Java EE application server). The container receives an HTTP request message before sending it to the Web service Servlet. We intercept this request message from the container by using a Servlet filter [11] which allows us to transparently examine the request before it reaches the Web service Servlet (Fig. 2). The filter reads the HTTP request header to get the Web service name and IP address and port number of the client, and reads the SOAP body in the HTTP request body to get the invoked operation name. Date and time of invocation are also recorded. Due to a limitation that an HTTP request body can be read only once, a wrapper has to be created to wrap the HTTP Servlet request in order for the Servlet to read the SOAP message within the HTTP request body for its processing of the request.

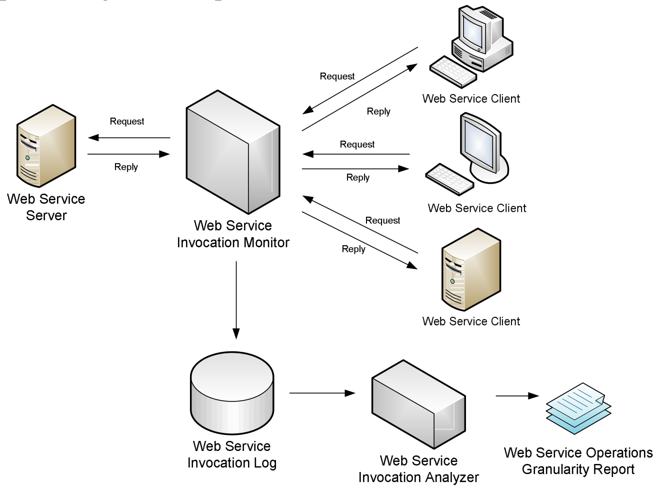

Request
Reply
Web Service Server
Web Service Invocation Monitor
Request
Reply
Web Service Client
Request
Reply
Web Service Client
Request
Reply
Web Service Client
Web Service Invocation Log
Web Service Invocation Analyzer
Web Service Operations Granularity Report

Fig. 1. Capability granularity analysis framework.

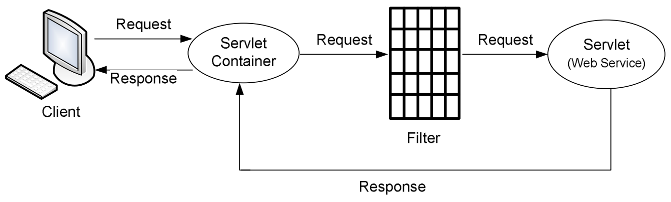


Fig. 2. Using filter to intercept invocations.

We use Java SE 6 update 11, Java EE 5, Apache Tomcat 6.0.18, and NetBeans IDE 6.5 in the implementation of the Web service invocation monitor.

B. *Web Service Invocation Log*

The Web service invocation log is a database of invocation data. The data schema consists of

1) *id:* ID of each invocation in the database
2) *timestamp:* date and time of invocation
3) *client_ip:* IP address of client
4) *client_port:* port number of client
5) *service:* Web service name
6) *operation:* Web service operation name.

We use MySQL Server 5.0.67, MySQL Query Browser 1.2.14, and phpMyAdmin 2.10.0.2 in the implementation of the Web service invocation log.

C. *Web Service Invocation Analyzer*

The Web service invocation analyzer performs an analysis on the logged invocation data. The analysis is based on an assumption that service operations of a Web service which are often observed to be called consecutively by clients are likely to form a logical group of tasks that are needed together at one time as part of some business process. These operations may be combined to reduce interaction cost for this group of tasks. The analysis produces a Web service operations granularity report which can suggest the service provider which service capabilities are likely to be fine-grained and may be combined into a more coarse-grained operation.

Before analyzing the invocation data, the Web service invocation analyzer prepares the transactions for the Apriori algorithm. Any two invocation entries in the log are “adjacent”, i.e. one invocation is immediately followed by the other, and hence belong to the same transaction if they are from the same client and the difference in their invocation time is within a boundary t. For example, given a Web service WS1 with the operations A, B, C, D, and E, and the logged entries in Fig. 3. If the time boundary is 1 second, we obtain transactions t_1-t_4 for the Apriori algorithm; there are two occurrences of a call to B followed by a call to C within 1 second from the same client, and a call to D is also followed by a call to E from the same client within this time boundary.

id	*timestamp*	*client_ip*	*client_port*	*service*	*operation*
1	2010-06-18 16:04:16	192.168.1.64	1235	WS1	B
2	2010-06-18 16:04:17	192.168.1.64	1235	WS1	C
3	2010-06-18 16:04:25	192.168.1.62	50648	WS1	A
4	2010-06-18 16:04:30	192.168.1.61	50101	WS1	D
5	2010-06-18 16:04:31	192.168.1.61	50101	WS1	E
6	2010-06-18 16:04:45	192.168.1.151	1039	WS1	B
7	2010-06-18 16:04:46	192.168.1.151	1039	WS1	C

⇩

t_1 = {B, C}
t_2 = {A}
t_3 = {D, E}
t_4 = {B, C}

Fig. 3. Preparation of transactions from logged data.

The Web service invocation analyzer analyzes the transactions using the Apriori algorithm. Since the problem here is to find two or more operations of a Web service which may be combined, the algorithm in Listing I starts with L_2 (i.e. the set of frequent itemsets of size 2) instead of L_1. Although time complexity of finding association rules is of $O(2^n)$ where n is the number of operations, the algorithm is considered practical for the problem because it is unlikely that a Web service would offer too many operations. The Web service invocation analyzer also allows for the time boundary, minimum support threshold, and minimum confidence threshold to be configured.

We use NetBeans IDE 6.5, Eclipse 3.4.1, and Java SE 6 update 11 in the implementation of the Web service invocation analyzer.

V. Evaluation

To evaluate the framework, we implement a Java Web service called Shopping which offers the operations in Table I, simulate invocations to this Web service, and mine the associations of invocations.

Table I. Operations of Shopping service.

Operations	*Arguments*	*Return Type*	*Description*
browseCatalog	String:keyword	int:categoryID	Browse catalog
cartCreate	int:customerID	int:cartID	Create cart
cartAdd	int:cartID String:item int:quantity	boolean:isFinished	Add item into cart
cartGet	int:cartID	Array:items	Get items in cart
cartRemove	int:cartID String:item int:quantity	boolean:isFinished	Remove item from cart
cartClear	int:cartID	-	Clear cart
calculatePrice	int:cartID	double:price	Calculate price
checkOut	int:customerID int:cartID	boolean:isValid	Check out cart
itemLookup	int:itemID	boolean:isFinished	Find item by id
itemSearch	String:itemName	int:itemID	Find item by name
sellerLookup	int:sellerID	boolean:isFinished	Find seller by id
sellerSearch	String:sellerName	int:sellerID	Find seller by name
customer Content Lookup	int:customerID	boolean:isFinished	Find customer by id
customer Content Search	String:customer Name	int:customerID	Find customer by name
login	String:username String:password	int:customerID	login
logout	int:customerID	-	logout

A. Invocation Sequences

We define ten patterns of invocation sequences, each specifying a sequence of calls to different number of operations at different time. The design of these patterns is based on Amazon E-Commerce Service [12]. As examples, two patterns are presented in Table II.

Table II. Two invocation sequences.

Pattern1	
Time (ith sec)	*Operation*
0	login
5	cartGet
10	cartAdd
11	calculatePrice
15	itemSearch
16	itemLookup
20	cardAdd
21	calculatePrice
25	itemSearch
26	itemLookup
30	cartAdd
31	calculatePrice
35	calculatePrice
36	checkOut
40	logout

Pattern2	
Time (ith sec)	*Operation*
10	login
15	browseCatalog
21	itemSearch
22	itemLookup
39	cartCreate
40	cartAdd
41	calculatePrice
45	browseCatalog
50	itemSearch
51	itemLookup
60	cartAdd
61	calculatePrice
70	logout

We simulate the invocation to the Shopping service using 15 clients, each iteratively selecting one of the patterns at random and making calls as in the pattern. We simulate the invocation until there are 20,000 operation calls recorded in the Web service invocation log.

B. *Mining Association Rules*

With the logged invocation data, we use the Web service invocation analyzer to analyze them. Fig. 4 shows its user interface. We specify the database of the Web service invocation log, Web service name, and necessary parameters for the analysis. Considering a possibility of service load and elapsed time of each request roundtrip, we specify the time boundary of 2,000 milliseconds for adjacent calls. The minimum support threshold can be specified as a percentage or the number of count, and the minimum confidence threshold as a percentage. The service provider can tune these parameters to obtain appropriate mining results.

After clicking the “Analyze” button, the Web service invocation analyzer shows the frequent itemsets (i.e. frequent groups of operations) which pass the minimum support threshold with their support count. Also all association rules and their confidence are listed. If the “Use Minimum Confidence to filter output rules” box is checked, the granularity report will show only the rules that pass the minimum confidence threshold. For example, in the figure, the rule cartGet $\rightarrow$ calculatePrice with confidence of 98.4127% passes the minimum confidence threshold. The service provider may consider combining them because when a customer lists all items in a cart, it is likely that the customer also wants to know the total price of all items.

VI. Discussion and Conclusion

This paper presents a capability granularity analysis framework that can find a trace of fine-grained capabilities of Web service operations by using association rules and the Apriori algorithm to analyze client invocation behavior. As a result of inappropriate design of service interface or change in invocation behavior over time, the analysis can guide the service provider to combine them to reduce invocation costs.

We see that the association rules that pass the minimum confidence threshold can only be a suggestion to the service provider. Combining service operations still needs careful consideration of the service provider since it changes the service contract and hence will affect service clients. Even though the fine-grained capabilities are desirable, the service provider can begin to explore the possibility of providing alternative WSDL definitions for the same Web service; one is fine-grained and the other is coarser-grained. Another possibility is to offer redundant fine-grained and coarser-grained operations in the same WSDL definition. These could de-normalize the service contract but can address performance issues and accommodate a range of clients [5].

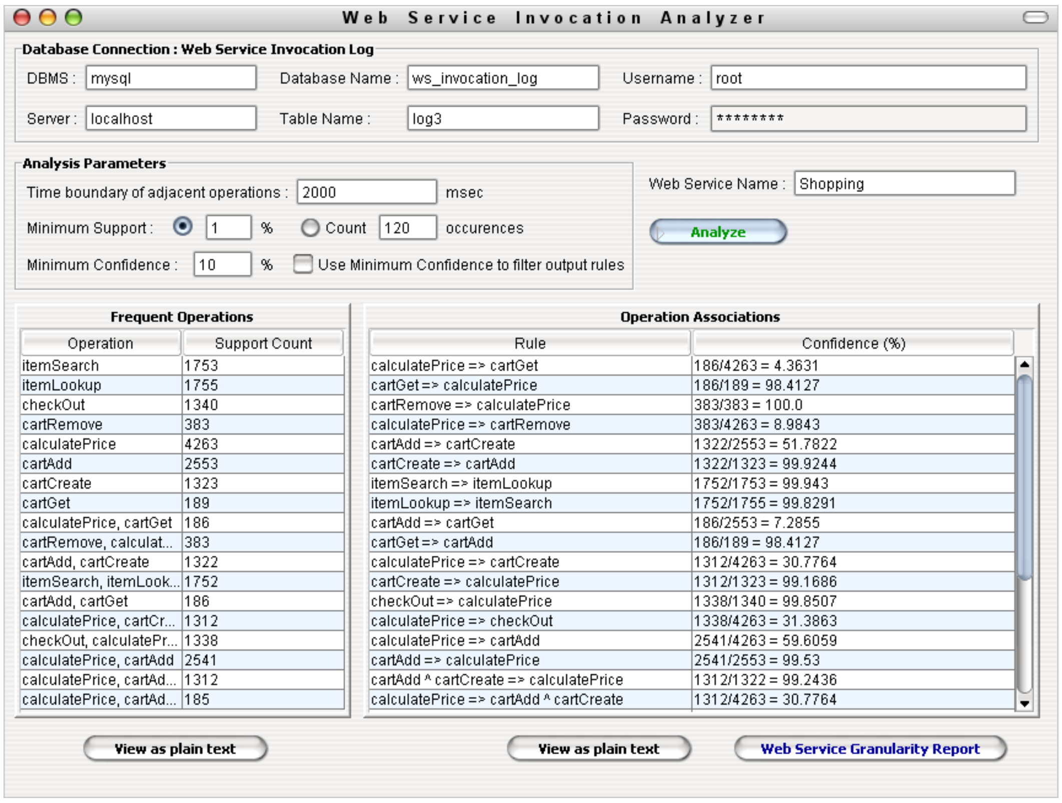
The granularity issue for services internal to an organization can be more relaxed than that for services offered at a larger scale with a diversity of clients. This is because patterns of service use by internal clients can be defined or modeled early. The organization can only focus on building the right set of services so that they serve the right set of problems for the organization.

Having only fine-grained services is beneficial when connecting or composing them with other services, but it would be hard to build anything large. Having only coarse-grained services that do certain amount of functionality would be good for such functionality, but they are not particularly reusable for building anything else [13]. Having an appropriate assortment of services, ranging from fine-grained to coarse-grained, to address particular business problems and clients would be an interesting approach.

Future work would be an application of this capability granularity analysis framework to real Web services, and a study of invocation patterns for services in particular vertical domains and the impact of all analysis parameters. Also an analysis of invocation costs that are reduced after combining service operations would help in a further evaluation of the framework.

References

[1] w3c, (2001, March 15). Web services description language (WSDL) 1.1 [Online]. Available: http://www.w3.org/TR/wsdl
[2] R. Haesen, M. Snoeck, W. Lemahieu, and S. Poelmans, “On the definition of service granularity and its architectural impact,” in *Proc. of 20th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2008)*, LNCS 5074, 2008, pp. 375-389.
[3] T. Erl *et al.*, *Web Service Contract Design and Versioning for SOA.* Prentice Hall, 2008, ch. 3.
[4] P. Herzum and O. Sims, *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise.* New York: John Wiley & Sons, Inc., 2000.
[5] T. Erl *et al., Service Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall, 2005, ch. 15.
[6] R. Schmelzer, (2007, August 3). The service granularity matrix [Online]. Available: http://www.zapthink.com/2007/08/03/the-service-granularity-matrix/
[7] D. Foody, (2005, August 13). Getting Web service granularity right [Online]. Available: http://soa-zone.com/index.php?/archives/11-Getting-web-service-granularity-right.html
[8] B. Shim, S. Choue, S. Kim, and S. Park, “A design quality model for service-oriented architecture,” in *Proc. of 15th Asia-Pacific Software Engineering Conference (APSEC 2008)*, 2008, pp. 403-410.
[9] R. Agrawal, T. Imielinski, and A. Swami, “Mining association rules between sets of items in large databases,” in *Proc. of 1993 ACM SIGMOD Conf.*, May 1993, pp. 207-216.
[10] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proc.of 20th VLDB Conf.*, 1994, pp. 487-499.
[11] B. Basham, K. Sierra, and B. Bates, *Head First Servlets and JSP, 2nd Ed.* O’Rielly Media, 2008, ch. 13.
[12] Amazon, (2007, April 4). Amazon E-Commerce Service Developer Guide [Online]. http://docs.amazonwebservices.com/AWSECommerceService/2007-04-04/DG/
[13] R. Schmelzer, (2005, November 15). Right-sizing services [Online]. Available: http://searchsoa.techtarget.com/tip/0,289483,sid26_gci1145391,00.html


Web Service Invocation Analyzer
Analyze
View as plain text
View as plain text
Web Service Granularity Report

Database Connection : Web Service Invocation Log

DBMS : mysql
Database Name : ws_invocation_log
Username : root
Server : localhost
Table Name : log3
Password : ********

Analysis Parameters

Time boundary of adjacent operations : 2000 msec
Minimum Support : [✓ 1 %] Count 120 occurences
Minimum Confidence : 10 % Use Minimum Confidence to filter output rules

Web Service Name : Shopping

Frequent Operations

Operation	Support Count
itemSearch	1753
itemLookup	1755
checkOut	1340
cartRemove	383
calculatePrice	4263
cartAdd	2553
cartCreate	1323
cartGet	189
calculatePrice, cartGet	186
cartRemove, calculat...	383
cartAdd, cartCreate	1322
itemSearch, itemLook...	1752
cartAdd, cartGet	186
calculatePrice, cartCr...	1312
checkOut, calculatePr...	1338
calculatePrice, cartAdd	2541
calculatePrice, cartAd...	1312
calculatePrice, cartAd...	185

Operation Associations

Rule	Confidence (%)
calculatePrice => cartGet	186/4263 = 4.3631
cartGet => calculatePrice	186/189 = 98.4127
cartRemove => calculatePrice	383/383 = 100.0
calculatePrice => cartRemove	383/4263 = 8.9843
cartAdd => cartCreate	1322/2553 = 51.7822
cartCreate => cartAdd	1322/1323 = 99.9244
itemSearch => itemLookup	1752/1753 = 99.943
itemLookup => itemSearch	1752/1755 = 99.8291
cartAdd => cartGet	186/2553 = 7.2855
cartGet => cartAdd	186/189 = 98.4127
calculatePrice => cartCreate	1312/4263 = 30.7764
cartCreate => calculatePrice	1312/1323 = 99.1686
checkOut => calculatePrice	1338/1340 = 99.8507
calculatePrice => checkOut	1338/4263 = 31.3863
calculatePrice => cartAdd	2541/4263 = 59.6059
cartAdd => calculatePrice	2541/2553 = 99.53
cartAdd ^ cartCreate => calculatePrice	1312/1322 = 99.2436
calculatePrice => cartAdd ^ cartCreate	1312/4263 = 30.7764

Fig. 4. User interface of Web service invocation analyzer.