Secure Web Service Using Call by Contract

R.Ashitha

Department of Computer Science Engineering

Anna University Tiruchirappalli, Tamilnadu, India

Abstract—**A methodology for designing and composing services in a secure manner. In particular, it's concerned with safety properties of service behavior. Services can enforce security policies locally and can invoke other services that respect given security contracts. This call-by- contract mechanism offers a significant set of opportunities, each driving secure ways to compose services. In this paper its discuss how we can correctly plan service compositions in several relevant classes of services and security properties. With this aim, we propose a graphical modeling framework based on a foundational calculus called formalism features the dynamic and static semantics, thus allowing for formal reasoning about systems. Static analysis and model checking techniques provide the designer with useful information to assess and fix possible vulnerabilities Such as Web services, call-by-contract, language-based security, static analysis, system verification**.

Keywords—**Call by contract, Hash table filtering, Systolic Array, Transaction trimming filter**

I.INTRODUCTION

The execution of a program may involve accessing security-critical resources and these actions are logged into histories. The security mechanism may inspect these histories and forbid those executions would violate the prescribed policies. Service composition heavily depends on which information about a service is made public, on

how those service match than user's requirements can be chosen and on their actual runtime behavior. Security makes service composition even harder. Services may be offered by different providers which only trust each other. On the other hand providers have to guarantee that the delivered service respects a given security policy in any interaction with the operational environment, regardless of who actually called the service. On the other hand, client may want to protect their sensitive data from the

service invoked. Security is a major concern in extensible software systems such as Java Virtual Machine and common language runtime. These system aim to enable simple, classic applets and also, for example, distributed applications, web services, and programmable networks, with applications, web services and programmable networks with appropriate security expectations. Accordingly, they feature elaborate constructs and mechanisms for associating rights with code, including for determining the run-time rights of piece of code as a function of the state of the execution stack. These mechanisms prevent many security holes, but they are inherently partial and they have proved difficult to use reliably. Motivated and described a new model for assigning right to code: in short, the run time rights of a piece of code that have run and any explicit request to augment rights. This history base model addresses security concerns avoiding pitfalls.

Analyzing the model in detail; in particular, discuss its relation to the stack-based model and to the policies and mechanisms of underlying operating systems and consider implementation techniques. In support of the model, also introduced and implemented high level constructs for security, which should be incorporated in libraries or in programming languages.

The formalism features dynamic and static semantics, thus allowing for formal reasoning about systems, Static analysis and model checking technique provide the designer with useful information to assess and fix possible vulnerabilities, A service oriented computing modeling framework supporting history based security and call by contract. The

distinguishing feature of the modeling framework is it provides high level constructs abstracting from the underlying middleware for service programming and deploying. The main result is a semantic-based methodology for synthesizing the skeleton structure of the orchestration engine, The orchestration plan details which services the orchestration engine has to choose in order to complete the original task while obeying the security policies on demand. Here dealt with security policies, but our methodology can be applied to handle variety of nonfunctional safety constraints.

II.TEQUNIQUE USED

(a)Hash table filtering

To build a hardware hash table filter, we use a hash value generator and hash table updating module. The former generates all the k-itemset combinations of the transactions and puts the k-itemsets into the hash function to create the corresponding hash values. As shown in Fig. 9, the hash value generator comprises a transaction memory, a state machine, an index array, and a hash function. The transaction memory stores all the items of a transaction. The state machine is the controller that generates control signals of different lengths k = {2; 3 . . .} flexibly. Then, the control signals are fed into the index

array. To generate a k-itemset, the first k entries in the index array are utilized.

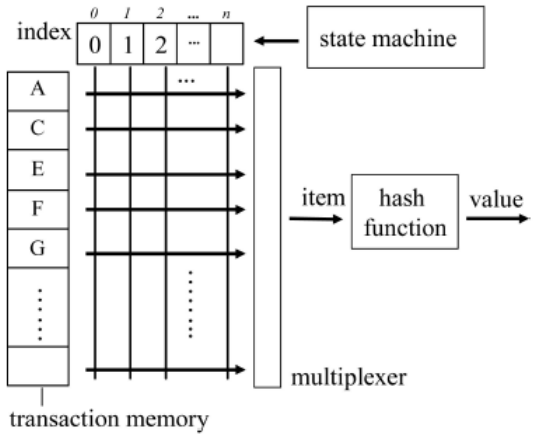


Fig. 9. The hash value generator.

The values in the index array are the indices of the transaction memory. The item selected by the ith entry of the index array is the ith item in a k-itemset. By changing the values in the index array, the state machine can generate different combinations of k-itemsets from the transaction. The procedure starts by loading a transaction into the transaction memory. Then, the values in the index array are reset, and the state machine starts to generate control signals. The values in the index array are changed by the different states. Each item in the generated itemset is passed to the hash function through the multiplexer. The hash function takes some bits from the incoming k-itemsets to calculate the hash values. Consider the example in Fig. 9. We assume the current k is 3. The first three index entries in the index array are used in this case. The transaction fA;C;E; F;Gg is loaded into the transaction memory. The values in the index array are initiated to 0, 1, and 2, respectively, so that the first itemset generated is < ACE > . Then, the state machine changes the values in the index array. The following numbers in the index array will be < 0; 1; 3 > , < 0; 1; 4 > , < 0; 2; 3 > , < 0; 2; 4 > , to name a few. Therefore, the corresponding itemsets are < ACF > , < ACG > , < AEF > , < AEG > , and so on. The hash values generated by the hash value generator are passed to the hash table updating module.

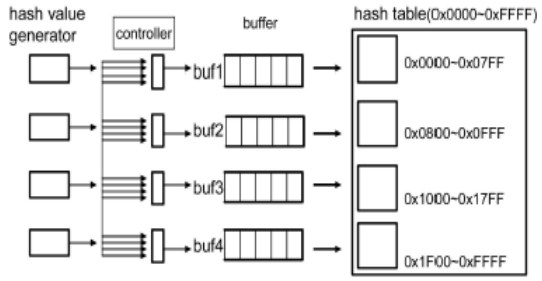


Fig. 10. The parallel hash table building module.

To speed up the process of hash table building, we utilize Nparallel hash value generators so that the hash values can be generated simultaneously. In addition, the hash table is divided into several parts to increase the throughput of hash table building. Each part of the hash table contains a range of hash values, and the controller passes the incoming hash values to the buffer they belong to. These hash values are taken as indexes of the hash table to accumulate the values

in the table, as shown in Fig. 10. There are four parallel hash value generators. The size of the whole hash table is 65,536, and it is divided into four parts. Thus, the range of each part is 16,384. If the incoming hash value is 5, it belongs to the first part of the hash table. The controller would pass the value to buffer 1. If there are parallel accesses to the hash table at the same time, only one access can be executed. The others will be delayed and be handled as soon as possible. The delayed itemsets are stored in the buffer temporally. Whenever the access port of hash table is free, the delayed itemsets are put into the hash table. After all the candidate k-itemsets have been generated, they are pruned by the hash table filter. Each candidate itemset is hashed by the hash function. By querying the number of itemsets in the bucket with the corresponding hash value, the candidate itemset is pruned if the number of itemsets in the bucket does not meet the minimum support criteria. Therefore, the number of the candidate itemsets can be reduced effectively with the help of the hash table filter

(b)Systolic Array

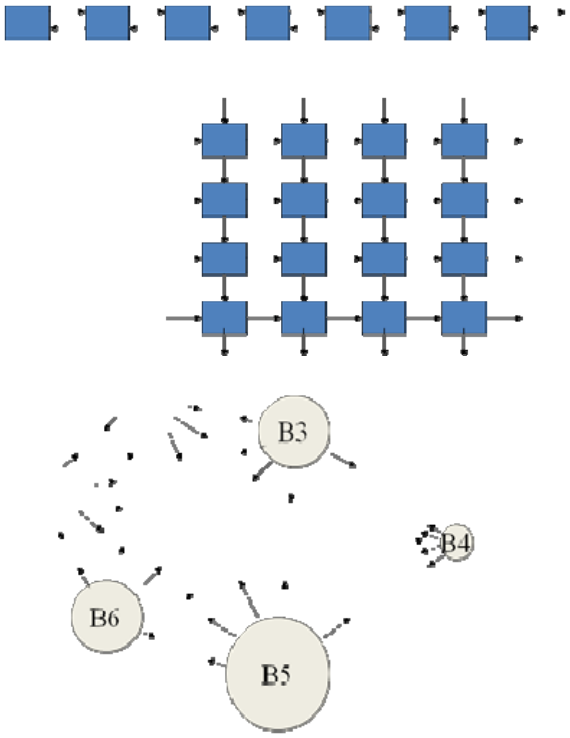
A systolic array is an arrangement of processors in an array where data flows synchronously across the array between neighbors usually with different data flowing in different directions.

Each processor at each step takes in data from one or more neighbors (e.g. North and West), processes it and, in the next step, outputs results in the opposite direction (South and East). H. T. Kung and Charles Leiserson were the first to publish a paper on systolic arrays in 1978, and coined the name.

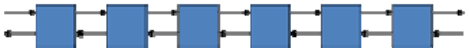
Systolic array is specialized form of parallel computing with multiple processors connected by short wires, unlike many forms of parallelism which lose speed through their connection. Cells(processors), compute data and store it independently of each other

Systolic unit is an independent processor in which every processor has some registers and an ALU. The cells share information with their neighbors, after performing the needed operations on the data.

Some simple examples of systolic array models.



Six planets in orbit with different masses, and different forces acting on each other, so are systolic array will look like this.

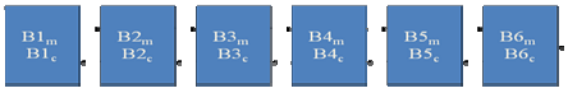


Each processor will hold the newtonian force formula:
$F_y = km_1m_2\ d^2_y$ **, k being the gravitational constant.**

Then load the array with values.

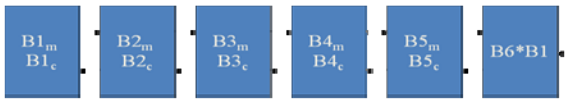
Now that the array has the mass and the coordinates of Each body we can begin are computation.

B6, B4, B4, B3, B2, B1



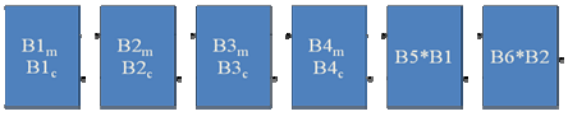
$F_{ij} = km_im_j/d^2_{ij}$

B6, B5, B4, B3, B2



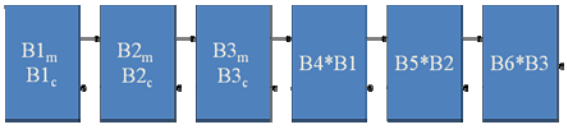
$F_{ij} = km_im_j/d^2_{ij}$

B6, B5, B4, B3



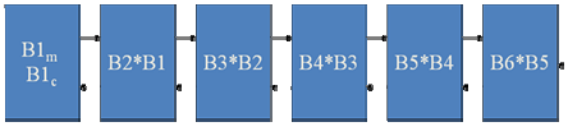
$F_{ij} = km_im_j/d^2_{ij}$

B6, B5, B4



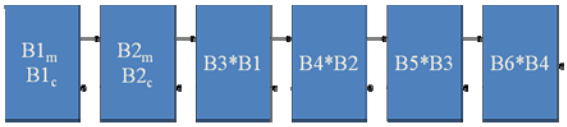
$F_{ij} = km_im_j/d^2_{ij}$

B6



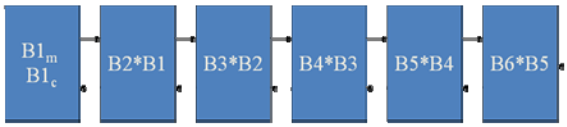
$F_{ij} = km_im_j/d^2_{ij}$

B6, B5

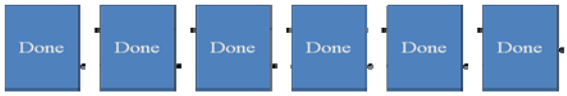


$F_{ij} = km_im_j/d^2_{ij}$

B6



$F_{ij} = km_im_j/d^2_{ij}$



$F_{ij} = km_im_j/d^2_{ij}$

Systolic is extremely fast, easily scalable architecture and can do many tasks single processor machines

cannot attain Turns some exponential problems into linear or polynomial time.

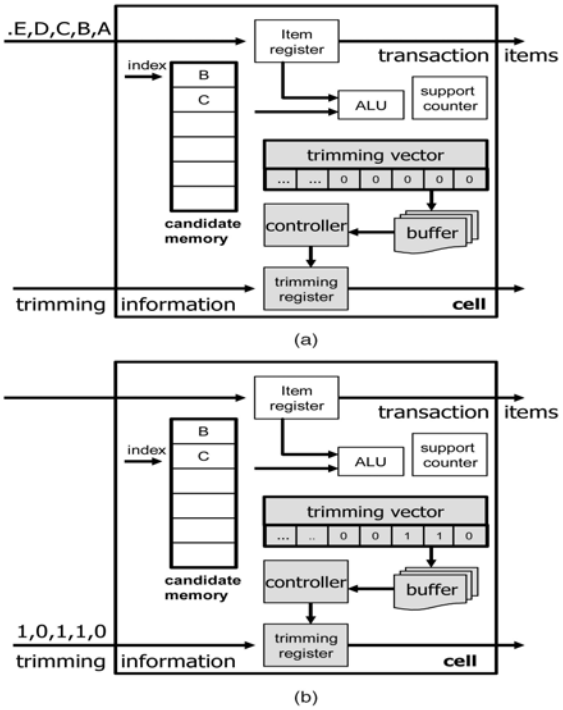
(c)

The theoretical backgrounds of the pruning scheme are based on the following two theorems which were presented.

Theorem 1. A transaction can only be used to support the set of frequent (k+1)-itemsets if it consists of at least (k+1) candidate k-itemsets.

reduced.

In the HAPPI architecture, the trimming information records the frequency of each item in a transaction that appears in the candidate itemsets. The support counting and trimming information collecting operations are similar since they all need to compare candidate itemsets with transactions. Therefore, in addition to transactions in the database, their corresponding trimming information is also fed into the systolic array in another pipe, while the support counting process is being executed. As shown in Fig. 7, a trimming vector is embedded in each hardware cell of the systolic array to record items that are matched with candidate itemsets



Theorem 2. An item in a transaction can be trimmed if it does not appear in at least k of the candidate k-itemsets contained in the transaction While the support counting procedure is being executed, the whole database is streamed into the systolic array. However, not all the

Fig. 7. An example of streaming a transaction and the corresponding trimming information into the cell. (a) Stream a transaction into the cell.(b) Stream trimming information into the cell.

The ith flag in the trimming vector is set to true if the ith item in the transaction matches the candidate itemset. After comparing the candidate itemset with all the items in a transaction, if the candidate itemset is a subset of the transaction, the incoming corresponding trimming information will be accumulated according to the trimming vector. Since transactions and trimming information are input in different pipes, support counters and trimming information can be updated simultaneously in a hardware cell. In Fig. 7a, the candidate itemset < BC > is stored in the candidate memory, and a transaction {A;B;C;D;E} is about to be fed into the cell. The resultant trimming vector after comparing < BC > with all the items in the transaction is shown in Fig. 7b. Because items B and C are matched with the candidate itemset, the trimming vector becomes < 0; 1; 1; 0; 0 > . Meanwhile, the corresponding trimming information is fed into the trimming register, and the trimming information is updated from < 0; 1; 1; 0; 1 > to < 0; 2; 2; 0; 1 > . After passing through the systolic array, transactions and their corresponding trimming information are passed to the trimming filter. The filter trims off items whose frequencies are less than k. As the example in Fig. 8 shows, the trimming information of the transaction {A;B;C;D;E}is < 2; 2; 2; 1; 2 > and the current k is 2. Therefore, the item D should be trimmed. The new transaction becomes {A;B;C;D}. In this way, the size of the database can be reduced. The trimmed transactions are sent to the hash table filter module for hash table building.

III PHASES

(a) Semantics of Services

That the services that initiate a computation are furnished with an arbitrary plan. In the next section, we shall discuss static machinery that will enable us to construct these plans to guarantee that computations will never go wrong, that is, they satisfy all of the contracts and the security policies on demand. We shall also show some strategies to adopt when services disappear unexpectedly. The graph-rewriting semantics for the case of dependent threads is split into two parts: basic activities, events, security blocks, requests, and returns .Details the rules that involve planning and recovering strategies, that is, planning blocks, requests to unavailable services, unresolved requests, services going down and up, and publication of new services. We shall briefly discuss the case of independent threads later in this section. All of the remaining axes in the taxonomy are covered by our semantics. When irrelevant, we omit the label in services. Note that the actual values for some labels in rules REQ and RET are defined later on, since they depend on the choice made on the security aspects discussed. This gives rise to different behaviors of requests and returns according to the possible choices in the taxonomy

(b)Network composed

A network composed of two services at locations, both stateful and sharing a global history. The service at starts with an action (the omitted target object is immaterial in this example) and then issues a request, resolved by the provided plan. The service at performs within a security block, where the policy prevents from being fired twice. A computation of the network is depicted, where we assume that the execution monitor is on. Note that the plan is not viable because it drives a computation that fails because of an attempted security violation right before the second is fired.

(c)Independent Threads

To model independent threads, each service must keep a separate history for each thread. Equivalently, we keep a history for each thread initiator. With this aim, instead of a single history, services now carry a function mapping the label of each initiator to its corresponding history. Moreover, we keep track of the initiator name: Each service invoked on behalf of the initiator is tagged as such. The semantics of services can now be easily adapted, making play the role of in the semantics for dependent threads. The rule REQ for independent threads.

(d)Security policies

Security policies are regular properties of histories and they are defined through template security automata. A template security automaton gives rise to a finite-state automaton when the parameter x is instantiated to an actual object o. These automata will be exploited to recognize those histories obeying.

IV.EXPERIMENTAL RESULTS

In this paper consider the example of such transactions, pay-pal-system in purchasing items, form an increasingly important sector within the retail economy and are usually conducted by customers entering relevant details, such as credit card number, name etc, on a form-page of a vendor\'s website. Alternatively it is possible to rely on the vendor having stored those details from a previous transaction, so as to simplify subsequent purchases from that customer.

A stateless service does not preserve its history across distinct invocations (yet it checks the history within each invocation). Instead, a stateful service keeps track of the histories of all the past invocations. Stateless services can enforce policies that inspect the history of the current invocation only, for example, resource usage control. Stateful services allow for more expressive security policies. For instance, a stateful service can bound the number of invocations on a per-client basis, whereas a stateless service cannot. More generally, stateful services can exploit their histories to record security-relevant information about the state of client sessions. Consider, for instance, a service that requires password authentication and that gives only three chances per hour to authenticate. This can be modeled as a stateful service. In this case, the history keeps track of the number of failed authentication attempts. The security policy prevents the service from being used by a caller for which the history has recorded three failed authentication attempts in the last hour.

Although stateless services admit security policies that are less expressive than those of stateful services, static analysis can usually infer enough information to ensure secure composition.

Security is of paramount importance in conducting such transactions, since access to the account details of a legitimate customer can be exploited for fraudulent purposes such as the manufacture of fake cards etc. If account details are transmitted over internet connections in an unencrypted form, an eavesdropper with access to an intermediate router can access these sensitive details. Alternatively, fake "look-a-like" websites can lure innocent customers into providing their account details in the belief that they are communicating with a reputable vendor. Finally, it is not usual for the web-servers of legitimate traders to become the subject of attacks, as a result of which databases of previously stored account details can be compromised.

To guard against many of these attacks, web-traders commonly employ Secure Sockets Layer (SSL)-protected servers. These address some, but not all, of the security threats by encrypting communications between the customer and the web-server, and also serve to ensure that the web-server must present a digital certificate which associates its identity with a cryptographic public key.

It is also noted that many customers find the process of entering details on web-forms confusing and error prone, and the invention serves to ease these difficulties also.

That is, currently, when a customer uses its web browser to access a "sales" website presented by an e-commerce server, the customer and device are anonymous to the server. An SSL connection is set-up between the browser in the PC or other terminal device used by the customer. The customers can, by using the SSL protocol, verify the authenticity of the server because it is part of a Public Key Infrastructure (PKI) and the server then presents a certificate which chains-back to a root certificate stored in the customer (client) browser. The customer\'s device can then choose a random value, send it to the server encrypted under its certified public key, and use it as a basis for the computation of symmetric keys used to provide confidentiality and integrity of the information that the customer wishes to send to the server.

As far as the e-commerce server is concerned, this information has come from an anonymous source, and the vendor then seeks authorization from the credit card company based on creditworthiness, card validity, security checks on supplied billing address etc. However no steps have been taken to authenticate the source of the information by either user authentication or device authentication. The authorization process is arranged simply to check that it looks like a valid order.

V.RELATED WORK

Several approaches have been developed to support the verification of service-oriented systems. For example, dynamic bisimulation-based techniques have been adopted to analyze the consistency between orchestration and choreography of services [21], [22], whereas state-space analysis has been exploited to check the correctness of service orchestration [31]. Our approach allows for synthesizing and checking the correctness of the orchestration statically.
Process calculus techniques have been used to formalize Web Services standards (for example, see [28], [18], [35], [37], [23]). A different approach is [38], an abstract programming model for structured orchestration of services. Web service authentication has been recently modeled and analyzed in [14], [15] through a process calculus enriched with cryptographic primitives. From a software engineering perspective, the advent of service-oriented applications has led to the development of service interfaces and orchestration (business) logic and abstract from the underlying programming middleware. A well-known example is provided by the Service Component Architecture (SCA) [24]. This framework aims at simplifying implementations by allowing designers to focus only on the business logic while complying with existing standards. Our approach complements the SCA view, providing a full-fledged mathematical framework for designing and verifying properties of service assemblies. It would be interesting to develop a (model-transformation) mapping from our formal framework to SCA. In addition, related to our approach is the work on SRML [29], a high-level core language, independent of the underlying programming middleware. SRML has a mathematical semantics, providing and offers both syntactic and

behavioral service interfaces. The logic for the specification of behavioral properties of services is still under development. The interconnections between services are specified in a declarative style, but they are not driven by the properties of a contract, as is proposal.

REFERENCES

[1] Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari, and Roberto Zunino "Semantic based design on secure web service"2008
[2] M. Abadi and C. Fournet, "Access Control Based on Execution History," Proc. 10th Ann. Network and Distributed System Security Symp., 2003.
[3] S. Anderson et al., "Web Services Trust Language (WS-Trust),"technical report, 2005.
[4] B. Atkinson et al., "Web Services Security (WS-Security),"technical report, 2002.
[5] A. Banerjee and D.A. Naumann, "History-Based Access Control and Secure Information Flow," Proc. Workshop Construction and Analysis of Safe, Secure, and Interoperable Smart Cards, 2004.
[6] H.P. Barendregt et al., "Term Graph Rewriting," Parallel Languages on PARLE: Parallel Architectures and Languages Europe, 1987.
[7] M. Bartoletti, P. Degano, and G.L. Ferrari, "History Based Access Control with Local Policies," Proc. Eight Int'l Conf. Foundations of Software Science and Computation Structures, 2005.
[8] M. Bartoletti, P. Degano, and G.L. Ferrari, "Planning and Verifying Service Composition," Technical Report TR-07-02, Dept. of Informatics, Univ. of Pisa, 2007, J. Computer Security, to appear.
[9] M. Bartoletti, P. Degano, G.L. Ferrari, and R. Zunino, "Types and Effects for Resource Usage Analysis," Proc. 10th Int'l Conf. Foundations of Software Science and Computation Structures, 2007.
[10] M. Bartoletti, P. Degano, and G.L. Ferrari, "Enforcing Secure Service Composition," Proc. 18th Computer Security Foundations Workshop, 2005.