SECURE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL

by

Wei She

APPROVED BY SUPERVISORY COMMITTEE:

_____
Bhavani Thuraisingham, Chair

_____
I-Ling Yen, Co-Chair

_____
Weili Wu

_____
Latifur Khan

SECURE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL

by

WEI SHE, B.S., M.S.

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

December 2011

PREFACE

This dissertation was produced in accordance with guidelines with permit the inclusion as part of the dissertation the text of an original paper or papers submitted for publication. The dissertation must still conform to all other requirements explained in the "Guide for the Preparation of Master's Theses and Doctoral Dissertations at The University of Texas at Dallas." It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts which provide logical bridges between different manuscripts are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signature of the Supervising Committee which precedes all other material in the dissertation attest to the accuracy of this statement.

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Dr. Thuraisingham, and my co-advisor, Dr. Yen, who were abundantly helpful and offered invaluable assistance, support, and guidance.

Deepest gratitudes are also due to the members of the supervisory committee, Dr. Wu and Dr. Khan, without whose knowledge and assistance, this study would not have been successful.

I would also express my love and gratitude to my beloved families and friends, for their understanding and endless love, through the duration of my studies.

This material is based in part upon work supported by the Air Force Office of Scientific Research under Award No. FA-9550-08-1-0260. I thank Dr. Robert Herklotz for his support.

November 2011

SECURE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL

Publication No. _____

Wei She, Ph.D.
The University of Texas at Dallas, 2011

Supervising Professor: Bhavani Thuraisingham, Chair
I-Ling Yen, Co-Chair

Service-Oriented Architecture (SOA) is the current paradigm to achieve global system integration and collaboration. Although SOA has many benefits, security is still a major concern. Access control is one of the major issues in secure SOA. It is necessary to develop suitable access control models to secure individual web services. Also, when multiple services hosted by different providers are composed together to realize certain business logic, it is desirable to ensure the secure interactions between the involved entities. In this dissertation, we focus on three major access control issues in service composition: (1) Information flow control, which controls the propagation of sensitive information in composite services, (2) Integration of action-level access control and fine-grained data resource level access control to secure web services, and (3) Composition-time access control validation to minimize the execution-time failure rate of the composed composite services.

In this dissertation, we take a close look at these three issues and provide a comprehensive set of solutions to the secure service composition in multi-domain web service environment. First, we have developed a fine-grained information flow control model (Chapter 4) and introduced the novel concept of transformation factor to model the computation and "transformation" effect of intermediate services. Our approach can significantly simplify the information flow control policies and improve the access control validation performance. Second, we have developed a fine-grained data resource level information flow control model (Chapter 7) based on the data flow analysis and tracking techniques. This model is capable of securing the flow of data that are dynamically generated in composite services. Third, we develop protocols to achieve composition-time access control considering both mediator-based (Chapter 5) and fully decentralized composition architectures (Chapter 6). Our protocols are highly efficient and can greatly enhance the performance in composing and executing composite services with proper information flow control constraints.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Service-Oriented Architecture (SOA) has currently been widely adopted in many applications/systems, especially in distributed collaborative applications, to flexibly and cost-effectively integrate services from multiple administrative domains into large-scale systems to achieve potentially globalized collaborative tasks. While SOA has many benefits, security is still a major concern. In the past decade, a lot of research efforts have been conducted to develop suitable models and mechanisms to secure web services. One major advance is the definitions of a set of web service security specifications, such as WS-Security [OAS06], defining how to use XML-signature and XML-encryption to secure SOAP message exchange, WS-Trust ([OAS07a], defining how to issue, renew, and validate credentials, WS-SecureConversation [OAS07b], defining how to secure conversations with multiple SOAP message exchanges, WS-SecurityPolicy [OAS07c], defining how to represent capabilities and constraints of web services as policies, etc. These security specifications mainly focus on authentication and secure message exchange issues in web service environment. They have provided standardized protocols to secure interactions in web service environment.

Access control is a challenging problem in web service systems, due to the open environment and involvement of many different administrative domains. It is difficult to model such systems and control the accesses to services and/or other resources from the entities in some domains that

are unknown to the service provider. The issues regarding web service access control can be considered in two directions. On one hand, it is necessary to develop suitable access control mechanisms to secure individual web services from unauthorized accesses due to misuse or malicious attacks. On the other hand, when multiple web services hosted by different service providers are composed together to realize certain business logic, it is necessary to ensure that the interactions between the user and the services and between different services conform to the access control policies of all the involved entities.

This dissertation focuses on access control in service composition. We consider several issues that have not been well addressed in the literature. First, most existing web service access control models focus on individual web services and do not consider controlling further propagation of sensitive information in a chain of services. To avoid undesirable information leakage, it is essential to build models and develop techniques to secure the information flow in composite services. Second, most existing web service access control models focus on the action level without considering data resource protection. Action-level access control is coarse-grained, considering only the rights of service invocation, and, hence, cannot control the accesses to data resources within the web services at a fine-grained level or to the data flowing through the services. Thus, it is necessary to develop suitable data resource level access control models to further secure web services. Third, the current secure service composition mechanisms do not consider the evaluation of access control policies at the composition time. As many service providers need to enforce their access control policies at the execution time, the composed composite services are likely to fail at execution time due to the access control violations. Thus, it is beneficial to develop a service composition protocol with composition-time access control

validation. We discuss these three issues in details in Section 1.1, and the solutions to these problems are presented in Section 1.2

1.1 ISSUES AND EXISTING WORKS

1.1.1 Information Flow Control in Composite Services

Most existing web service access control models only consider the direct accesses to web services [BER06, BHA04, ARD11, PAC11, WON04]. When considering indirectly interacting web services and/or users, they implicitly assume a transitive trust relationship, i.e. if *A* trusts *B* and *B* trusts *C* then *A* trusts *C*. However, in a widely distributed multi-domain environment such as the web service systems, such a transitive trust relationship can hardly be true and, hence, and it is necessary to develop models and mechanisms to control the flow of sensitive information in composite services.

The information flow control problem in service chains is two fold. First, the sensitive information of a service may be disseminated, directly or indirectly (through a chain of services), to another service in raw or processed forms. For example, in the service chain $<s_0, s_1, \ldots, s_{n+1}>$, the output data of service s_i, $0 \leq i \leq n$, may be computed from some sensitive information of s_i and/or of s_i's prior services. When delivered to the subsequent services, s_j, $i < j \leq n+1$, directly or indirectly, s_j may derive the sensitive information of s_i and/or s_i's prior services using the data it has received. Second, the information that a service writes to its local data storage may be computed using some potentially corrupted information. For example, in the above service chain, the information that s_j, $0 < j \leq n+1$, writes to its backend database may be computed using the potentially corrupted information of its prior services, i.e. $s_{j-1}, \ldots, s_0$. When the write operation is

performed, the critical data stored in the backend database may be contaminated. As such information flows may result in severe confidentiality (first case) and/or integrity (second case) problems, suitable models and mechanisms are required to achieve information flow control.

There have been some preliminary works that address the information flow control problem in composite services [CHA05, YIL07, SRI07]. However, these works are either too strict, treating direct or indirect accesses exactly the same way [CHA05, YIL07], or too complex, requiring exhaustive enumeration of all the possible combinations of intermediate services for specifying information flow control policies [SRI07]. To achieve fine-grained information flow control while avoiding the pitfalls in existing models, it is necessary to model the computation effect that a service has over its input/output.

1.1.2 Action Level and Data Resource level Access Control

Existing web service access control models can be classified into two categories, action level and data resource level access control. In action-level access control, the policies are defined over service operations and enforced at the service invocation time, whereas in data resource level access control, the policies are defined over the data resources accessed within services, such as files, tables, etc., and are enforced during service execution when each critical resource is accessed. Action-level access control models are usually coarse-grained, in the sense that, once a user is granted the privilege to access a service operation, he/she is also granted the privileges to access all the data resources that may be accessed within the service operation.

Most existing web service access control models focus on action level protection [ARD11, BER06, BHA04, PAC11]. They consider advanced models to secure the accesses to service

operations, such as adaptive web service access control [BER06], context-aware access control [BHA04], access control for conversational web services [PAC11], credential-based access control [ARD11], etc. Some existing works [OAS07, WON04] also consider the data resource level protection by integrating conventional data resource level access control models [BEL73, FER01, LAM74, RHA05].

Action-level access control cannot guarantee the data resource level protection unless the following two conditions are satisfied: (1) the accesses to the data resources by each service operation are static; (2) the service operations are sufficiently fine-grained such that the data resources accessed by one service operation require the same privilege as the privilege required by the service operation. But in practice, in many services, the accesses to the data resources may only be determined at run time. Even if the data resources accessed by a service operation can be determined statically, the business logic in the service operation may be too complex to allow a thorough validation to determine whether the privileges required by all the data resource accesses are compatible (no mutually exclusive privileges exist). In these cases, action-level access control is insufficient and it is necessary to provide data resource level protection.

When action-level access control is feasible, it is a desirable approach as it only requires the validation once at the service invocation time. In data resource level access control, the access control decisions need to be made each time when the critical data resources are accessed, resulting in a high overhead and potentially wasted computation efforts and time (if failure to access data resources implies aborting the execution). Also, current data resource level access control models cannot adequately address the security requirements in composite services, as in composite services, there are many data dynamically computed and exchanged, but in existing

data resource level models, the access control policies are usually defined on preexisting data resources.

1.1.3 Composition-Time and Execution-Time Access Control

It is necessary to perform access control at execution-time to ensure the secure use of web services and/or other resources, as the requester information and/or the accesses to some data resources within a service execution may not be determined prior to the execution. All existing access control models consider the enforcement of access control policies at the execution time. However, it is beneficial to also consider the access control policies at the service composition time. Without the composition-time access control, the composed composite services may be very likely to fail at the execution time due to the access control violations, wasting composition and execution efforts. To avoid repeating failed compositions, bookkeeping of the failures becomes necessary, resulting in a more complicated and time consuming composition and execution protocol. Thus, it is desirable to develop a secure service composition mechanism to minimize the execution-time failure rate of the composite services by introducing composition-time access control.

There have been some security-aware composition mechanisms proposed in recent years [BAR08, CAR06, DEN03, HAN06, PAC08]. These works characterize security as a set of attributes that can be quantitatively measured. The security properties of a service can be specified in terms of these attributes. To protect critical data resources and/or services, the user and the service providers may define security constraints (i.e. policies) for a composition in terms of these attributes. A composite service is considered to be secure if the security properties

of all individual component services satisfy all the security constraints defined by the service providers and the user. One major issue with these works is that they consider very simple attributes such as the type of encryption algorithm, the type of authentication protocol, trust and reputation, etc. Although it is possible to extend these mechanisms to include access control policies as security constraints, none of these works consider the specific issues involved in modeling, specification, and enforcement of these policies.

There are several additional issues when considering access control at the service composition time. The first issue is who should evaluate the access control policies. Existing security-aware service composition mechanisms usually implicitly assume a fully trusted service composer. However, in a distributed multi-domain environment, there are many users in different domains and they may use different service composers. These distributed service composers may be in different domains from those of the web services. As some of these domains may have protected access control policies that should not be released to some parties (e.g. some service composers), it is unlikely that a service composer is fully trusted by the providers of all the involved services (all the concrete services considered by the service composer, not just those actually selected) for accessing their protected policies. Consequently, the service composer cannot complete the policy evaluation without interacting with the service providers with protected access control policies.

Second, the performance issue in secure service composition should be carefully considered. The service composer may have to explore a lot of candidate compositions to find one candidate that satisfies the security constraints of all component services. To validate each candidate composition, the service composer needs to validate each pair of services (s_i, s_j) by evaluating

s_i's and s_j's security properties against s_j's and s_i's access control policies. The access control policies of some component services may be very complex and require a significant amount of evaluation time. Also, a service composer may not readily have the access control policies of all component services from all domains and may need to retrieve the needed policies at the composition time. In case that some policies are protected, the service composer needs to interact with the security authorities of the corresponding domains for remote policy evaluation or negotiation. If such a policy evaluation process is applied to every candidate composition, the cost can be prohibitively high. Note that if policy evaluation is only performed at the execution time, the problem of exploring many potentially illegitimate compositions would occur and the cost would be even higher due to the involvement of actual execution.

1.2 SOLUTION OVERVIEW

This section presents brief overviews of the solutions to the issues discussed in the previous sections. First, we propose a fine-grained yet efficient mechanism for information flow control in service chains (Section 1.2.1) by introducing the concept of transformation factor to model the computation and transformation effects of the intermediate services. Second, we study the data resource level information flow control for web services (Section 1.2.2). As in existing data resource level access control models, access control policies can only be defined over preexisting data resources, we use the information flow tracking technique to determine the source data from which the dynamically generated data are computed from, and enforce their information flow control policies accordingly. Finally, we consider access control at service composition time to minimize the execution-time failure rate of the composed composite services (Section 1.2.3). We

consider the commonly used mediator-based (service composer) approach as well as a fully decentralized composition architecture. In both models, we develop efficient composition protocols to address the performance issue and the concern of restricting policy dissemination in composition-time access control validation.

1.2.1 Fine-grained Information Flow Control Model

As discussed above, to secure critical resources from indirect accesses, existing works either treat the indirectly interacting services exactly the same way as directly interacting services, or consider all possible compositions of intermediate services in policy specification. To facilitate fine-grained information flow control but avoid the prohibitive complexity of policy specification, we introduce the concept of transformation factor and use it to measure the impact of the intermediate services between two indirectly communicating services. The transformation factor specifies how a service computes its input and some locally stored data to generate its output. We define discrete transformation factor levels to estimate the risk of deriving the sensitive information contained in the input or local data of a service from its output. The information flow control policies regarding two services are defined based on the maximal transformation factor of the intermediate services between them.

1.2.2 Data Resource Level Information Flow Control

To secure the usage of the dynamically generated data, the key is to determine the source data from which it is computed. In the service chain $<s_0, s_1, \ldots, s_{n+1}>$, some local data of service s_i, x, is first computed into s_i's output data which is further processed by s_i's subsequent services, s_{i+1}, …, s_{j-1}, and computed into x' which is to be delivered to s_j. For s_i to validate s_j, it is necessary to

determine whether *x'* includes the information contained in *x* and whether this information is in the raw or processed form of the original information contained in *x* (whether *x'* is dependent on *x*). If *x'* is dependent on *x*, then it is necessary for s_i to validate s_j against the information flow control policies defined for *x* when disseminating *x'* to s_j. On the other hand, when service s_i uses a data object *y* to update an entry in its backend database, it is necessary to determine from which source data *y* is computed and then enforce the information flow control policies accordingly. As *y* may be computed from the local data resources of s_i's prior services, s_0, ..., s_{i-1}, say y_0, …, y_{i-1}, the information flow control policies of y_0, …, y_{i-1} may need to be enforced. However, if s_k, $0 < k < i$, computes its output solely based on the input data received from s_{k-1}, then the information flow control policies of y_k need not be enforced. Or, if s_k computes its output solely based on its local data resources or has transformed its input completely, then the information flow control policies of y_0, …, y_{k-1} need not be enforced. Thus, it is necessary to track how information flows inside individual web services and between different services and to determine the source data of the dynamically generated data to make effective information flow control decisions.

We use the information flow tracking technique to determine the source data of the dynamically computed data in a service chain. We first derive the internal data dependencies of web services (intra-service data dependency). Existing information flow tracking mechanisms suffer from a flow-insensitivity issue, i.e., they may not discover all the implicit data dependencies caused by control flows. To overcome this problem, we use the implicit data dependencies detected in previous service invocations to complement the analysis result. Then, we consider a dependency composition mechanism in which the intra-service data dependencies are further computed into inter-service data dependencies. Based on the data dependencies, we

determine the source data of the dynamically generated data and, upon disseminating the data to a service or writing it into the local data storage, decide which information flow control policies to enforce.

1.2.3 Service Composition with Composition-Time Access Control

We consider secure service composition in which the information flow control policies of individual component services are evaluated at composition time to maximize the success rate of the composed composite services.

We first introduce a centralized service composition protocol with a service composer which is not fully trusted by the providers of all services considered in the composition. The protocol includes three composition phases to minimize the composition overhead caused by evaluating numerous information flow control policies. In the first phase, as the search space may be very large, we consider a more efficient but less precise method to quickly evaluate and prune the candidate compositions. We use the information of the historical service composition transactions to estimate the fitness of the candidate compositions, rank them, and select the top candidates. In the second phase, we consider a local policy evaluation process to achieve more precise evaluation of the candidate compositions. In this process, the service composer uses the information flow control policies and/or credentials cached or newly downloaded from the security authorities of involved services to locally validate the candidate compositions. As service composers may not be fully trusted, the accesses to the information flow control policies are considered as special privileges granted by the security authorities rather than assumed. Although this makes it unlikely that the service composer can validate all candidate

compositions, a majority of the service pairs may still be validated, and many invalid candidate compositions can be eliminated. In the third phase, we consider a remote policy evaluation process to further validate the candidate compositions by validating previously unverifiable service pairs. In this process, the security authorities of these services validate their own information flow control policies, and the service composer interacts with them to obtain their information flow control decisions to derive the final composition decision. In this process, negotiation may be frequently needed to exchange the credentials and/or policy information between the service composer and the security authorities and/or between the security authorities in different domains. As the service pairs validated in the second phase need no longer to be validated, this time-consuming process will only be performed on a few service pairs of very few final candidates.

We also consider a distributed service composition protocol without a service composer. In this protocol, each service in the service chain makes its own selection of the subsequent service in the service chain and the selection process of each service includes two phases. In the first phase, the service uses the information of the historical service composition transactions to estimate the likelihood of each candidate service being valid with the information of already composed services in the service chain as the priori, ranks them, and selects the top candidates. In the second phase, the selected candidate services are validated against the information flow control policies of already composed services, and the selection is made. In this phase, we introduce the novel notions of the back-check process and the carry-along policy to facilitate the validation of indirectly interacting services. In the back-check process, the credentials of the candidate services under consideration are delivered to the security authorities of all composed

services and these security authorities evaluate their information flow control policies to validate the candidate services. As the back-check process may significantly degrade the system performance, to minimize the communication overhead, we allow each service *x* to send its information flow control policies to another service *y*, such that *y* can help evaluate the information flow control policies of *x* when composing other services. In this protocol, if a composed service cannot find a valid concrete service for the next service invocation, it will send an adjustment message to its prior services, such that the prior services can revise their selection.

1.3 CONTRIBUTIONS AND ORGANIZATION

This dissertation has analyzed in depth the issues in secure service composition and developed comprehensive solutions to secure the use of web services and the data resources accessed through service interfaces, both at the composition time and at the execution time. The major contributions of this dissertation are summarized as follows.

- We have developed a fine-grained information flow control model using transformation factor to model the impact of intermediate services. We have also designed a semi-automated transformation factor analyzer for individual web services based on the symbolic execution technique.
- We have developed an execution-time data resource level information flow control model which leverages existing information flow tracking technique to analyze the dependencies between the data stored in each domain and the data dynamically generated in service chains. The solution includes a historical approach that achieves flow-sensitive data

dependency analysis for individual services and a scheme that derives inter-service data dependencies from intra-service dependencies.

- We have developed a mediator-based secure service composition mechanism that evaluates the information flow control policies of individual component services at the service composition time to minimize the failure rate of the executions of composite services. The solution includes a three-phase composition protocol with a semi-trusted service composer to address the performance issue and the issue of the trustworthiness of service composers.
- We have also developed a fully distributed service composition protocol without a service composer. We introduce the novel notions of the back-check process and the carry-along policy to facilitate the validation of indirectly interacting services as well as the selection and service composition in a distributed manner.

The remainder of the dissertation is organized as follows. Chapter 2 discusses the related work. Chapter 3 presents the overview of the dissertation, including a general system model and a motivating example. Chapter 4 discusses our fine-grained information flow control model. It formally defines the transformation factor and the information flow control policy. It also presents a semi-automated transformation factor analyzer based on the symbolic execution technique and formally proves the capability and complexity of the information flow control model. Chapter 5 presents the secure service composition mechanism with a semi-trusted service composer. It introduces the three-phase composition protocol to achieve effective and efficient service composition with composition-time access control validation. Chapter 6 presents the distributed secure service composition mechanism. It defines the back-check process and carry-

along policy and shows how individual services interact with each other to compose a service chain in a distributed manner. Chapter 7 presents the execution-time resource-level information flow control system. It discusses a rule-based intra-service data dependency analysis system and an inter-service data dependency composition mechanism. Chapter 8 concludes the dissertation and presents some future research directions.

CHAPTER 2

RELATED WORK

There are several related fields to the issues and mechanisms discussed in this dissertation. First, many web service access control models have been proposed in the last decade to secure web services at the execution time. Some of these models only focus on access control and a few of them also consider information flow control. Second, there have been some research works in recent years that provide secure and trustworthy service composition. As we consider both execution-time and composition-time access control assurance in composite services, these two fields are directly related to this dissertation. The web service access control models are discussed in details in Section 2.1 and the existing security-aware service composition mechanisms are presented in Section 2.2. We also look into some other fields that consider similar issues or provide partial solutions. They include (1) the program level information flow analysis (Section 2.3), including the static information flow analysis (Section 2.3.1) and run-time information flow tracking techniques (Section 2.3.2), (2) the secure interoperation mechanisms that consider securing the cross-domain interactions with classical lattice-based or role-based access control (Section 2.4), and (3) the delegation mechanisms that may provide partial help to securing the interactions between indirectly communicating entities (Section 2.5).

2.1 WEB SERVICE ACCESS CONTROL

Web service access control is a challenging topic which emerged in early 2000s. In recent years, many new models have been proposed to address various issues in providing access control to secure individual web services [AGA04, ARD11, BER04, BER06, BHA04, DAM01, OAS05, OAS07d, PAC11, SKO03, WON04, YUA05, ZHU06]. These issues include the extension of conventional access control models to secure web services, the development of credential-based and attribute-based access control models to secure the multi-institutional web service environment, the use of the state information to help secure the conversational web services, etc. Besides these access control models for individual web services, there are some information flow control models that secure the interactions between different component services in composite services [CHA05, SRI07, YIL07].

2.1.1 Extending Conventional Access Control Models for Web Services

It is necessary to extend conventional access control models developed to secure databases and/or operating systems [BEL73, FER01, LAM74, PAR04, RHA05] to secure the web service environment. In these conventional models, access control policies are usually defined to secure data resources, such as files, relations, etc. However, in the web service environment, it is necessary to develop mechanisms to secure web services and SOAP documents. Providing access control to secure SOAP documents is not straightforward. According to [DAM01], different data resources may be included in a single SOAP document and disseminated. Note that each data resource is represented as an XPath (a path in the tree representation of SOAP). Also, different access control policies may be applied to these different data resources. In order to

provide fine-grained access control for SOAP documents, it is necessary to consider the XPath as the first-class object in the access control model. Many web service access control models consider extending conventional models to secure web services. In [WON04], Wonohoesodo et al. extend the role-based access control by considering web service as a special type of data resource. They consider performing action level access control (at the level of service operations) at the service invocation time and then performing data resource level access control (at the level of data resources accessed within web services). Providing access control to secure web services involves additional issues. First, as indicated in [BER04, BER06], it is desired to provide action-level access control at different protection granularities. On one hand, the domain of the service input may be divided into multiple value ranges and different access control policies may be defined for different value ranges (fine-grained). On the other hand, multiple web services may be grouped into a service class and access control policies may be defined for the service class rather than individual web services (coarse-grained). Second, due to the dynamicity of the web service environment, it is necessary to consider the dynamically changing security requirements. In [BHA04], Bhatti et al. propose to define contextual parameters such as time, location, environmental state, etc., to capture such dynamic requirements. In this model, each access request is associated with a set of contextual parameters and the requester's information is evaluated in conjunction with the access context against the access control policies.

2.1.2 Access Control for Multi-Institutional Web Service Environment

The conventional access control models developed for closed systems should be properly enhanced to secure the web service environment, as the web service environment may involve

multiple administrative domains and many accesses posed by “strangers”. On one hand, conventional identity-based access control (e.g. user/password, etc.) becomes insufficient and it is necessary to support credential-based access control (privileges are included in a certificate). In [AGA04], Agarwal et al. propose to use DAML-S to specify access control lists and includes them in an SPKI/SDSI credential to achieve a credential-based access control. Also, in [ARD11], Ardagna et al. consider the credential-based access control system with the abstraction of complex concepts, such as a set, a disjunction, or a conjunction of concepts, etc., into a single concept in the policy specification. In order to support delegation and re-delegation, they also consider the recursive reasoning on credentials (Suppose that *A* delegates his/her privileges to *B* which are further delegated to *C*. When validating *C*, it is necessary to determine that *C* accesses on the behalf of *A* and, hence, *A*’s privileges rather than *C*’s should be verified.). On the other hand, conventional access control models, such as role-based access control model, multi-level security model, etc., may complicate the interoperation among different domains. In such a multi-institutional environment, a general access control model based on attributes is desired. In [YUA05], Yuan et al. consider the attribute-based access control to secure web services. In this model, the requesters (e.g. users, user applications, etc.), the requested resources (e.g. web services, data resources, etc.), and the access context (e.g. data and time, location, etc.) are all described by a set of attributes and the access control policies are defined over these attributes.

2.1.3 Access Control for Conversational Web Services

The interaction between the client and the web services may involve a set of related service invocations (a conversation) rather than a single one. For example, an editing operation may only be performed after the client performs either new editor registration or login operation. For such

conversations, the client may be at different state at different time and this state information should be considered as a major factor in access control decision making. In case that the mutual validation (the client and the web service validate each other) is necessary, the negotiation based on the state information will be performed. In [SKO03], Skogsrud et al. propose a role-based trust negotiation mechanism for web services. It expresses a trust negotiation policy as a state machine. Each state is associated with a role. A role specifies a set of service operations that can be performed and the credentials that can be disclosed. Each requester can enter new states by revealing his/her credentials or completing certain service operations. In this process, protected resources (service operations and credentials) are gradually disclosed to the requester. In [PAC11], Paci et al. consider the controlled dissemination of the policy information to users in conversational web services (the user requires the policy information of the web service to decide which privilege he/she should use). They model a service as a finite state machine in which, each transition is associated with a service operation and a set of access control policies. At each state *x*, they determine the set of all transitions that may lead to a final state. The set of access control policies associated with all these transitions are the set of policies that can be disseminated to the user at state *x*.

2.1.4 Information Flow Control

There have been a few works that provide preliminary solutions to the information flow control problem [CHA05, SRI07, YIL07]. They can be categorized into two approaches.

The first approach is to validate the interactions between nonconsecutive services in exactly the same way as that of consecutive services [CHA05, YIL07]. For example, in the service chain

<s_0, s_1, s_2, s_3, s_4>, s_0 will validate s_2, s_3, and s_4 as if they are directly interacting with s_0 (i.e. s_1). This approach disregards the computation effects of the intermediate services (s_0 disregards the computation effect of s_2 and s_3 when validating s_4) and, hence, poses overly restrictive constraints (s_0's output may have no correlation with s_4's input), though it is capable of ensuring information flow security. Without proper privileges, the composition of s_2, s_3, and s_4 is prohibited even if s_0's output does not really flow into s_2, s_3, and s_4.

In the second approach, the accesses are considered to be made by a composition of services rather than a single web service [SRI07]. For example, in the above example service chain, s_0 considers that its output is delivered to the service chain <s_1, s_2, s_3, s_4>. When specifying information flow control policies, s_0 needs to consider all the possible compositions of its subsequent services, i.e. s_1, s_2, s_3, and s_4. As can be seen, this approach is almost infeasible, considering its complexity for policy specification.

2.2 SECURITY-AWARE SERVICE COMPOSITION

Existing works in security-aware service composition [BAR06, BAR08, CAR06, DEN03, HAN06, KAG04, PAC08, PAR09] characterize security as a set of attributes that can be quantitatively measured, such as the type of encryption scheme, the type of authentication protocol, trust and reputation, etc. Each service is associated with a set of security properties specified in terms of these attributes. The users and service providers may specify their security constraints, also in terms of these attributes, to ensure the secure use of their services and/or data resources. To achieve secure composition, the service composer ensures that the security properties of all the selected concrete services satisfy all the security constraints specified by the user and/or service providers.

2.2.1 Specification of Security Properties and Constraints

The development of specification languages to represent security properties (capabilities) and constraints is the first research direction in this field. In [DEN03], Denker et al. consider the definition of security ontology in DAML+OIL to facilitate the specification of security properties and constraints. It uses Java Theorem Prover to match security properties with security constraints. It also considers several situations in which different levels of match may be achieved, and suggests using negotiation when an exact match between the security properties and constraints cannot be found. In [KAG04], Kagal et al. propose to use Rei [KAG03] to represent the authorization policies of each service and includes them in the service profile [W3C04] in the registry. On selecting a service, the policies of the service and the attributes of the user are delivered to the Rei reasoner for reasoning and the result is returned to the matchmaker which decides whether to select the service. All these works are standardized into [OAS07c] which uses WS-Policy to specify the security properties and constraints.
In [KAG04], Kagal et al. propose to use Rei [KAG03] to represent the authorization policies of each service and includes them in the service profile [W3C04] in the registry. On selecting a service, the policies of the service and the attributes of the user are delivered to the Rei reasoner for reasoning and the result is returned to the matchmaker which decides whether to select the service.

2.2.2 Security-Aware Composition Protocol

More recent works focus on the development of more sophisticated security-aware service composition protocols by considering additional issues [BAR08, CAR06, HAN06, PAC08]. In

[CAR06], security properties of services are evaluated by a trusted authority and certified by SAML assertions issued by the authority. The validated security properties are stored in the WSDL document of the corresponding service. The security constraints specified by the user are included in the SOAP requests, and are used to prune the candidate services prior to the composition process. The security constraints specified by the service providers are included in the WSDL documents, and are validated when allocating a concrete service to an activity in the workflow. This work uses the AI planning techniques to achieve service selection and composition. In [BAR06, BAR08], Bartoletti et al. map the composition/matchmaking problem into the type system of an enriched λ-calculus. It considers security properties and constraints from the perspective of events. With a predefined set of events, both security properties and constraints can be expressed as a temporally ordered sequence of events. To decide whether a security property matches a security constraint, one only needs to verify whether the order of the events in the security property conforms to the security constraint. However, this work provides very little information about how actual access control policies can be abstracted in this manner. In [PAC08], Paci et al. model each web service as a finite state machine where, each transition arc is associated with a service operation and defined a precondition (an access control policy). They also model each composite service as a finite state machine where, each state includes the states of all its component services and each transition records the invoker/invokee information and the service operation to be invoked. They consider that each service defines a set of access control policies to specify the credentials required to grant the access and a set of credential disclosure policies to specify how a specific credential can be released. The verification of a composition is to verify, for each transition (in the composite service), whether the invoker's

credential disclosure policies comply with the access control policies of the invokee. In [HAN06], Han et al. consider the negotiation in security-aware service composition. The negotiation is performed at a centralized negotiation agent that may be the service composer or may be different. Each service is associated with multiple sets of security properties (they are protected rather than publicly available), each having a preference level. Prior to the negotiation, the services select their most preferable sets of security properties and send them to the negotiation agent. When a mismatch is detected, some services may be prompted to select their less preferable sets of properties to achieve the composition. This negotiation protocol is also applied to the service adaptation when the security constraints posed by the service providers or the user are changed.

2.3 LANGUAGE-LEVEL INFORMATION FLOW CONTROL

Language-based information flow control deals with the secure information flow issue inside a program. Information flow analysis can be performed either statically or in run-time.

2.3.1 Static Information Flow Analysis

Using static program analysis to ensure information flow security was initiated by Denning [DEN77]. In this mechanism, each program variable is labeled with a security class, and all security classes form a security lattice (a directed graph). A directed edge from security classes l_1 to l_2 in the security lattice specifies permitted information flow from l_1 to l_2. It evaluates each program statement to ensure that all the information flows between program variables are legitimate according to the security lattice. In case that the statement is an assignment, there must be a permissible flow from the least upper bound of the security classes of all the variables on the

right hand side to the security class of the variable on the left hand side. Such an information flow is considered as *explicit* flow. If the statement is a conditional construct or a loop, there must be a permissible flow from the least upper bound of the security classes of all the variables appearing in the conditions to the greatest lower bound of the security classes of all the variables appearing in the body of the structure. If all the program statements are valid according to the above rules, the program is secure.

In [GOG82], J. Goguen and J. Meseguer formalize the concept of noninterference (NI) which becomes the basis for many subsequent information flow analysis works. Consider a security lattice with label *low* (public) and *high* (confidential). A program satisfies the NI property if and only if its low outputs are not dependent on its high inputs.

In [VOL96], the information flow analysis is formalized into a security-type system in order to facilitate the soundness proof. A security-type system includes a set of typing rules that assign a security type to the program based on the security types of subprograms.

The above information flow mechanisms support only flow-insensitive analysis, in which, a secure program requires all of its subprograms to be secure. In [JOS00], Joshi et al. propose a flow-sensitive static information flow analysis mechanism by considering a property equivalent to NI. They construct a subroutine *HH* whose informal semantics is "assigning arbitrary values to *h*", where *h* is the set of all high (confidential) variables in program *P*. They show that program *P* is secure if and only if, for any given input, program *HH*; *P*; *HH* and program *P*; *HH* agree on their outputs.

There are other techniques which can also be used for information flow analysis purpose, for instance, program slicing [WEI81] and dependency analysis [AUS92]. The connection between these techniques and information flow analysis is natural. For example, for any variable *u*, *v*, if there is an information flow from *u* to *v*, then *u* must appear in the backward slice of *v*. Also, if there is an information flow from *u* to *v*, there must be a path from the node containing *u* to the node containing *v* in the dependency graph. Note that these techniques can be also used for dynamic information flow analysis, if the program slice or dependency graph is computed during run-time.

2.3.2 Dynamic Information Flow Analysis

There are two common approaches to the run-time information flow analysis. The first approach is through information flow tracking (taint analysis) [LAM06, QIN06, SUH04, VOG07]. The basic idea is to mark the input variables of a program with tags and propagate these tags to the program output. Note that most existing information flow tracking systems are proposed to protect web applications from malicious attacks and, hence, they only mark the input variables from potentially spurious sources (users or other web applications).

In [SUH04], Suh et al. consider 1-bit security tag (0 for authentic, and 1 for spurious) for each register and each 1-byte memory block. These tags are stored in specialized chipsets. The operating system identifies the spurious input channels and initializes the tags (to 1) for the data received from these channels. On running the program, the processor initializes the tag for registers (to 0), and evaluates each executed instruction to propagate the tag to impacted registers or memory locations.

In [QIN06], Qin et al. also consider 1-bit tag (0 for authentic, and 1 for spurious) for each register and 1-byte memory block, but does not design additional hardware to hold the tags. Their approach is based on dynamic binary instrumentation using StarDBT [WAN07] and, hence, they directly use the available memory space instead of designing extra hardware.

The mechanisms in [SUH04, QIN06] are quite primitive against the threats caused by implicit information flows. As they only track a specific execution path of the program, the attackers could leverage the information flows that are not covered by the dynamic analysis to launch attacks. In [VOG07], Vogt et al. consider applying an on-demand static analysis to complement the dynamic information flow tracking when the program execution encounters a conditional construct whose condition part includes spurious data.

In [LAM06], Lam et al. propose a general-purpose information flow tracking system. Unlike the above works, the proposed system is based on source code level instrumentation. That is, it associates each program variable with a 4-byte tag (i.e. a special variable), and creates a set of functions that can be invoked by the application programmers to develop various rules used to control the propagation of tags.

The second approach is through dynamic information flow monitoring [ASK09, SAB10]. In [ASK09, SAB10], an event-based run-time monitoring mechanism is proposed. While evaluating each program statement, the execution engine generates an event which is sent to the monitor. The event records the security classes of all the variables in the statement. The monitor, given a set of rules, decides whether the program execution is secure, based on the events reported.

The above mechanisms can only achieve flow-insensitive analysis and are not able to identify certain implicit information flows caused by control flows [RUS10]. In [GUE06, SHR07], additional mechanisms are incorporated in dynamic analysis to achieve flow-sensitive analysis. [GUE06] considers performing static analyses for the non-executed codes of the program. [SHR07] considers defining program points and analyzing the dependencies between program points to complement the analysis result.

2.4 SECURE INTEROPERATION

Secure interoperation considers the inter-domain access control issue in multi-domain federated environment. The secure interoperation problem has been widely studied in the contexts of lattice-based access control and role-based access control. In order to secure the critical resources of a domain *A* from the access posed by a user in another domain *B*, it is necessary to map the user to a security level in the local security lattice (a partially ordered set of security levels) or a role in the local role hierarchy. For convenience, let H_A and H_B denote the hierarchies (security lattice or role hierarchy) in domain *A* and *B*. To this end, a straightforward solution is to combine the individual hierarchies H_A and H_B into a global hierarchy *H*, by introducing the mappings between the security levels or roles. However, such cross-domain mappings may introduce a cycle in the global hierarchy *H*, enabling a low user in H_B, U_{BL}, to pursue the permissions of a high user in H_B, U_{BH}, which is called the cyclic conflict. To remove such a conflict, it is necessary to remove one of the cross-domain mappings that result in the conflict. Hence, major efforts have been devoted to the resolution of such conflicts.

In [GON96], Gong et al. study the decidability and complexity of the secure interoperation problem, and prove that the problem is NP-complete by introducing a polynomial time reduction of the Feedback Arc Set problem to the secure interoperation problem.

In [BON96], Bonatti et al. consider the secure interoperation problem in the context of lattice-based access control, i.e., given a set of security lattices H_1, …, H_n, and a set cross-domain mappings between security levels, decide whether H_1, …, H_n are combinable (without conflicts). They propose to encode the security lattices and the cross-domain mappings into logical formulae and use logical programming approach to solve the problem.

In [SHA05], Shafiq et al. study the secure interoperation problem in the context of role-based access control. It considers an optimization version of the problem, i.e., given a set of role hierarchies, H_1, …, H_n, and a set of cross-domain role mappings M, select a subset of M, M', such that the composed global role hierarchy H contains no cycles and the total number of allowed cross-domain accesses is maximized. This work casts the original problem (the maximization of total number of cross-domain accesses) into the integer programming (IP) problem by encoding the role hierarchies and role mappings into IP constraint inequalities, and solving it accordingly.

The mechanisms proposed in [BON95, SHA05] require a trusted mediator which has the global view of the federation to generate the global hierarchy. Also, they are both static solutions and suffer from a fairness issue, i.e. such solutions restrict the accesses of users in some domains but do not affect the users in other domains. To address these issues, some mediator-free secure interoperation mechanisms are developed [DAW00, SHE05]. They share the same basic idea but study the problem in different contexts. In [DAW00], Dawson et al. study this problem in the

context of lattice-based access control. In [SHE05], Shehab et al. consider this problem in the context of role-based access control. Consider the role-based access control for example. In [SHE05], they do not perform any role hierarchy composition, but record the path that a user pursues the roles. If the path information may result in a cycle, then the access will be denied. Note that, the roles can be pursued in two ways. First, a user with a superior role can directly enter its immediate subordinate role. Second, a user with role r_A in domain A can enter role r_B in domain B, if there is a inter-domain role mapping from r_A to r_B.

Secure interoperation considers a problem which is orthogonal to ours. It focuses on the interoperability issue when different domains need to collaborate. Although it considers both direct and indirect accesses, it treats them exactly the same way and ignores the impact of intermediate computation units. Also, secure interoperation does not consider the service selection and composition issues.

2.5 DELEGATION

Delegation has been extensively studied in many areas, including role-based access control, workflow management systems, multi-agent systems, grid security, web service, etc. Delegation is the activity that one party hands over part or all of his/her privileges to another party.

As delegation may authorize someone the accesses to the resources that he/she is originally not entitled to, it needs to be carefully controlled. First, it is necessary to develop suitable mechanisms to specify what privileges a user can delegate and can be delegated [CRA08a, HWA06, MAB08], and when delegation of privileges can be made (e.g. in a certain time frame [WAN06], when certain tasks are delegated, etc.). Second, as trust relationship is generally non-transitive (x trusts y and y trusts z does not imply that x trusts z), it is necessary to control the

scope of delegation. This can be achieved by (1) defining an upper bound for the depth of delegation [WAN06] and/or (2) specifying re-delegation as a privilege that can be granted [WAI07]. Third, it is also desired to specify other constraints to further control the delegation. For example, separation-of-duty constraints may be enforced to ensure that some mutually exclusive privileges are not delegated to the same person. Fourth, it is necessary to ensure the least privilege principle in delegation model, i.e. only the required privileges should be delegated. Some researchers suggest that, when delegated a task, the delegatee requests the delegator for required privileges [AHS09, CRA08b]. This approach, although possible to ensure the least privilege principle (if assuming the delegatee is honest), may result in high overhead, as the delegatee may have to request the delegator multiple times, i.e. when he/she has insufficient privileges to continue with the processing.

Delegation may be used to provide partial solutions for information flow control in service chains. Consider service chain $<s_0, s_1, s_2, s_3, s_4>$. When s_0 has the privilege for accessing s_4 but s_1, s_2, and s_3 do not, s_0 can delegate its privileges to s_1 and further to s_2 and s_3 to enable their accesses. However, even with a proper control of delegation and re-delegation, using the delegation alone is still not sufficient in handling the information flow control problem in service chains. Delegation cannot help when s_3 has the privilege for accessing s_4 but s_0, s_1, and s_2 do not have the privilege, or when s_0 is willing to disseminate its sensitive information to s_1 but not to s_2, s_3, and s_4.

CHAPTER 3

SYSTEM MODEL AND MOTIVATING EXAMPLE

The focus of this dissertation is on the information flow control issue in service chains in multi-domain environment. In Section 3.1.1, we present the system model for the multi-domain web service environment. We also define the service chain model independent of the domains, but focus on the information flows in service chains. The service chain definition is given in Section 3.1.2. We present a motivating example in Section 3.2 to demonstrate the need for performing information flow control.

3.1 SYSTEM MODEL

3.1.1 Web Service System

We consider a general web service system (Figure 3.1), which consists of multiple security domains and multiple service composers. Each security domain includes a set of web services, a set of data resources, and a security authority (SA) which manages a set of access control policies to control the accesses to the services and data resources in the domain. The web service system is defined in Definition 3.1.

Definition 3.1. A web service system includes a set of security domains $\{d_1, d_2, \ldots\}$, and a set of service composers $\{c_1, c_2, \ldots\}$. Each domain d_i is a tuple $<d_i.U, d_i.S, d_i.R, d_i.SA>$ where,

- $d_i.U = \{d_i.u_1, d_i.u_2, \ldots\}$ is the set of all users in domain d_i.

- $d_i.S = \{d_i.s_1, d_i.s_2, \ldots\}$ is the set of all web services in domain d_i.
- $d_i.R = \{d_i.r_1, d_i.r_2, \ldots\}$ is the set of all data resources in domain d_i.
- $d_i.SA$ is the security authority of domain d_i which manages a set of access control policies $d_i.Pol = \{d_i.pol_1, d_i.pol_2, \ldots\}$ to control the accesses to $d_i.S$ and $d_i.R$. □

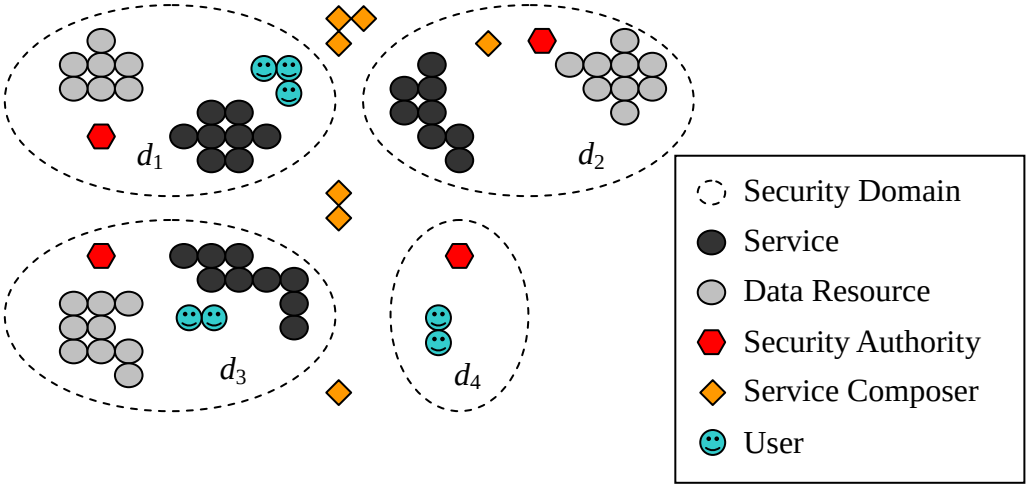

Figure 3.1. Web service system.

Data resource refers to the data/information itself and any entity that may store or receive data/information. Such an entity can be a data container, such as a file, a directory, a relation, a view, etc., or a physical resource, such as a printer, a scanner, the disk space, the CPU cycle, bandwidth, etc. A web service is a piece of software with an interface described in a machine-readable format. Web service can be viewed as a special type of data resource and protected accordingly. Moreover, a web service, upon being invoked, may read/write certain data resources. We assume that all services are semi-honest. They follow the protocol and conform to the access control policies, but some services may attempt to derive the sensitive information of

others from the information they have received. Also, we do not consider the interoperability issues. There are techniques in the literature that can help resolve these issues [NAT04].

For convenience, we use *dom*(*s*) to represent the domain of service *s*. Also, we use *Pol*(*x*) to represent the set of all access control policies defined for *x* where, *x* is a service or a data resource.

3.1.2 Service and Service Chain

We consider that each service reads in a set of input data resources and writes out a set of output data resources. The set of input data resources of a service *s* includes all the data resources that *s* receives from another service or a user and may include the data resources in *dom*(*s*) that *s* reads from in the computation of *s*. The set of output data resources of *s* includes all the data resources that *s* sends to another service or a user and may include the data resources in *dom*(*s*) that *s* writes to in the computation of *s*. As we only consider the deterministic system, the set of output data resources of a service can be expressed as a function of the set of its input data resources. We formally define the web service in Definition 3.2.

Definition 3.2. A web service *s* is a tuple <*s.In*, *s.Out*, *s.F*> where,

- *s.In* is the set of all input data of *s*. $s.In = s.In_L \cup s.In_F$ where,
  - $s.In_L = \{s.in_{L1}, s.in_{L2}, ...\}$ is the set of all input data resources in *dom*(*s*) that *s* reads from in its computation.
  - $s.In_F = \{s.in_{F1}, s.in_{F2}, ...\}$ is the set of all input data resources that *s* receives from another service or a user.
- *s.Out* is the set of all output data of *s*. $s.Out = s.Out_L \cup s.Out_F$ where,

- $s.Out_L = \{s.out_{L1}, s.out_{L2}, ...\}$ is the set of all output data resources in $dom(s)$ that s writes to in its computation.
- $s.Out_F = \{s.out_{F1}, s.out_{F2}, ...\}$ is the set of all output data resources that s sends to another service or a user.

- $s.F$ is the computation function of s where, $s.Out = s.F(s.In)$. □

A service may have multiple service operations, implementing different functionalities. Hence, a service may have multiple computation functions. For simplicity, we assume that all services are atomic, i.e. containing only one computation function.

Generally, a composite service can be defined using a workflow, which is a composition of component services. We consider abstract and concrete workflows. In an abstract workflow, each component service is abstract and is to be grounded to a concrete service. In a concrete workflow, each component service is a concrete web service. Upon composition, the service composer is given the desired abstract workflow and instantiates each abstract component service by a concrete service. In this dissertation, we only consider a simplified workflow, a service chain. The simplification is for the convenience in defining the notations and algorithms. The solutions provided in this dissertation are applicable to general workflows with parallel composition and loop. We define the abstract and concrete service chains in Definition 3.3.

Definition 3.3. An abstract service chain $<s_0, as_1, \ldots, as_n, s_{n+1}>$ consists of two end users, s_0 and s_{n+1}, where s_0 is the user who sends the input data to as_1 and s_{n+1} is the user who receives the output data from as_n, and a sequence of abstract services, $as_1, \ldots, as_n$, that should be grounded to

concrete services. A concrete service chain $<s_0, s_1, \ldots, s_n, s_{n+1}>$ consists of the two end users, s_0 and s_{n+1}, and a sequence of concrete services $s_1, \ldots, s_n$. □

For convenience, we consider a user as a service. Also, in service composition, only the selection of $as_1, \ldots, as_n$ will be considered. However, when evaluating the validity of a concrete service chain $<s_0, \ldots, s_{n+1}>$, both the two end users, s_0 and s_{n+1}, and all the services, $s_1, \ldots, s_n$, will be considered.

3.2 MOTIVATING EXAMPLE

We consider an example application workflow (Figure 3.2) to motivate demonstrate the need for the information flow control in service composition and the benefit of considering access control at composition time. It is also used as a running example it later on to help to illustrate various concepts in our model. The workflow is used to help with screening of disease *x* by first extracting association rules from medical data of patients with and without disease *x*. The association rules are then used to determine how likely a new patient does have disease *x*. The workflow consists of the following abstract services, a client program *CLN*, a medical database *MDB*, a template image database *TDB*, an image enhancement service *IES*, an image registration service *IRS*, an object recognition service *ORS*, an association rule mining service *ARM*, and a classifier *CLS*. *CLN* first searches *MDB* (with keyword *x*) for the medical records of the patients who are diagnosed to have the disease *x*, and searches *TDB* (with keywords such as "bone", "polyp", "nodule", etc.) for the template images for object recognition. Each medical record stored in *MDB* includes the alphanumeric medical data, e.g. the patient's medical history, family history, personal data (e.g. gender, age, height, weight, living area, etc.), and the medical images

(e.g. CT, X-ray, Nuclear, etc.). The alphanumeric medical data of *MDB* are sent to *ARM*. The template images are sent to *ORS*. The medical images are first sent to *IES* which performs image enhancement (e.g. noise cancellation, etc.). The enhanced images are sent to *IRS* which performs image registration to align different images into one coordinate system. The aligned images are sent to *ORS* which detects and recognizes the objects in the images (e.g. bones, polyps, nodules, etc.) using the template images. After recognition, it assigns labels to the recognized objects in the image. The labeled images are sent to *ARM*, which uses these images together with the alphanumeric medical data received from *MDB* to extract association rules (in the form of $(y_1, \ldots, y_n) \Rightarrow x$, where y_i, $1 \leq i \leq n$, is an object label or a string extracted from the alphanumeric medical data and x is the disease name) which are sent to *CLS*.

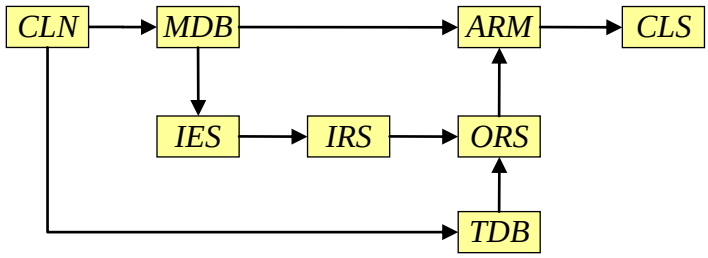

Figure 3.2. Example workflow.

Now consider the sensitivity of the data that are used by the composite service (Note that this abstract composite service includes three abstract service chains). We assume that the search keywords that *CLN* sends to *MDB* and *TDB* are not sensitive and, hence, require no protection. The alphanumeric medical data that *MDB* sends to *ARM* and the medical images that *MDB* sends to *IES* are sensitive and the recipients are required to have read permissions to these data. (Note that the recipient rather than the invoker needs to have the proper privilege. For example, *IES* needs to have the read permission to the medical images in *MDB*, but *CLN* does not have to.)

The template images are used in a pay-per-use manner and, hence, require the recipients to present proper privilege.

Next, consider the concrete services that can be used to instantiate the abstract services and the privileges they have (Figure 3.3). For simplicity, we assume that *CLN*, *MDB*, *TDB*, *IES*, *IRS*, and *CLS* are already concretized by cln_1, mdb_1, tdb_1, ies_1, irs_1, and cls_1, respectively. cln_1 and cls_1 are hosted by hospital *A* (domain d_A). mdb_1 and tdb_1 are hosted by hospital *B* (domain d_B) and research institute *C* (domain d_C), respectively. ies_1 and irs_1 are hosted by research institute *D* (domain d_D). *ORS* can be instantiated by ors_1, ors_2, and ors_3. Note that ies_1 does not modify the content of the medical image received from mdb_1 and irs_1 does not modify the content of the images received from ies_1. Hence, the medical images of mdb_1 are essentially delivered to the *ORS* service (ors_1, ors_2, or ors_3) in their raw forms. *ARM* can be instantiated by arm_1 and arm_2. We consider that ors_1 and arm_1 are hosted by institute *D*, ors_2 is hosted by research institute *E* (domain d_E), and ors_3 and arm_2 are hosted by university *F* (domain d_F).

For simplicity, we assume that all the services can be invoked by anyone and we only define the resource-based access control policies here. Consider that service *x* invokes service *y*. If *y* does not read/write any sensitive local data (e.g. a table, etc.) in its computation, then the invocation can be directly granted. If *y* reads some sensitive local data resources *r*, then the invocation is granted when *x* has the read permission to *r*. Similarly, if *y* writes to *r*, then the invocation is granted when *x* has the write permission to *r*. Figure 3.3 depicts the resource access rights. We consider that institutes *D* and *E* are federated with hospital *B* and, hence, services ies_1, irs_1, ors_1, arm_1, ors_2 have the read permission to the medical data in domain d_B. Also, we consider that institute *E* has purchased the service of tdb_1 from institute *C* and, hence, ors_2 has

the read permission to the template images in domain d_C. No other read/write accesses to the medical data or the template images are allowed.

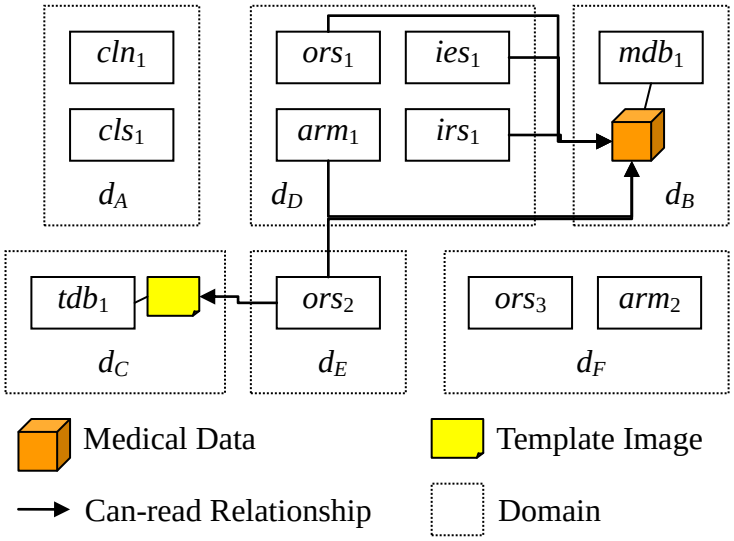

Figure 3.3. Concrete services and their permissions.

Information flow control illustration. Conventional access control models cannot address the security needs in this application. For example, the medical images of mdb_1 are delivered to the *ORS* service in raw forms and ors_3 does not have the read permission for the medical images of mdb_1, the selection of ors_3 should be prohibited. Also, as arm_2 does not have the privilege for the template images in tdb_1, arm_2 should not be selected either. However, conventional models do not consider information flow control and will allow *ORS* to be grounded to ors_3 or *ARM* to be grounded to arm_2, resulting in undesirable information flows.

Benefit of composition-time access control validation. Assume that access control is not validated at composition time. Suppose that the workflow (Figure 3.2) is first concretized by the composite services {cln_1, mdb_1, tdb_1, ies_1, irs_1, ors_3, arm_2, cls_1}. During execution, cln_1, mdb_1, and tdb_1 are invoked and executed successfully. But when mdb_1 sends out the alphanumeric

medical data to arm_2, since arm_2 does not have read permission to the alphanumeric medical data of mdb_1, mdb_1 cannot send its output to arm_2 and the execution fails. All the execution of cln_1, mdb_1, and tdb_1 are wasted. After this failure, the execution coordinator needs to pass the execution information to the service composer and the composer records the access control violation between mdb_1 and arm_2 and then replaces arm_2 with arm_1. The execution of the new composite service $\{cln_1, mdb_1, tdb_1, ies_1, irs_1, ors_3, arm_1, cls_1\}$ also fails as ors_3 has neither the read permission to the medical images of mdb_1 nor the permission to the template image of tdb_1. Again, cln_1, mdb_1, ies_1, irs_1, and tdb_1 have completed their execution and the efforts are wasted. Such failure may continue until the only feasible composition $\{cln_1, mdb_1, tdb_1, ies_1, irs_1, ors_2, arm_1, cls_1\}$ is selected and executed.

If access control validation is performed at composition time, then the selected composition is likely to succeed, avoiding wasting unsuccessful execution efforts. (Note that there may still be policies that can only be evaluated at execution time, which can still results in failures).

CHAPTER 4

INFORMATION FLOW CONTROL MODEL

As discussed in Section 1.1.1, most existing web service access control models do not consider the information flow control issue in composite web services. The current solutions to the information flow control problem are quite primitive. They either treat direct and indirect interactions exactly the same way and disregard the computation effects of intermediate services or require the exhaustive enumeration of all possible compositions of intermediate services for policy specification. In order to provide information flow control but avoid the pitfalls in existing works, we introduce the novel concept of transformation factor to model the computation and "transformation" effect of the intermediate services and consider it as a major factor in making information flow control decisions.

Based on transformation factor, we develop a fine-grained information flow control model to allow the services in a service chain to effectively and efficiently control the information flows from/to their critical resources. The model will be used as a basis for further development of secure service composition protocols. In Section 4.1, we first introduce the attribute-based access control (ABAC) model. Our information flow control model discussed in this dissertation is built on the attributed-based model. In Section 4.2, we formally define the transformation factor and present a semi-automated transformation factor analysis mechanism. In Section 4.3, we define the information flow control policy, which specifies how to control the dissemination of sensitive

information to other services (both directly and indirectly interacting services) and to control the write to critical data resources with potentially corrupted information. In Section 4.4, we study the capability and complexity of our information flow control model and compare it with conventional access control models.

4.1 ATTRIBUTE-BASED ACCESS CONTROL

We consider a general attribute-based access control model [HEB09, WAN04, YEH11, YUA05]. Each service or data resource is associated with a set of attributes. The attributes of a service may include, but not limited to, the service name, WSDL location, the permissions granted to the service, trust and reputation, clearance level, transformation factor, etc. The attributes of a data resource may include, but not limited to, the owner, security classification, data type, size, etc. For simplicity, we define a unified set of attributes for the services and data resources in all security domains. The attributes of a data resource or a service (when the service is being invoked) are included in the metadata stored together with the data resource or service. Before a service x can access another service or a data resource y, the attributes of x must first be asserted by a security authority z (e.g. using SAML assertion [OAS09]) trusted by $dom(y).SA$. The asserted attributes of x are included in a certificate, called the *attribute certificate*, and signed by the issuer (z) for authentication and non-repudiation purpose. The attribute certificate of x is then presented to $dom(y).SA$. $dom(y).SA$ extracts the attributes of x from the certificate and evaluates the attributes of x and the attributes of y against the access control policies that are applicable to y. The attribute and attribute certificate are formally defined in Definition 4.1.

Definition 4.1. Each service or data resource x is associated with a set of attributes $Attr(x) = \{attr_1(x), attr_2(x), \ldots\}$. Each attribute $attr(x)$, $attr(x) \in Attr(x)$, is defined as a tuple ($attr(x).name$, $attr(x).value$) where, $attr(x).name$ is a string that uniquely specifies the name of the attribute and $attr(x).value$ is the value of the attribute.

Each service owns a set of attribute certificates $s.AC = \{s.ac_1, s.ac_2, \ldots\}$. Each attribute certificate $s.ac$, $s.ac \in s.AC$, is issued by a security authority to s certifying that s owns certain attributes $s.ac.Attr(s)$, where $s.ac.Attr(s) \subseteq Attr(s)$. □

We consider that the security authority in each domain manages the attribute certificates of all services in the domain. The attributes of a service can be sensitive or non-sensitive. Attribute certificates containing only non-sensitive attributes can be freely exchanged among different parties. Attribute certificates containing sensitive attributes requires protection.

Note that attribute-based access control model can be used to instantiate various types of access control systems. For example, by defining clearance levels for users and services and security classes for data resources, a multilevel security system is instantiated. By defining roles for users and services and mapping data resources to permissions, a role-based access control system is instantiated.

Figure 4.1 shows an example attribute certificate describing a service whose subscription type is “regular”, expiration date is “12/31/2012”, and the penalty is 0.05. Figure 4.2 shows an example attribute-based access control policy which specifies that the request is permitted if the subscription type of the requester is “regular”, the expiration date of the requester’s subscription is before “12/31/2015”, and the penalty of the requester is less than or equal to 0.1.

```
<AttributeCertificate>
   <AttributeStatement>
      <Attribute>
         <AttributeName value = “subscriptionType”/>
         <AttributeValue value = “regular” type = “string”/>
      </Attribute>
      <Attribute>
         <AttributeName value = “expirationDate”/>
         <AttributeValue value = “12/31/2012” type = “date”/>
      </Attribute>
      <Attribute>
         <AttributeName value = “penalty”/>
         <AttributeValue value = “0.05” type = “float”/>
      </Attribute>
   </AttributeStatement>
</AttributeCertificate>
```

Figure 4.1 Example attribute certificate.

```
<Policy>
   <Rule id = “01” version = “02” decision = “permit”>
      <Operator value = “and”>
         <Operator value = “equalTo”>
            <AttributeName value = “subscriptionType”/>
            <AttributeValue value = “regular” type = “string”/>
         </Operator>
         <Operator value = “lessEqualTo”>
            <AttributeName value = “expirationDate”/>
            <AttributeValue value = “12/31/2015” type = “date”/>
         </Operator>
         <Operator value = “lessEqualTo”>
            <AttributeName value = “penalty”/>
            <AttributeValue value = “0.05” type = “float”/>
         </Operator>
      </Operator>
   </Rule>
</Policy>
```

Figure 4.2. Example attribute-based access control policy.

4.2 TRANSFORMATION FACTOR

4.2.1 Multi-level Transformation Factor

According to [CHA05, YIL07], in order to validate the service chain $<s_0, s_1, \ldots, s_{n+1}>$, it is necessary to perform mutual validation for each pair of services (s_i, s_j), $0 \leq i \leq n$, $i < j \leq n+1$. To validate the pair of services (s_i, s_j), s_i needs to ensure that s_j has the read permission to $s_i.In_L$ and s_j needs to ensure that s_i has the write permission to $s_j.Out_L$. However, there are several special situations under which the validation between some pairs of services can be skipped. First, service s_k, $0 < k \leq n$, may receive some sensitive data, $s_k.In_F$, from s_{k-1}, but after s_k's computation, the sensitive information contained in $s_k.In_F$ may no longer be derivable from the data that s_k sends to s_{k+1}, $s_k.Out_F$. In this case, the mutual validation between any prior services of s_k, s_i, $0 \leq i < k$, and any subsequent services of s_k, s_j, $k < j \leq n+1$, can be fully ignored. Second, if the sensitive data that s_k reads from its local storage, $s_k.In_L$, is not derivable from its output, $s_k.Out_F$, s_k does not need to validate all its subsequent services, s_j, $k < j \leq n+1$, on whether s_j has the read permission to $s_k.In_L$. Third, if the information that s_k writes to its local data resources, $s_k.Out_L$, is unrelated to its input $s_k.In_F$, s_k does not need to validate all its prior services, s_i, $0 \leq i < k$, on whether s_i has the write permission to $s_i.Out_L$. Moreover, in the applications with lower security requirements, there are situations under which a certain degree of information leakage may be allowed to trade for the functionality or performance. Therefore, it is necessary to model the computation and transformation effects of the intermediate services. We introduce the multi-level transformation factor (TF) to model how a service processes its input data to generate the output, so as to estimate the risk of deriving the sensitive information contained in the input from the output. The transformation factor of a service is formally defined in Definition 4.2.

Definition 4.2. The transformation factor of service *s*, *tf*(*s*) specifies how much transformation *s* makes when generating the output data of *s*, *s.Out* from the input data of *s*, *s.In*. *tf*(*s*) also measures how likely the information contained in *s.In* can be derived from *s.Out*. □

A service may have multiple execution paths for different input parameters. These execution paths may implement different functionalities and, hence, may have different transformation factors. However, for simplicity, we assume that each computation function only has a single transformation factor, which is the worst-case estimation over all possible execution paths. Table 4.1 lists four transformation factor levels in our model. Note that HR < MR < LR < LR.

Table 4.1. Transformation factor levels.

TF Level	Description
NR (No risk)	It is impossible to derive the sensitive information contained in the input of a service from its output.
LR (Low risk)	It is difficult to derive the sensitive information contained in the input of a service from its output.
MR (Moderate risk)	Some of the sensitive information contained in the input of a service is derivable from its output.
HR (High risk)	A majority or all of the sensitive information contained in the input of a service is derivable from its output, or the output of a service contains the raw information in its input.

4.2.2 Automated Transformation Factor Analysis

We design an automated transformation factor analyzer to help decide the transformation factors of the web services with alphanumeric input/output. The main idea is to use symbolic execution [KHU03, KIN76] techniques to derive the relations between the input and output of the service and assign transformation factor accordingly.

Symbolic execution is a technique widely used for software testing and verification. It uses symbols instead of actual data as the program input to derive the output. Consequently, the program output and all the internal variables are represented as expressions of the input symbols. In many cases, there may be infinite possibilities of program execution due to nondeterministic loops or dynamic data structures and, hence, exhaustive symbolic execution is not possible. Some symbolic executors take multiple input patterns or multiple samples of loop iterations to generate the output expressions. Here, we first consider a single input pattern and then extend it to multiple input patterns.

Consider service *s*. We first attach a sensitivity label to each input symbol $s.in_i \in s.In$ to indicate whether the data is sensitive. The transformation factor analysis only considers the sensitive data. All the symbols without a sensitivity label will be ignored. Note that we assume that each output data item $s.out_j$ in *s.Out* is a basic data type. The reason is that, if an output data item is not a basic data type, we can decompose it recursively into elements of basic data types and treat each as one output data item. For example, each entry of an array is considered as a single data item. We define rules for transformation factor analysis and the analysis process is shown in Figure 4.3.

Input: $s.In$, $s.Out$, τ ($0 \leq \tau \leq 1$), Output: $tf(s)$.
1. For each output data $s.out_j$ in $s.Out$,
   a) Obtain f_j and $s.In_j$, $s.In_j \subseteq s.In$, s.t. $s.out_j = f_j(s.In_j)$.
   b) Compute $r(s) := n_s/n_a$, where n_s is the number of sensitive symbols in $s.In_j$ and n_a is the number of sensitive symbols in $s.In$.
   c) If $s.In_j$ does not include any sensitive symbols then set $tf_j(s)$ to NR.
   d) If f_j is multiplicative and $s.In_j$ includes sensitive symbols, then set $tf_j(s)$ to HR.
   e) If f_j is additive and $r(s) > \tau$, then set $tf_j(s)$ to MR.
   f) If f_j is additive and $r(s) \leq \tau$, then set $tf_j(s)$ to LR.
2. Set $tf(s)$ to $min_j\{tf_j(s)\}$.

Figure 4.3. Transformation factor analysis.

From cryptography, if x is the product of $x_1, x_2, \ldots, x_n$, then it is possible to guess x_i by factorization. On the other hand, if x is the sum of $x_1, x_2, \ldots, x_n$, then it is very difficult to decompose x and get information of individual x_i, unless the domain of x and/or some x_i is very small. Thus, we assign higher risk (lower transformation factor) to s, if the computation function of s is multiplicative.

The process above considers only one input pattern. If there are multiple input patterns as discussed above, then there will be multiple versions of $s.out_j$, for each j. Let $s.out_j^k$ denote the output data item $s.out_j$ corresponding to the k^{th} input pattern. Each $s.out_j^k$ can be analyzed in the same way as stated above. Let $tf^k(s)$ denote the derived transformation factor for the k^{th} input pattern. The overall transformation factor can be derived as $tf(s) = min_k\{tf^k(s)\}$.

4.2.3 Manual Transformation Factor Analysis

For some services (e.g. alphanumeric input/output, the local data access is static), static program analysis (e.g. symbolic execution) may help determine their transformation factors. However, in

case that a service takes in or generates non-alphanumeric data (e.g. image, audio, video, etc.), the transformation factor may only be determined based on its functionality. In this case, we consider that, in each domain, a security officer decides the transformation factors of services based on the functionality of the services. We consider several example services in the example system in Section 3.2 to illustrate the determination of the transformation factors of some services.

The output data of some services may contain the raw data of their input. The image enhancement service ies_1 takes in a set of input images, removes the noise in them, and reconditions them for object recognition. Since the enhanced images contains the raw data in the original image, we have $tf(ies_1)$ = HR.

In some cases, part of the sensitive information contained in the input may be derivable from its output. The search engine mdb_1 searches for the medical records of the patients diagnosed to have a disease x. Though the search keyword x may not be directly seen in the search result, it may be possible to guess the value of x from the common information in these records. If the likelihood of making a successful guess (e.g. when the output always contains very few records) is very low, then the transformation factor can be set to LR. If the likelihood is moderate, then the transformation factor is set to MR.

In some other situations, the sensitive information contained in the input of a service cannot be derived from its output. The association rule mining service, arm_1 or arm_2, takes in a set of medical images with object labels and the associated alphanumeric medical data and derives association rules (an association rule is in the form $(x_1, \ldots, x_n) \Rightarrow y$, where $x_1, \ldots, x_n$ are object

names and y is the disease name). As it is impossible to derive the sensitive information, such as the patient medical history, etc., from the output association rules, its transformation factor can be set to NR.

4.3 INFORMATION FLOW CONTROL POLICY

Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. On one hand, each service s_i, $0 \leq i \leq n$, may specify detailed policy to decide whether the information retrieved from s_i's local input data resource, $s_i.In_L$, can be disseminated to the subsequent service s_j, $i < j \leq n+1$. On the other hand, each service s_j, $0 < j \leq n+1$, may specify detailed policy to decide whether the data potentially computed from s_i's local input data ($0 \leq i < j$) can be written to the local data resource, $s_j.Out_L$. Such a policy is called the *information flow control* (IFC) policy. In the first case, service s_i defines the *reader* IFC policy, and evaluates the attributes of the reader service s_j, $Attr(s_j)$, the attributes of s_i's local input data resource $s_i.In_L$, $Attr(s_i.In_L)$, and the maximal transformation factor of the intermediate services, $max\{tf(s_i), \ldots, tf(s_{j-1})\}$, against s_i's reader IFC policy. In the second case, s_j defines the *writer* IFC policy, and evaluates the attributes of the writer service s_i, $Attr(s_i)$, the attributes of s_j's local output data resource $s_j.Out_L$, $Attr(s_j.Out_L)$, and the maximal transformation factor of the intermediate services, $max\{tf(s_{i+1}), \ldots, tf(s_j)\}$, against s_j's writer IFC policy. We formally define the information flow control policy in Definition 4.3.

Definition 4.3. The reader information flow control policy of service s_i specifies how to evaluate a reader service s_j, $i < j \leq n+1$. The reader IFC policy specifies a set of conditions over the attributes of s_j, $Attr(s_j)$, the attributes of s_i's local input data resource $s_i.In_L$, $Attr(s_i.In_L)$, and

the maximal transformation factor of the intermediate services between s_i and s_j, $max\{tf(s_i), \ldots, tf(s_{j-1})\}$.

The writer information flow control policy of s_j specifies how to evaluate a writer service s_i, $0 \leq i < j$. The writer IFC policy specifies a set of conditions over the attributes of s_i, $Attr(s_i)$, the attributes of s_j's local output data resource $s_j.Out_L$, $Attr(s_j.Out_L)$, and the maximal transformation factor of the intermediate services between s_i and s_j, $max\{tf(s_{i+1}), \ldots, tf(s_j)\}$. □

We consider the information flow control decision to be *true*, *false*, or *unknown*. We use $Pol_R(s)$ and $Pol_W(s)$ to represent the reader and writer IFC policies of service s, i.e. the reader and writer information flow control policies defined for s and for all the data resources that may be read/written in the computation of s.

We present several example information flow control policies in Figure 4.4. For simplicity, we consider that each data resource r is defined a security class $sc(r)$ which measures the sensitivity level of r. Each service s is defined a clearance level $cl(s, d)$ by the security authority in each domain d, which specifies the maximal security class of the data that s can access in domain d.

IFC_1.	$(cl(s_j, dom(s_i)) < sc(s_i.In_L))$ $\Rightarrow (auth(Attr(s_j), Attr(s_i.In_L), max\{tf(s_i), \ldots, tf(s_{j-1})\}, Pol_R(s_i)) := false)$.
IFC_2.	$(\lvert(cl(s_j, dom(s_i)) - sc(s_i.In_L)\rvert \leq 1) \wedge (max\{tf(s_i), \ldots, tf(s_{j-1})\} \leq \mathrm{LR})$ $\Rightarrow (auth(Attr(s_j), Attr(s_i.In_L), max\{tf(s_i), \ldots, tf(s_{j-1})\}, Pol_R(s_i)) := true)$.
IFC_3.	$(cl(s_i, dom(s_j)) \neq sc(s_j.Out_L))$ $\Rightarrow (auth(Attr(s_i), Attr(s_j.Out_L), max\{tf(s_{i+1}), \ldots, tf(s_j)\}, Pol_W(s_j)) := false)$.
IFC_4.	$(\lvert(cl(s_i, dom(s_j)) - sc(s_j.Out_L)\rvert \leq 1) \wedge (max\{tf(s_i), \ldots, tf(s_{j-1})\} \leq \mathrm{LR})$ $\Rightarrow (auth(Attr(s_i), Attr(s_j.Out_L), max\{tf(s_{i+1}), \ldots, tf(s_j)\}, Pol_W(s_j)) := true)$.

Figure 4.4. Example information flow control policies.

IFC_1 is a reader IFC policy that ensures no information flow from a local input data resource to a reader service whose clearance level is below the security class of the data.

IFC_2 is a reader IFC policy specifying that, if the difference between the security class of the local input data resource and the clearance level of the reader service is less than or equal to 1 and the maximal transformation factor of intermediate services is LR, the access will be permitted.

IFC_3 is a writer IFC policy ensuring that the clearance level of the writer service must be equivalent to the security class of the local output data resource.

IFC_4 is a writer IFC policy specifying that, if the difference between the security class of the local output data resource and the clearance level of the writer service is less than or equal to 1 and the maximal transformation factor of intermediate services is LR, the access will be permitted.

Here, $auth(Attr(s_j), Attr(s_i.In_L), max\{tf(s_i), \ldots, tf(s_{j-1})\}, Pol_R(s_i))$ denotes the information flow control decision made by evaluating the attributes of the reader service s_j, $Attr(s_j)$, the attributes of s_i's local input data resource $s_i.In_L$, $Attr(s_i.In_L)$, and the maximal transformation factor of intermediate services, $max\{tf(s_i), \ldots, tf(s_{j-1})\}$, against the reader IFC policy of s_i, $Pol_R(s_i)$. $auth(Attr(s_i), Attr(s_j.Out_L), max\{tf(s_{i+1}), \ldots, tf(s_j)\}, Pol_W(s_j))$ denotes the information flow control decision made by evaluating the attributes of the writer service s_i, $Attr(s_i)$, the attributes of s_j's local output data resource $s_j.Out_L$, $Attr(s_j.Out_L)$, and the maximal transformation factor of intermediate services, $max\{tf(s_{i+1}), \ldots, tf(s_j)\}$, against the writer IFC policy of s_j, $Pol_W(s_j)$.

4.4 MODEL ANALYSIS

In this section, we analyze the capability and complexity (i.e. total number of information flow control decisions to be made) of the information flow control model. We first show that the conventional access control model is a special case of the IFC model. That is, through proper configuration, the IFC model can be equivalent to the conventional access control model. Then, we show that there are cases where the IFC model can provide desired information flow control in service chains while the conventional model cannot. The validation of a service chain generally involves the generation of multiple information flow control decisions and the combination of these decisions. For examples, under conventional access control models, the validation of a length-n+2 service chain requires $2\cdot(n+1)$ information flow control decisions (two decisions for each pair of services (s_i, s_{i+1}), for all i, $0 \leq i \leq n$). To achieve information flow control, the IFC model may require the generation of more information flow control decisions, and this may incur additional overhead, such as additional message exchanges. We analyze the IFC model and derive its worst-case complexity in terms of the total number of decisions required. The complexity of the IFC model can be reduced when it is only required to achieve conventional access control in service chains. We show that, in this case, the IFC model has the same complexity as conventional access control model.

To facilitate the analysis, we define an abstract model, P_C, which is generalized from conventional access control models. As we focus on the information flow control issue in service chain, we define P_C on the basis of a service chain. The formal definition of P_C is given in Definition 4.4.

Definition 4.4. P_C is an attribute-based access control model defined for a length-n+2 service chain $<s_0, s_1, \ldots, s_{n+1}>$. Let $Attr(x)$ denote the set of attributes of a service or a data resource x. Let $Pol_R(s_i)$ denote the reader policy of s_i on disseminating some of its local data resources to a reader service and $Pol_W(s_j)$ denote the writer policy of s_j on a writer service writing some of s_j's local data. Let $auth(Attr(s_j), Attr(s_i.In_L), Pol_R(s_i))$, $i < j$, denote the access control decision (*true* or *false*) made by the access control module in service s_i by evaluating the attributes of the reader service s_j, $Attr(s_j)$, and the attributes of the local input data resource $s_i.In_L$, $Attr(s_i.In_L)$, against s_i's reader policy. Let $auth(Attr(s_i), Attr(s_j.Out_L), Pol_W(s_j))$, $i < j$, denote the access control decision made by the access control module in service s_j by evaluating the attributes of the writer service s_i, $Attr(s_i)$, and the attributes of the written data resource $s_j.Out_L$, $Attr(s_j.Out_L)$, against s_j's writer policy. Let $valid_R(s_i, s_j)$ denote the validity of s_j reading $s_i.In_L$ from s_i, and $valid_W(s_i, s_j)$ denote the validity of s_i writing $s_j.Out_L$. The validities of various read/write accesses in a length-n+2 service chain are described as follows.

- $valid_R(s_i, s_j) = auth(Attr(s_j), Attr(s_i.In_L), Pol_R(s_i))$, for all i, j, $0 \le i < j \le n+1$, $j = i+1$.
- $valid_W(s_i, s_j) = auth(Attr(s_i), Attr(s_j.Out_L), Pol_W(s_j))$, for all i, j, $0 \le i < j \le n+1$, $j = i+1$.
- $valid_R(s_i, s_j) = true$, for all i, j, $0 \le i < j \le n+1$, $j > i+1$.
- $valid_W(s_i, s_j) = true$, for all i, j, $0 \le i < j \le n+1$, $j > i+1$.
- $valid(<s_0, \ldots, s_{n+1}>) = \wedge valid_R(s_i, s_j) \cdot valid_W(s_i, s_j)$, for all i, j, $0 \le i < j \le n+1$. □

P_C is defined to capture the major characteristic of conventional web service access control models in which only the direct interactions between services are considered. When $j = i+1$, $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$ represents the validity of the read (s_j reads from s_i) and write (s_i writes to s_j) operations between two consecutive services, s_i and s_j. When $j > i+1$, $valid_R(s_i, s_j)$

and $valid_W(s_i, s_j)$ represent the validity of the read and write operations between two nonconsecutive services. In P_C, the validity of a pair of nonconsecutive services (read or write) is assumed to be *true* (conventional access control consideration). The validity of a service chain requires all the pairs of services to be valid.

Note that we do not consider the intermediate or unknown decision in P_C, as in practice, such a decision either has the same effect as positive or negative decision, or triggers negotiation which consumes additional messages but eventually leads to an effective decision (*true* or *false*). Also note that we can replace the attribute-based access control module by any other existing access control modules such as role-based access control, multilevel security system, etc. Choosing attribute-based access control is due to its generosity that it can be used to represent most of the existing access control models.

4.4.1 Capability Assessment

We assess the capability of the IFC model theoretically and compare it with P_C. We prove that the conventional access control model is essentially a special case of the IFC model. We also prove that the IFC model is more powerful than the conventional model by showing that there exist cases that are achievable by the IFC model but not achievable by P_C. To simplify the capability assessment, we define an abstract model, P_S, which is equivalent to the IFC model in Definition 4.5.

Definition 4.5. P_S is an attribute-based information flow control model defined for a length-n+2 service chain $<s_0, s_1, \ldots, s_{n+1}>$. Let $Attr(x)$ denote the set of attributes of a service or a data resource x. Let $Pol_R(s_i)$ denote the reader policy of s_i on disseminating some of its local data

resources to a reader service and $Pol_W(s_j)$ denote the writer policy of s_j on a writer service writing some of s_j's local data. Let $auth(Attr(s_j), Attr(s_i.In_L), max\{tf(s_i), \ldots, tf(s_{j-1})\}, Pol_R(s_i))$, $i < j$, denote the information flow control decision (*true* or *false*) made by the information flow control module in service s_i by evaluating the attributes of the reader service s_j, $Attr(s_j)$, the attributes of the local input data resource $s_i.In_L$, $Attr(s_i.In_L)$, and the maximal transformation factor of the intermediate services, $max\{tf(s_i), \ldots, tf(s_{j-1})\}$, against s_i's reader policy. Let $auth(Attr(s_i), Attr(s_j.Out_L), max\{tf(s_{i+1}), \ldots, tf(s_j)\}, Pol_W(s_j))$, $i < j$, denote the information flow control decision made by the information flow control module in service s_j by evaluating the attributes of the writer service s_i, $Attr(s_i)$, the attributes of the written data resource $s_j.Out_L$, $Attr(s_j.Out_L)$, and the maximal transformation factor of the intermediate services, $max\{tf(s_{i+1}), \ldots, tf(s_j)\}$, against s_j's writer policy. Let $valid_R(s_i, s_j)$ denote the validity of s_j reading $s_i.In_L$ from s_i, and $valid_W(s_i, s_j)$ denote the validity of s_i writing $s_j.Out_L$. The validities of various read/write accesses in a length-n+2 service chain are described as follows.

- $valid_R(s_i, s_j) = auth(Attr(s_j), Attr(s_i.In_L), tf(s_i), Pol_R(s_i))$, $\forall i, j, 0 \le i < j \le n+1, j = i+1$.
- $valid_W(s_i, s_j) = auth(Attr(s_i), Attr(s_j.Out_L), tf(s_j), Pol_W(s_j))$, $\forall i, j, 0 \le i < j \le n+1, j = i+1$.
- $valid_R(s_i, s_j) = auth(Attr(s_j), Attr(s_i.In_L), maxtf_R, Pol_R(s_i)) \vee (maxtf_R = \text{NR})$, $\forall i, j, 0 \le i < j \le n+1, j > i+1$, where $maxtf_R = max\{tf(s_i), \ldots, tf(s_{j-1})\}$.
- $valid_W(s_i, s_j) = auth(Attr(s_i), Attr(s_j.Out_L), maxtf_W, Pol_W(s_j)) \vee (maxtf_W = \text{NR})$, $\forall i, j, 0 \le i < j \le n+1, j > i+1$, where $maxtf_W = max\{tf(s_{i+1}), \ldots, tf(s_j)\}$.
- $valid(<s_0, \ldots, s_{n+1}>) = \wedge valid_R(s_i, s_j) \cdot valid_W(s_i, s_j)$, $\forall i, j, 0 \le i < j \le n+1$. □

Note that, when the maximal transformation factor of the intermediate services between s_i and s_j, $i < j$, is NR, s_i will directly authorize the reader service s_j and s_j will directly authorize the

writer service s_i. That is, their information flow control decisions will be simply *true*. It can be easily seen that P_S is equivalent to the IFC model.

Now we show that the IFC model can do whatever the conventional models can do by showing that the P_C model is essentially a special case of the P_S model. We specialize the P_S model to construct a new model P_S' and prove that it is equivalent to P_C in capability.

Theorem 4.1. P_C is a special case of P_S.

Proof. We construct a specialized model P_S' based on P_S as follows. Consider a length-n+2 service chain. In P_S', we set $tf(s_i)$ = NR, for all i, $0 \leq i \leq n+1$, regardless what the actual transformation effect s_i may have and, hence, in P_S', $max\{tf(s_i), \ldots, tf(s_j)\}$ is NR for all i, j, $1 \leq i < j \leq n+1$. Subsequently, in P_S', $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$ are *true* for all i, j, $1 \leq i < j \leq n+1$, $j > i+1$. Thus, the validation of the service chain $<s_0, \ldots, s_{n+1}>$ only requires the generation of the access control decisions $auth(Attr(s_j), Attr(s_i.In_L), tf(s_i), Pol_R(s_i))$ and $auth(Attr(s_i), Attr(s_j.Out_L), tf(s_j), Pol_W(s_j))$, for all i, j, $0 \leq i < j \leq n+1$, $j = i+1$. Also, in the P_S' model, we construct the access control module for each service such that it ignores the transformation factor in its decision making process. Then, we have $auth(Attr(s_j), Attr(s_i.In_L), tf(s_i), Pol_R(s_i)) = auth(Attr(s_j), Attr(s_i.In_L), Pol_R(s_i))$ and $auth(Attr(s_i), Attr(s_j.Out_L), tf(s_j), Pol_W(s_j)) = auth(Attr(s_i), Attr(s_j.Out_L), Pol_W(s_j))$ and, hence, the P_S' model is equivalent to the P_C model in capability, and the latter is a special case of the P_S model. □

In the following, we show that the IFC model is more powerful that conventional web service access control models by proving that there exists at least one case in which the user's access control requirements cannot be achieved in the P_C model but can be achieved in the P_S model.

Intuitively, the P_S model allows the definition of more flexible policies to let the users (services) control the flow of their data.

Theorem 4.2. In a service chain, there exists at least one case in which some services in the service chain cannot control the flow of its information under the P_C model, which can be achieved under the P_S model.

Proof. Consider length-3 service chains $<s_0/s_0', s_1, s_2/s_2'>$. Note that s_i/s_i' represents candidate concrete services that can instantiate the same abstract service as_i. Note that s_i and s_i' may have different attributes and access control policies. s_0 has certain sensitive data $s_0.In_L$ and s_0 does not wish to release it to s_2. That is to say, $<s_0, s_1, s_2>$ should not be established. Also, s_0' has certain critical data that should not be released to s_2' and, hence, $<s_0', s_1, s_2'>$ should not be established. Assume that the transformation factors of all services are HR.

In the P_S model, we have $auth(Attr(s_1), Attr(s_0.In_L), tf(s_0), Pol_R(s_0)) = true$, $auth(Attr(s_1), Attr(s_0'.In_L), tf(s_0'), Pol_R(s_0')) = true$, $auth(Attr(s_0), Attr(s_1.Out_L), tf(s_1), Pol_W(s_1)) = true$, and $auth(Attr(s_0'), Attr(s_1.Out_L), tf(s_1), Pol_W(s_1)) = true$. Also, we have $auth(Attr(s_2), Attr(s_1.In_L), tf(s_1), Pol_R(s_1)) = true$, $auth(Attr(s_2'), Attr(s_1.In_L), tf(s_1), Pol_R(s_1)) = true$, $auth(Attr(s_1), Attr(s_2.Out_L), tf(s_2), Pol_W(s_2)) = true$, and $auth(Attr(s_1), Attr(s_2'.Out_L), tf(s_2'), Pol_W(s_2')) = true$. Thus, the two service chains $ch_1 = <s_0, s_1, s_2'>$ and $ch_2 = <s_0', s_1, s_2>$ can be established successfully in both the P_C model and the P_S model.

It is also necessary to ensure that $ch_3 = <s_0, s_1, s_2>$ and $ch_4 = <s_0', s_1, s_2'>$ will not be established, as in ch_3, s_2 violates the access control policies of s_0 and, in ch_4, s_2' violates the access control policies of s_0'. This can be achieved in P_S as $auth(Attr(s_2), Attr(s_0.In_L)$, HR,

$Pol_R(s_0)) = false$ and $auth(Attr(s_2'), Attr(s_0'.In_L)$, HR, $Pol_R(s_0')) = false$. However, there is no way in P_C to achieve this goal, as in P_C, for any services s_i and s_j, if $j > i+1$, then the decision will be always *true*. Therefore, it is impossible to obtain a design to disallow the construction of the service chains ch_3 and ch_4 in the P_C model, but possible in the P_S model. □

4.4.2 Complexity Assessment

We analyze the complexity of the IFC model in terms of the total number of information flow control decisions to be made to validate a service chain. As can be seen, the P_C model has the minimal complexity, $O(n)$, which is necessary for the validation of a length-n service chain. In the following, we first show the worst-case complexity of the IFC model. Then, we show that when we only need to achieve the same capability as the conventional access control model (P_C), the IFC model (P_S) can be specialized as in Theorem 4.1 and its complexity becomes the same as that of the P_C model.

Theorem 4.3. The worst-case complexity, i.e. the total number of information flow control decisions required to validate a service chain, of the IFC model is $O(n^2)$, where n is the length of the service chain.

Proof. Consider a length-n service chain $<s_0, \ldots, s_{n-1}>$. $valid(<s_0, \ldots, s_{n-1}>)$ requires the generation of $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$, for all for all i, j, $0 \le i < j \le n-1$. In the worst case, i.e. when the transformation factors of all the computation functions are not NR, the generation of each $valid_R(s_i, s_j)$ or $valid_R(s_i, s_j)$ requires the generation of one access control decision. As the total number of service pairs to be validated is $n\cdot(n-1)$, the total number of required information flow control decisions is $2\cdot n\cdot(n-1)$ and, hence, the complexity of the IFC model is $O(n_2)$. □

As discussed in Theorem 4.1, the P_C model is a special case of the P_S model. When we assign NR to the transformation factors of all the services in the service chain, we can specialize P_S to P_S' and P_S' has the same capability as P_C. In the following, we show that in the case P_S is specialized to have the same power as P_C, then, it has the same complexity as that of P_C.

Theorem 4.4. When $tf(s_i)$ = NR, for all i, $0 \leq i \leq n-1$, the total number of information flow control decisions required to validate a length-n service chain under the IFC model is $O(n)$, where n is the length of the service chain.

Proof. Consider a length-n service chain $<s_0, \ldots, s_{n-1}>$. When $tf(s_i)$ = NR, for all i, j, $0 \leq i < j \leq n-1$, $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$, for all i, j, $0 \leq i < j \leq n-1$, $j > i+1$, can be simply validate to *true* without enforcing access control policies. In this case, to validate this service chain, one only needs to generate $auth(Attr(s_j), Attr(s_i.In_L), tf(s_i), Pol_R(s_i))$ and $auth(Attr(s_i), Attr(s_j.Out_L), tf(s_j), Pol_W(s_j))$, for all i, j, $j = i+1$, that is, $2\cdot(n-1)$ access control decisions. Therefore, the best-case complexity of the IFC model is $O(n)$ which is the same as that of the conventional access control model. □

CHAPTER 5

SECURE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL: A MEDIATOR-BASED APPROACH

As discussed in Section 1.1.3, in composite services, each component service may define detailed access control policies and enforce them at the execution time to ensure the secure use of the web service and the data resources of the service provider. However, existing web service composition mechanisms do not consider these access control policies and, hence, the composed composite services may be very likely to fail at the execution time due to the access control violations. Therefore, it is beneficial to evaluate the access control policies of the candidate component services at the service composition time, in addition to the execution-time access control enforcement, so as to minimize the execution-time failure rate of the composed composite services. Considering access control at the composition time is not straightforward, raising two additional issues regarding the trustworthiness of the service composers and the cost of composition-time policy evaluation (Section 1.1.3).

In this chapter, we discuss the mediator-based approach for secure service composition with information flow control policies. First, we extend the information flow control model discussed in Chapter 4 and introduce the global information flow control rules (Section 5.1). Then, we introduce a three-phase service composition protocol to address the trusted composer issue and performance issue in composition-time access control validation. In the first phase, as the search

space may be very large, we consider a more efficient but less precise method to quickly evaluate and prune the candidate compositions. Specifically, we use the information of the historical service composition transactions to estimate the fitness of the candidate compositions, rank them, and select the top candidates. In the second phase, we consider a local policy evaluation process to achieve more precise evaluation of the candidate compositions. In this process, the service composer uses the information flow policies and/or attribute certificates cached or newly downloaded from the security authorities of the involved services to locally validate the candidate compositions. As service composers may not be fully trusted, the accesses to the policies are considered as special privileges granted by the security authorities rather than assumed. Although it is unlikely that the service composer can validate all candidate compositions, a majority of the service pairs may still be validated, and many invalid candidate compositions can be eliminated. In the third phase, we consider a remote policy evaluation process to further validate the candidate compositions by validating previously unverifiable service pairs. In this process, the security authorities of the involved services evaluate their protected information flow control policies and return their decisions to the service composer to help derive the final composition decisions. Negotiation may be needed in this phase to exchange the credentials and/or policy information between the service composer and the security authorities and between the security authorities in different domains. Since the service pairs validated in the second phase need not be validated again, this time consuming process will only be performed on a few service pairs of very few final candidates. The detailed three phase composition protocol is discussed in Section 5.2.

To study the performance of the proposed mechanism, we develop a simulation system to simulate various protocols and compare their performance, including the three-phase composition protocol, the single-phase composition protocol, and the protocol without composition-time access control validation. The result shows that, without composition-time access control, the composition and execution cost increases dramatically as the success rate decreases (If the information flow control policies of component services are strict, then it is difficult to find a valid composition and the success rate is low). When the success rate is around 50%, even the single-phase composition protocol performs better than the protocol without composition-time access control. The three-phase composition protocol performs much better than the other two mechanisms even when the success rate is high (90%). We also compare the performance of the protocols under various service chain sizes. With the increasing service chain size, the performance gain by the three-phase composition protocol becomes more significant. The experimentation system setup and the performance results are discussed in Section 5.3.

5.1 GLOBAL INFORMATION FLOW CONTROL RULES

Consider each pair of services (s_i, s_j), $0 \leq i < j \leq n+1$, in a service chain. On one hand, s_i may read some critical information from some local data resource, $s_i.In_L$, and use it together with the input data received from its prior service s_{i-1}, $s_i.In_F$, to generate its output data, $s_i.Out_F$, which is directly ($j = i+1$) or indirectly ($i+1 < j \leq n+1$) delivered to the subsequent services s_j, $i < j \leq n+1$. In this case, s_j may be able to derive the sensitive information contained in $s_i.In_L$ from the data it receives, $s_j.In_F$. On the other hand, s_j may use the input data received from its prior service s_{j-1}, $s_j.In_F$, and the information contained in s_j's local data resource, $s_j.In_L$, to produce some

intermediate data which is written into some protected local data resource, $s_j.Out_L$. As $s_j.In_F$ may be computed from $s_i.In_L$ which may contain corrupted information, $s_j.Out_L$ may be contaminated by the write operation. Thus, during composition, the service composer needs to make sure that the information flow from s_i to s_j, for all i, j, $0 \leq i < j \leq n+1$, does not violate any confidentiality (s_i's) and/or integrity (s_j's) requirements. To ensure the secure information flow, it is necessary for each service in the service chain to define policies to control the flow of its sensitive information (Section 4.3). Also, it is important to define rules to govern the composition process such that the service chain generated by the service composer does not violate any information flow policies specified by the individual component services in the service chain. In this section, we define the global information flow control (GIFC) rules.

5.1.1 Basic Global Information Flow Control Rules

The goal of information flow control is to guarantee that the sensitive information of each service in a service chain is not only secure from direct accesses but also secure after it flows (in its raw or processed form) to the subsequent services. To achieve secure information flow, the service composer needs to ensure that, for any pair of services (s_i, s_j), $0 \leq i < j \leq n+1$, in a service chain $<s_0, s_1, \ldots, s_{n+1}>$, s_j is authorized to read the sensitive information contained in $s_i.In_L$, and s_i is authorized to write to the protected data resource $s_j.Out_L$. This principle is specified by the basic GIFC rules given in Figure 5.1.

$BGIFC_1$.	$(auth(Attr(s_j), Attr(s_i.In_L), max\{tf(s_i), \ldots, tf(s_{j-1})\}, Pol_R(s_i)) = true)$ $\Rightarrow (valid_R(s_i, s_j) := true), \forall i, j, 0 \le i < j \le n+1.$
$BGIFC_2$.	$(auth(Attr(s_i), Attr(s_j.Out_L), max\{tf(s_i), \ldots, tf(s_j)\}, Pol_W(s_j)) = true)$ $\Rightarrow (valid_W(s_i, s_j) := true), \forall i, j, 0 \le i < j \le n+1.$
$BGIFC_3$.	$((valid_R(s_i, s_j) = true) \wedge (valid_W(s_i, s_j) = true))$ $\Rightarrow (valid(s_i, s_j) := true), \forall i, j, 0 \le i < j \le n+1.$
$BGIFC_4$.	$valid(<s_0, ..., s_{n+1}>) := \wedge valid(s_i, s_j), \forall i, j, 0 \le i < j \le n+1.$

Figure 5.1. Basic global information flow control rules.

Note that $valid_R(s_i, s_j)$ denotes the validity of the read operation (s_j reads from s_i) in service pair (s_i, s_j) and $valid_W(s_i, s_j)$ denotes the validity of the write operation (s_i writes to s_j) in service pair (s_i, s_j). If both $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$ are true, then the selection of service pair (s_i, s_j) for the service chain is valid, i.e. $valid(s_i, s_j)$ = true. If all the selections are valid, i.e. $valid(s_i, s_j)$ = true, for all $i, j, 0 \le i < j \le n+1$, then the service composition is valid.

5.1.2 Advanced Global Information Flow Control Rules

To validate a candidate composition according to the basic GIFC rules, we need to validate each pair of services in the service chain, (s_i, s_j), for all $i, j, 0 \le i < j \le n+1$, i.e. evaluating the attributes of s_j/s_i, $Attr(s_j)/Attr(s_i)$, and the attributes of the read/written data resource $s_i.In_L/s_j.Out_L$, $Attr(s_i.In_L)/Attr(s_j.Out_L)$, against the reader/writer IFC policy of s_i/s_j, $Pol_R(s_i)/Pol_W(s_j)$. However, there are situations under which the security checks for some service pairs can be fully ignored.

Consider the case where there exists a service s_k, $0 < k \le n$, such that the transformation factor of s_k, $tf(s_k)$, is NR. This implies that the information flow from s_k's input data, $s_k.In_F$, to its output, $s_k.Out_F$, has a very low risk of leaking sensitive information contained in $s_k.In_F$ to $s_k.Out_F$ (it is

very difficult to derive the sensitive information contained in $s_k.In_F$ from $s_k.Out_F$). Also, it is very difficult to contaminate $s_k.Out_F$ using potentially corrupted information contained in $s_k.In_F$. In this case, s_k essentially breaks the service chain into two partial service chains, $<s_0, \ldots, s_{k-1}>$ and $<s_{k+1}, \ldots, s_{n+1}>$. When considering the information flow control, such partial service chains can be considered separately, as the data generated by any two services in different partial service chains can be considered to be unrelated. More specifically, for any pair of services (s_i, s_j), $i < k < j$, s_i can directly authorize service s_j (s_j is not a reader service of $s_i.in_L$) without enforcing s_i's reader IFC policy, i.e. $valid_R(s_i, s_j)$ is always true. Similarly, for any pair of services (s_i, s_j), $i < k < j$, s_j can directly authorize service s_i (s_i is not a writer service of $s_j.Out_L$) without enforcing s_j's writer IFC policy, i.e. $valid_W(s_i, s_j)$ is always true.

Based on these observations, we define advanced GIFC rules using the transformation factor we introduced in Section 4.2. The advanced GIFC rules specify several special situations under which some security checks, for a pair of services (s_i, s_j), for some i, j, can be skipped. The Advanced GIFC Rules are specified in Figure 5.2

$AGIFC_1$.	$(max\{tf(s_i), \ldots, tf(s_{j-1})\} = NR)$ $\Rightarrow (valid_R(s_i, s_j) := true), \forall i, j, 0 \leq i < j \leq n+1.$
$AGIFC_2$.	$(max\{tf(s_i), \ldots, tf(s_j)\} = NR)$ $\Rightarrow (valid_W(s_i, s_j) := true), \forall i, j, 0 \leq i < j \leq n+1.$

Figure 5.2. Advanced global information flow control rules.

5.2 THREE-PHASE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL

In service composition, a service composer takes an abstract service chain $<s_0, as_1, \ldots, as_n, s_{n+1}>$ from the user and selects appropriate concrete services to instantiate $as_1, \ldots, as_n$ while satisfying the information flow control policies. The service composer retrieves a set of concrete services for each abstract service as_i, $1 \leq i \leq n$, from the UDDI registries and generates the set of candidate compositions $CH_0 = \{<s_0^k, \ldots, s_{n+1}^k> \mid \text{for all } k\}$. Note that, during composition, we only need to consider the selection for $as_1, \ldots, as_n$, i.e. $s_1^k, \ldots, s_n^k$, for all k ($s_0^k = s_0$, $s_{n+1}^k = s_{n+1}$, for all k), but when considering the access control issue, all services in the concrete service chain, $<s_0^k, \ldots, s_{n+1}^k>$, should be considered. For each candidate, the service composer follows the GIFC rules and validates whether $valid(<s_0^k, ..., s_{n+1}^k>)$ is true. More specifically, the service composer validates whether for all i, j, $0 \leq i < j \leq n+1$, $valid_R(s_i, s_j)$ and $valid_W(s_i, s_j)$ are true.

In service composition, the service composer may have to explore $O(c^n)$ candidate compositions in the worst case, where n is the number of abstract services in the abstract service chain and c is the average number of candidate concrete services per abstract service. For each candidate service chain $<s_0^k, ..., s_{n+1}^k>$, the service composer may have to check whether $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$ are true, for all i, j, $0 \leq i < j \leq n+1$. This involves $O(n^2)$ information flow control decisions. In some cases, this policy evaluation process involves the retrieval of $Pol_R(s_i^k)$ and $s_j^k.ac$ (containing $Attr(s_j^k)$), and $Pol_W(s_j^k)$ and $s_i^k.ac$ (containing $Attr(s_i^k)$), from $dom(s_i^k).SA$ and $dom(s_j^k).SA$, respectively. Even if the required information flow control policies and attribute certificates are cached at the service composer, the policy evaluation may

still be relatively expensive, as there may be many policy rules in $Pol_R(s_i^k)$ and $Pol_W(s_j^k)$ to be evaluated and results to be combined. In case they are not cached, the service composer needs to interact with $dom(s_i^k).SA$ and $dom(s_j^k).SA$ to compute $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$, respectively. If this expensive validation process is applied to each candidate composition, the composition cost can be prohibitively high.

We consider a three-phase composition protocol to achieve more efficient service composition. In the first phase (Section 5.2.1), there may be many candidate compositions and an efficient method is needed to quickly evaluate and eliminate some candidates. Instead of actually evaluating information flow control policies to compute $valid_R(s_i^k, s_j^k)$ and $valid_W(s_i^k, s_j^k)$, the service composer uses some locally available information of the historical composition transactions to compute the likelihood of $valid_R(s_i^k, s_j^k)$ or $valid_W(s_i^k, s_j^k)$ being true (denoted as $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$, respectively). Then, the fitness value for each candidate service chain $<s_0^k, \ldots, s_{n+1}^k>$, $fit(<s_0^k, \ldots, s_{n+1}^k>)$, are computed, and the top L_1 (L_1 is the percentage that ranges from 0 to 1) candidates are selected and included in CH_1.

In the second phase (Section 5.2.2), a more accurate but potentially more time-consuming process is used to evaluate the candidates in CH_1. Specifically, for each candidate service chain in CH_1, $<s_0^k, \ldots, s_{n+1}^k>$, for all k, $1 \leq k \leq |CH_1|$, the service composer uses the cached or newly downloaded (if the information is not cached or is stale) information flow control policies and/or attribute certificates to evaluate $valid(<s_0^k, \ldots, s_{n+1}^k>)$ by locally evaluating $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$, for all i, j, $0 \leq i < j \leq n+1$. In this process, some invalid candidate service

chains may be eliminated. If a valid candidate is identified, then it is directly returned to the user and the third phase is skipped. Since some of the policies and attributes may be protected and cannot be downloaded, the validity of some of the candidate compositions may not be verifiable by the service composer. In this case, the fitness value for each candidate service chain in CH_1, $fit(<s_0^k, \ldots, s_{n+1}^k>)$, are recomputed, and the top L_2 (L_2 is the percentage that ranges from 0 to 1) candidates are selected and included in CH_2.

In the third phase (Section 5.2.3), any selected candidates in the second phase (the potential solution) should be fully validated. For each service pair (s_i^k, s_j^k) in the candidate service chain $<s_0^k, \ldots, s_{n+1}^k>$, $1 \leq k \leq |CH_2|$, if $Pol_R(s_i^k)$ or $Pol_W(s_j^k)$ is protected, then the service composer needs to forward the attributes of s_j^k or s_i^k, i.e. $Attr(s_j^k)$ or $Attr(s_i^k)$, to $dom(s_i^k).SA$ or $dom(s_j^k).SA$ for remote policy evaluation. In case the policy evaluation requires some protected attributes of s_j^k or s_i^k, the service composer may initiate a negotiation session in which, $dom(s_i^k).SA$ and $dom(s_j^k).SA$ gradually exchange their protected attributes.

In case that the third phase analysis does not yield a valid solution from CH_2, the process will rewind to the second phase to choose the next best L_2 candidate compositions in CH_1 and perform third-phase analysis again. If CH_1 does not include a valid composition, then the process will rewind to the first phase and choose the next best L_1 candidates in CH_0 and perform the whole process again. The overall three-phase composition process is shown in Figure 5.3.

1. Run first phase analysis on CH_0 and sort CH_0.
2. Include top-L_1 candidates in CH_1.
3. Run second phase analysis on CH_1 and sort CH_1.
4. Include top-L_2 candidates in CH_2.
5. Run third phase analysis on CH_2.
6. $CH_1 := CH_1 - CH_2$ and go to 4.
7. $CH_0 := CH_0 - CH_1$ and go to 2.
8. Return *nil*.

Figure 5.3. Three-phase composition process.

5.2.1 First Phase Analysis

5.2.1.1 Likelihood of Valid Composition

We develop a set of *likelihood computation* (LLC) *rules* to compute $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$. First, we consider the case when there is an information flow break between s_i^k and s_j^k.

- LLC$_1$: If $max\{tf(s_i), \ldots, tf(s_{j-1})\}$ = NR, then $LL_R(s_i^k, s_j^k) = 1$.
- LLC$_2$: If $max\{tf(s_i), \ldots, tf(s_j)\}$ = NR, then $LL_W(s_i^k, s_j^k) = 1$.

If rule LLC$_1$ and LLC$_2$ are not applicable, then we need to use historical validation results to estimate $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$. To facilitate the estimation of the likelihoods, the service composer maintains a database *VDB* to store the validation results of all service pairs in the candidate compositions in the historical composition transactions. The transformation factor between the two services also impacts the information flow control decisions. Thus, each record in VDB is a tuple $(x, y, v_R, v_W, tf(x), tf(y), tf_{inter})$. Here, x and y are two services and x is a prior service of y. $tf(x)$ and $tf(y)$ are the transformation factors of services x and y. tf_{inter} is the maximal transformation factor of the intermediate services between x and y, i.e. $tf_{inter} = max\{tf(post(x)), \ldots, tf(pre(y))\}$, where $post(x)$ denotes the service right after x in the original service chain, and

pre(y) denotes the service right before y in the original service chain. Note that tf_{inter} is computed based on the original service chain but the record does not need to keep the original service chain information. v_R is the final result of $valid_R(x, y)$ and v_W is the final result of $valid_W(x, y)$.

We first consider retrieving strongly matched records for (s_i^k, s_j^k) to estimate $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$.

- LLC_3: If there exist a set of records V in VDB, such that, for all $v \in V$, $s_i^k = v.x$, $s_j^k = v.y$, and $max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\} = max\{v.tf(x), v.tf_{inter}\}$, then (s_i^k, s_j^k) and $(v.x, v.y)$, for all $v \in V$, have strong matches, and we have $LL_R(s_i^k, s_j^k) = \sum_{v \in V} \{v.v_R\}$.
- LLC_4: If there exist a set of records V in VDB, such that, for all $v \in V$, $s_i^k = v.x$, $s_j^k = v.y$, and $max\{tf(s_i^k), \ldots, tf(s_j^k)\} = max\{v.tf(x), v.tf_{inter}, v.tf(y)\}$, then (s_i^k, s_j^k) and $(v.x, v.y)$, for all $v \in V$, have strong matches, and we have $LL_W(s_i^k, s_j^k) = \sum_{v \in V} \{v.v_W\}$.

Here, we convert Boolean values into numerical values. That is, we convert *true* into 1 and *false* into 0.

If rules LLC_3 and LLC_4 are not applicable, then we consider retrieving weakly matched records in *VDB*.

- LLC_5: If there exist a set of records V in VDB, such that, for all $v \in V$, $s_i^k = v.x$, $s_j^k = v.y$, and $max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\} \neq max\{v.tf(x), v.tf_{inter}\}$, then (s_i^k, s_j^k) and $(v.x, v.y)$, for all $v \in V$, have weak matches with matching level $ml_R(v) = (1 - \Delta tf_R(v))$, where $\Delta tf_R(v) = |max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\} - max\{v.tf(x), v.tf_{inter}\}|/maxtf$, and we have $LL_R(s_i^k, s_j^k) = (\sum_{v \in V} v.v_R \cdot ml_R(v))/(\sum_{v \in V} ml_R(v))$.

- LLC_6: If there exist a set of records V in VDB, such that, for all $v \in V$, $s_i^k = v.x$, $s_j^k = v.y$, and $max\{tf(s_i^k), \ldots, tf(s_j^k)\} \neq max\{v.tf(x), v.tf_{inter}, v.tf(y)\}$, then (s_i^k, s_j^k) and $(v.x, v.y)$, for all $v \in V$, have weak matches with matching level $ml_W(v) = (1 - \Delta tf_W(v))$, where $\Delta tf_W(v) = |max\{tf(s_i^k), \ldots, tf(s_j^k)\} - max\{v.tf(x), v.tf_{inter}, v.tf(y)\}|/maxtf$, and we have $LL_W(s_i^k, s_j^k) = (\sum_{v \in V} v.v_W \cdot ml_W(v))/(\Sigma_{v \in V} ml_W(v))$.

Here, we convert transformation factor levels into numerical values (HR = 0, MR = 1, LR = 2, NR = 3). Note that $maxtf = 4$.

If rules LLC_1, LLC_3, and LLC_5 are not applicable, then there is no matched record in VDB that can help compute $LL_R(s_i^k, s_j^k)$. In this case, we consider assigning 1 to $LL_R(s_i^k, s_j^k)$. By assigning 1 to $LL_R(s_i^k, s_j^k)$, we increase the likelihood that s_i^k and s_j^k will be selected in the first and second phases and, hence, force them to be validated in the third phase which generates the validation results which may be used in future composition transactions. We also consider that the service composers may exchange their historical information offline or when there is insufficient information to help evaluate the likelihoods and compute the fitness of candidate compositions. Therefore, it is very unlikely that the rules LLC_1, LLC_3, and LLC_5 are all not applicable. Similarly, if rules LLC_2, LLC_4, and LLC_6 are not applicable, then we set $LL_W(s_i^k, s_j^k) = 1$.

- LLC_7: If rules LLC_1, LLC_3, and LLC_5 are not applicable, then $LL_R(s_i^k, s_j^k) = 1$.
- LLC_8: If rules LLC_2, LLC_4, and LLC_6 are not applicable, then $LL_W(s_i^k, s_j^k) = 1$.

5.2.1.2 First Phase Protocol

We define $fit(<s_0^k, \ldots, s_{n+1}^k>) = \prod_{ij} LL(s_i^k, s_j^k)$, for all i, j, $0 \le i < j \le n+1$, where $LL(s_i^k, s_j^k) = LL_R(s_i^k, s_j^k) \cdot LL_W(s_i^k, s_j^k)$,. The service composer will rank all candidate compositions based on their fitness and select the top L_1 candidates and include them in CH_1. The detailed protocol for the first-phase is given in Figure 5.4.

Input: CH_0, L_1, Output: CH_1, LS.
1. For each candidate service chain $<s_0^k, \ldots, s_{n+1}^k>$ in CH_0,
   a) For all i, j, $0 \le i < j \le n+1$,
      i. Compute $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$.
      ii. $LL(s_i^k, s_j^k) := LL_R(s_i^k, s_j^k) \cdot LL_W(s_i^k, s_j^k)$.
   b) $fit(<s_0^k, \ldots, s_{n+1}^k>) := \prod_{ij} LL(s_i^k, s_j^k)$, for all i, j, $0 \le i < j \le n+1$.
2. $LS := \{(LL_R(s_i^k, s_j^k), LL_W(s_i^k, s_j^k)) \mid$ for all i, j, k, $0 \le i < j \le n+1\}$.
3. Sort elements in CH_0 in the descending order of their fitness and copy the top $\lceil L_1 * \mid CH_0 \mid \rceil$ elements in CH_0 into CH_1.
4. Return CH_1 and LS.

Figure 5.4. First-phase protocol.

5.2.2 Second Phase Analysis

In the second phase, the service composer computes $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$, for all i, j, $0 \le i \le n$, $i < j \le n+1$, using its cached attributes, $Attr(s_j^k)$ and $Attr(s_i^k)$, and/or policies, $Pol_R(s_i^k)$ and $Pol_W(s_j^k)$. If the required information is not stored in the cache, then the service composer downloads $Pol_R(s_i^k)$ and $s_i^k.ac$ (containing $Attr(s_i)$) from $dom(s_i^k).SA$ and $Pol_W(s_j^k)$ and $s_j^k.ac$ (containing $Attr(s_j)$) from $dom(s_j^k).SA$. We consider the process that the service composer downloads the information flow control policies and/or attribute certificates as special delegations, and call them the *policy delegation* and *certificate delegation*, respectively.

5.2.2.1 Policy Delegation and Certificate Delegation

To avoid potential frauds, the download of information flow control policies and/or attribute certificates needs to be properly controlled. We consider each composer is associated with a set of attributes. These attributes may include the name, the domain, the trust level of a security authority on the service composer, the permissions granted to the service composer for downloading some policies/certificates from a specific domain, etc. The attributes of a service composer must be asserted by a security authority. The asserted attributes are included in an attribute certificate and signed by the issuer. When requesting for the delegation of certain policies and/or certificates from $d_i.SA$, for some i, the service composer must present its attribute certificates to $d_i.SA$. If the attribute certificate that the service composer presented to $d_i.SA$ is issued from another domain d_j, $j \neq i$, $d_i.SA$ must validate the certificate from $d_j.SA$. Definition 5.1 defines the attribute and attribute certificate of service composer.

Definition 5.1. Each service composer c is associated to a set of attributes $Attr(c) = \{attr_1(c), attr_2(c), \ldots\}$. Each attribute $attr(c)$, $attr(c) \in Attr(c)$, is defined as a tuple ($attr(c).name$, $attr(c).value$) where, $attr(c).name$ is a string that uniquely specifies the name of the attribute, and $attr(c).value$ is the value of the attribute.

Each service composer owns a set of attribute certificates $c.AC = \{c.ac_1, c.ac_2, \ldots\}$. Each attribute certificate $c.ac$, $c.ac \in c.AC$, is issued by a security authority to c certifying that c owns certain attributes $c.ac.Attr(c)$, where $c.ac.Attr(c) \subseteq Attr(c)$. □

We consider each security authority defines a set of policies, called the *delegation policies*, to control the delegation of the policies and/or certificates of $d_i.sa$. We use $d_i.Pol_D = \{d_i.pol_{D1},$

$d_i.pol_{D2}$, …} to represent the set of all delegation policies in domain d_i. When a service composer c requests for downloading certain policies and/or certificates from $d_i.SA$, c must present one of its attribute certificates, $c.ac$, to $d_i.SA$. After receiving the download request of the service composer c, $d_i.SA$ extracts $c.ac.Attr(c)$ from $c.ac$ and evaluates $c.ac.Attr(c)$ against the delegation policies $d_i.Pol_D$. If the decision is true, $d_i.SA$ sends the requested policies and/or certificates to c with a *delegation certificate*. Each delegation certificate includes the attribute certificate that c presented to $d_i.SA$, the hash value of the delegated policies and/or certificates, and a set of delegation constraints that specify the conditions under which the delegation will be revoked or remain valid. Definition 5.2 defines the delegation certificate.

Definition 5.2. A delegation certificate $c.dc$ is a certificate issued by some security authority $d_i.SA$, to the service composer c, to certify that c is granted the privilege of downloading some information flow control policies and/or attribute certificates from $d_i.SA$ and performing policy evaluation using these policies/certificates for $d_i.SA$. $c.dc = (d_i.SA, c.ac, hash(Pol^* \cup AC^*), Con_D)$, where $c.ac$ is an attribute certificate of c, $hash(Pol^* \cup AC^*)$ is the hash value of the downloaded policies, Pol^*, $Pol^* \subseteq d_i.Pol$, and attribute certificates, AC^*, $AC^* \subseteq \cup s.AC$, for all $s \in d_i.S$, and Con_D is a set of delegation constraints under which the policy or certificate delegation can be revoked or remain valid. □

5.2.2.2 Second Phase Protocol

In the second phase, for each candidate service chain in CH_1, $<s_0^k, \ldots, s_{n+1}^k>$, the service composer refines $LL_R(s_i^k, s_j^k)$ and $LL_W(s_i^k, s_j^k)$ computed in the first phase, for all i, j, $0 \leq i < j \leq n+1$, using cached/downloaded information flow control policies, $Pol_R(s_i^k)$ and $Pol_W(s_j^k)$, and/or

the attribute certificates, $s_j^k.ac$ and $s_i^k.ac$, to compute $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$ locally. If $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ is true, then $LL_R(s_i^k, s_j^k)$ is set to 1. If $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$ is true, then $LL_W(s_i^k, s_j^k)$ is set to 1. If any evaluation result is false, then the candidate service chain is removed from CH_1. If it cannot be fully validated, then the likelihood values computed in the first phase will still be used. Based on the updates, the service composer re-ranks the candidate compositions in CH_1, and selects the top L_2 candidate compositions. The second phase protocol is shown in Figure 5.5.

Input: CH_1, LS, L_2, Output: CH_2, LS, ch.
1. $ResultCache := \varnothing$.
2. For each candidate service chain $<s_0^k, \ldots, s_{n+1}^k>$ in CH_1,
   a) For all $i, j, 0 \le i < j \le n+1$,
      i. If there exists $valid_R(s_i^k, s_j^k)$ in $ResultCache$, then $LL_R(s_i^k, s_j^k) := 1$, else
         1. If there does not exist $Pol_R(s_i^k)$ or $Attr(s_j^k)$, then retrieve $Pol_R(s_i^k)$ from $dom(s_i^k).SA$ and $Attr(s_j^k)$ from $dom(s_j^k).SA$, respectively.
         2. $valid_R(s_i^k, s_j^k) := auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$.
         3. If $valid_R(s_i^k, s_j^k) = false$, then remove $<s_0^k, \ldots, s_{n+1}^k>$ from CH_1 and go to 2.
         4. If $valid_R(s_i^k, s_j^k) = true$, then $LL_R(s_i^k, s_j^k) := 1$ in LL_1 and include $valid_R(s_i^k, s_j^k)$ in $ResultCache$.
      ii. If there exists $valid_W(s_i^k, s_j^k)$ in $ResultCache$, then set $LL_W(s_i^k, s_j^k) := 1$, else
         1. If there does not exist $Pol_W(s_j^k)$ or $Attr(s_i^k)$, then retrieve $Pol_W(s_j^k)$ from $dom(s_j^k).SA$ and $Attr(s_i^k)$ from $dom(s_i^k).SA$, respectively.
         2. $valid_W(s_i^k, s_j^k) := auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$.
         3. If $valid_W(s_i^k, s_j^k) = false$, then remove $<s_0^k, \ldots, s_{n+1}^k>$ from CH_1 and go to 2.
         4. If $valid_W(s_i^k, s_j^k) = true$, then $LL_R(s_i^k, s_j^k) := 1$ in LL_1 and include $valid_W(s_i^k, s_j^k)$ in $ResultCache$.
   b) $fit(<s_0^k, \ldots, s_{n+1}^k>) := \prod_{ij} LL(s_i^k, s_j^k)$, for all $i, j, 0 \le i < j \le n+1$.
   c) If $fit(<s_0^k, \ldots, s_{n+1}^k>) = 1$, then $ch := <s_0^k, \ldots, s_{n+1}^k>$ and return ch.
3. Sort elements in CH_1 in the descending order of their fitness and copy the top $\lceil L_2^* \mid CH_1 \mid \rceil$ elements in CH_1 into CH_2.
4. Return CH_2 and LS.

Figure 5.5. Second-phase protocol.

5.2.3 Third Phase Analysis

In the third phase, the service composer contacts the security authorities $dom(s_i^k).SA$ and $dom(s_j^k).SA$, for all i, j, to remotely evaluate all $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$ which have not been validated in the second phase. In this phase, the service composer may send the cached

attributes and/or attribute certificates of s_j^k or s_i^k to $dom(s_i^k).SA$ or $dom(s_j^k).SA$. However, some attributes in $Attr(s_j^k)$ or $Attr(s_i^k)$ may be protected and cannot be revealed to the service composer. In this case, $dom(s_i^k).SA$ or $dom(s_j^k).SA$, for all i, j, $0 \leq i < j \leq n+1$, where the evaluation of $Pol_R(s_i^k)$ or $Pol_W(s_j^k)$ requires some protected attributes of s_j^k or s_i^k, need to negotiate with $dom(s_j^k).SA$ or $dom(s_i^k).SA$ to retrieve the protected attributes.

5.2.3.1 Negotiation Process

Though negotiation has been widely studied to establish trust relationship between unknown parties and to exchange security-sensitive policies/credentials [LEE06, OLS06], we consider a synchronized negotiation process in which, the service composer acts as the negotiation broker between all security authorities. This is to avoid direct negotiation between all $dom(s_i^k).SA$, $0 \leq i \leq n$, and all $dom(s_j^k).SA$, $i < j \leq n+1$, which results in $O(n^2)$ negotiation channels.

A negotiation session includes a startup round, and m negotiation rounds. The service composer decides the maximal number of negotiation rounds M, and may terminate the negotiation session if m exceeds M. Note that, m may be 0, indicating that all attributes required by the remote policy evaluation are already provided by the service composer and, hence, no negotiation is required.

In the startup round, the service composer initiates the negotiation by sending the cached attributes and/or attribute certificates required by the remote policy evaluation within a special message to all $dom(s_i^k).SA$.

In each round of the negotiation, each security authority identifies the missing attributes required for the policy evaluation and the security authorities who own these attributes, and

sends an attribute request to the service composer. The service composer routes the attribute requests to the designated security authorities. On receiving the attribute request from the service composer, each security authority checks its policies and may return the requested attributes in an attribute response. The attribute response is also routed by the service composer.

During the negotiation, the service composer reorganizes the packages received from different security authorities, and consolidates the packages with the same destination into one message. This approach can reduce the number of negotiation channels into $O(n)$. Note that, sensitive attributes should be encrypted during communication, such that even though the message transfer is through the service composer, the service composer does not know the actual attribute values.

By the end of the negotiation session, either all $dom(s_i^k).SA$ retrieves the required attributes for policy evaluation, or the negotiation fails. For the latter case, the service composer will set $valid(<s_0^k, \ldots, s_{n+1}^k>)$ to false. For the former case, $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$ and $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$, for all i, j, will be either true or false and, hence, $valid(<s_0^k, \ldots, s_{n+1}^k>)$ can be fully decided.

5.2.3.2 Third Phase Protocol

In the third phase, the service composer takes the top candidate composition $<s_0^k, \ldots, s_{n+1}^k>$ in CH_2 and start negotiation process for the service pairs in the candidate service chain that have not been fully validated. Then, $valid(<s_0^k, \ldots, s_{n+1}^k>)$ will have a definite result. If the result is true, $<s_0^k, \ldots, s_{n+1}^k>$ is returned to the user. If the result is false, it is removed from CH_2 and the

next highest ranked candidate composition will be selected to go through the same validation process. The third phase protocol is given in Figure 5.6.

Input: CH_2, LS, Output: ch.
1. For each candidate composition $<s_0^k, \ldots, s_{n+1}^k>$ in CH_2,
   a) For all i, j, $0 \le i < j \le n+1$,
      i. If $auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k)) = unknown$, then send $Attr(s_j^k)$ and $Attr(s_i^k.In_L)$ to $dom(s_i^k).SA$ for remote policy evaluation.
      ii. If $auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k)) = unknown$, then send $Attr(s_i^k)$ and $Attr(s_j^k.Out_L)$ to $dom(s_j^k).SA$ for remote policy evaluation.
2. Initiate negotiation session and wait for decisions.
3. For each candidate composition $<s_0^k, \ldots, s_{n+1}^k>$ in CH_2,
   a) For all i, j, $0 \le i < j \le n+1$,
      i. $LL_R(s_i^k, s_j^k) := auth(Attr(s_j^k), Attr(s_i^k.In_L), max\{tf(s_i^k), \ldots, tf(s_{j-1}^k)\}, Pol_R(s_i^k))$.
      ii. $LL_W(s_i^k, s_j^k) := auth(Attr(s_i^k), Attr(s_j^k.Out_L), max\{tf(s_i^k), \ldots, tf(s_j^k)\}, Pol_W(s_j^k))$.
   b) $fit(<s_0^k, \ldots, s_{n+1}^k>) := \prod_{ij} LL_R(s_i^k, s_j^k) \cdot LL_W(s_i^k, s_j^k)$, for all i, j, $0 \le i < j \le n+1$.
   c) If $fit(<s_0^k, \ldots, s_{n+1}^k>) = 1$, then $ch := <s_0^k, \ldots, s_{n+1}^k>$ and return ch.
4. Return ch.

Figure 5.6. Third-phase protocol.

5.3 PERFORMANCE EVALUATION

To validate the effectiveness and evaluate the performance of the three-phase composition mechanism, we design a set of experiments and compare the three-phase approach with the conventional single phase composition approach. To facilitate the performance comparison and validation, we set up a simulation system to simulate the composition of service chains.

5.3.1 Experimental Setup

The simulation system includes 80 domains and 400 concrete services. Each domain has a domain ID (1 to 80) and the longitude and latitude. The longitude and latitude are used to generate the communication latency between the service composer and the security authority of the domain. We use WS-Sim toolset [SHE10a] to generate the longitude and latitude for each domain and use the correlation between communication latency and distance collected from the Internet to generate the simulated latency.

For each concrete service, we first uniformly randomly select its domain (with equal probability of being in any of the 80 domains). Each concrete service instantiates an abstract service and we consider 200 different abstract services (each abstract service is realized by 2 concrete services). Each concrete service has its own concrete service ID, the ID of the corresponding abstract service, its domain ID, the size of the output, and the service latency. We also use WS-Sim toolset to generate the service output size and the service latency.

The generation of transformation factor for each service is based on the categorization of actual web services/applications [SHE10a]. The categorization and generation process is as follows. We first define two main categories of services based on their major functions. Then, we divide each main category into subcategories based on the type of data they process. For each subcategory, we choose several sample web services. Finally, we generate the transformation factor for each of these services.

The policy download flag and policy evaluation time are randomly generated to facilitate the simulation of the policy evaluation process. The set of all security policies in each domain is

associated with a single policy download flag which is uniformly distributed between 0 (non-downloadable) and 1 (downloadable). We use MatLab to analyze the result of a rule engine scalability test [YOU05] to generate the policy evaluation time (i.e. the time required to evaluate an access control policy). The result shows that the policy evaluation time follows a Gaussian distribution where $\mu = 106.61$ (in milliseconds) and $\lambda = 169.33$. This distribution is used to generate the policy evaluation time. To simulate the policy download time, we randomly generate policy sizes. The generation of policy size SP follows the equation: $SP = \sum_{1 \le i \le K} SR_i$, where K is the number of policy rules and SR_i, $1 \le i \le K$, is the size of the i^{th} policy rule. For simplicity, we assume that K is uniformly distributed between 1 and 50 and SR_i is uniformly distributed between 64 and 512 bytes.

Though we simulate the policy sizes and evaluation time, we do not actually perform policy evaluation. Instead, we use a simple rule to generate the validation results to avoid inconsistent validation outcomes in the simulation. The rule is that the access is granted only if the clearance level of the requesting service is greater than or equal to the security class of the requested data object.

One factor that has significant impact on the performance is the success rate of the composition tasks. When the success rate is high, the conventional single-phase composition mechanism can easily find out a valid composition, while the three-phase composition process needs to spend extra time on the first and second phases and, hence, the advantage it has cannot pay off the extra overhead. On the other hand, when the success rate is low, the single-phase approach may spend much longer time on finding out a valid composition and, hence, the three-phase method may perform much better. We intend to study the impact of the success rate, but it

is difficult to directly control the success rate. Instead, we generate different percentages of public services to indirectly control the success rate.

For simulation purpose, we consider an attribute-based access control system in which each data resource has a security class and each service has a clearance level defined by each domain. The security class *sc*(*r*) measures the sensitivity level of data object *r* and also the level of security protection required by *r*. The security class is defined by multiple levels, including P (Public), C (Confidential), S (Secret), and TS (TopSecret), where P < C < S < TS. We consider each service *s* to be cleared at a certain clearance level in each domain *d*, denoted by *cl*(*s*, *d*). We consider a simple set of security policies: (1) the clearance level of a reader service must be greater than or equal to the security class of the data objects that are read by the service; (2) the clearance level of a writer service must be equal to the security class of the data objects that are written by the service. In the simulation, we ignore data resources that do not need protection and generate the security classes for sensitive data resources following a uniform distribution between C and TS. We use public services ratio (PSR, $0 \leq PSR \leq 1$) as a parameter, and generate the clearance levels of non-public services following a uniform distribution between C and TS. Note that the simple model here is to ease the simulation system generation process as it will be very difficult to simulate real policies and attribute-based certificates.

The client generates a length-*n* abstract service chain by randomly selecting *n* different abstract services (uniform distribution) and submits it to the service composer. The composer uses a conventional single-phase method and the three-phase composition process to select concrete services for the abstract service chain.

We also design a caching system to store the downloaded policies and the validation results of historical composition transactions. We consider a first-in first-out cache replacement policy.

5.3.2 Experimental Results

We conduct three experiments to study the performance of the three-phase composition protocol (P1), the conventional single-phase composition protocol (P2), and the protocol without composition-time access control validation (P3), under different settings including variant success rate, variant service chain length, and variant L_1 and L_2. The results are given as follows.

In Figure 5.7, we compare the time of finding and successfully executing a length-10 service chain required by P1, P2, and P3 under different success rates. As can be seen, when the success rate decrements, the time required by P3 increases dramatically as the failure in execution requires re-composition. When the success rate is below 68%, the cost of P3 is even higher than the single-phase protocol (P2). It can also be seen that, the performance gain of the three-phase composition protocol P1 increases as the success rate decrements. When the success rate is high (97%), P1 is only 1s faster than P2 and performs slightly worse than P3. When the success rate is low (53%), P3 is 8s faster than P2 and 14s faster than P3.

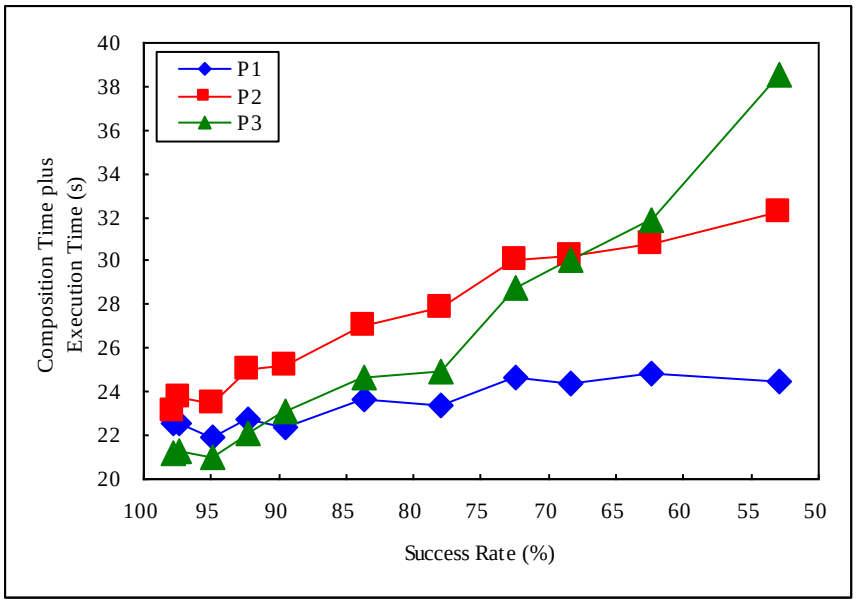

Figure 5.7. Composition and execution time vs. success rate.

In Figure 5.8, we compare the composition time of the P1 and P2 under different service chain lengths. For shorter service chains (length ≤ 10), both methods can find out a valid composition quickly. In these cases, the three-phase protocol (P1) may perform a little worse due to the extra time spent for fitness calculation. For longer service chains (length > 10), P1 performs much better than the single-phase mechanism (P2), from 30% improvement at length 11 to 70% improvement at length 14. From the growing trend, the performance improvement in the three-phase scheme can be much more significant with increasing composition problem size.

Figures 5.9 shows how the selection of top candidate ratios in first and second phases, L_1 and L_2, can impact the performance gain of three-phase composition protocol. The result shows that the selection of these parameters does not have significant impact the performance. With different L_1, the composition time of the three-phase protocol oscillates between 2.3s and 3s. With different L_2, the composition time oscillates between 2.6s and 2.8s.

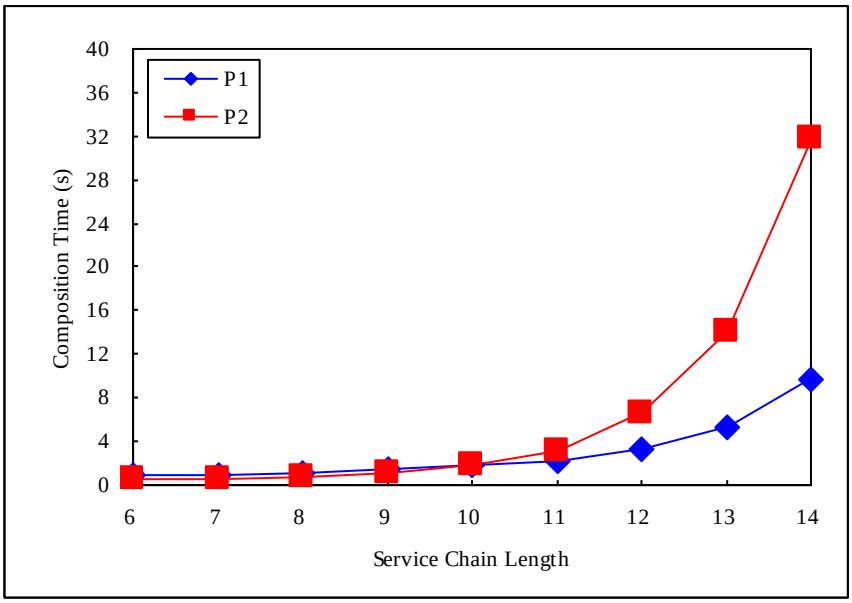

Figure 5.8. Composition time vs. service chain length.

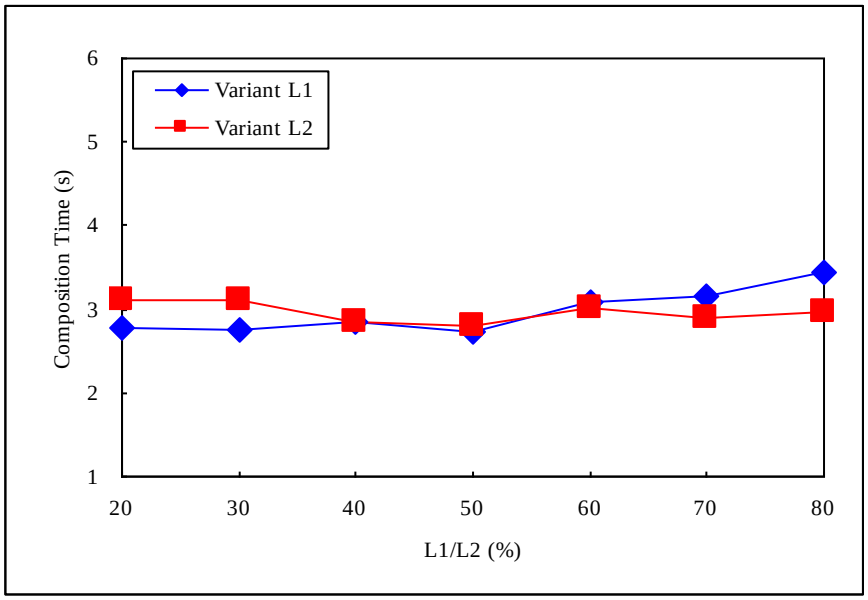

Figure 5.9. Composition time vs. variant L_1 and L_2.

CHAPTER 6

SECURE SERVICE COMPOSITION WITH INFORMATION FLOW CONTROL: A MEDIATOR-FREE APPROACH

In Chapter 5, we have discussed the secure service composition mechanism with a service composer (mediator). In this chapter, we consider a variant of this mechanism, a fully decentralized secure service composition mechanism without a mediator.

We first consider the policy evaluation cost issue in the composition-time access control. As already discussed in Section 5.2, given a length-n abstract service chain, the cost to find a valid concrete service chain is exponential to n ($O(c^n)$, where c is the average number of candidate concrete services per abstract service). To achieve efficient service selection and composition, we also consider using the information extracted from historical composition tasks to help evaluate and rank candidate compositions, so as to validate most promising candidate first. Note that, in the mediator-based approach (Chapter 5), we evaluate each candidate service chain, but in the mediator-free decentralized composition process, we consider that each composed concrete service evaluates the fitness for each candidate service that can instantiate next abstract service in the service chain.

In the distributed composition model, each service makes its own decision on which concrete service to be selected to instantiate the next abstract service in the service chain. Each service s_i (that instantiates as_i), $0 \leq i < n$, cannot predetermine which concrete services its subsequent

services s_j, $i < j \leq n$, may compose, i.e. s_{j+1}. Thus, s_i cannot ensure the secure information flow between s_i and s_{j+1} when making its composition decision (i.e. selecting s_{i+1} to instantiate as_{i+1}). In order to secure the information flow between s_i and s_{j+1}, when s_j composes s_{j+1}, s_j needs to deliver s_{j+1}'s information back to s_i such that s_i can decide whether s_{j+1} is valid according to s_i's information flow control policies. Such a process is called the back-check process (Section 6.1).

We consider additional mechanisms to improve the efficiency of the distributed validation process. First, in order to minimize the communication overhead raised by back-check, we consider that each service may carry along and enforce other services' information flow control policies. That is, s_i may propagate its information flow control policies to other services s_j, $j > i$, through the intermediate service chain between them, such that s_j can enforce s_i's policies when further composing s_{j+1} without interacting with s_i. The basic mechanisms for the carry-along policy communication and evaluation are discussed in Sections 6.2.

Second, we consider a caching scheme to further reduce the communication cost raised by transmitting carry-along policies. Each service may cache the carry-along policies it has received and reuse them. On propagating carry-along policies, each service s_i will enquire s_{i+1} about which policy rules are cached, and only send out the policy rules that are not cached by s_{i+1}. To realize the goal, we design a tree-based representation for carry-along policies to facilitate a space and time efficient interchange of the availability information and the versions of the cached policies. This efficient carry-along policy propagation protocol is discussed in Section 6.3.

Third, we design an efficient back-check process by recording and sending back partial policy evaluation results. The carry-along policies of s_i may not be fully evaluated at s_j, $j > i$, due to

insufficient information. In this case, s_j will send s_{j+1}'s credential back to s_i for back-check. If designed naively, s_i may reevaluate all the policy rules while some of these rules may have already been evaluated based on the carry-along policies. We also use the policy tree to record the state of the carry-along policy evaluation at s_j and send the condensed validation information back to s_i to avoid redundant policy evaluation. This redundancy-free back-check process is discussed in Section 6.4.

Based on all the processes discussed above, we develop a complete mediator-free service composition protocol and present it in Section 6.5.

6.1 BACK-CHECK PROCESS

In distributed service composition, each concrete service s_i, $0 \leq i < n$, dynamically selects the next concrete service in the service chain based on the desired functionality (i.e. an abstract service). Hence, each service s_i may not know which services s_j, $i < j \leq n$, will compose, i.e. s_{j+1}, and cannot ensure the secure information flow between s_i and s_{j+1} when making its composition decision. In order to enable s_i to validate s_{j+1}, the simplest way is to let s_j forward s_{j+1}'s attribute certificate back to s_i such that s_i can evaluate s_{j+1}'s attributes against its information flow control policies. Service s_j will not compose s_{j+1} until all its prior services s_i, $0 \leq i < j$, agree on the selection of s_{j+1}. This process is called the back-check process.

6.2 CARRY-ALONG POLICY

The back-check process may significantly degrade the system performance. That is, each service s_j, $0 < j \leq n$, upon composing s_{j+1}, must back-check with all its prior services, i.e. s_i, $0 \leq i < n$. To reduce the overhead, we allow each service s_k, $0 < k < n$, to carry along the information flow

control policies regarding disseminating the sensitive information contained in $s_i.In_L$, $0 \leq i < k$, and the information needed for the policy evaluation (e.g. the attributes of s_i, $Attr(s_i)$, and/or the attributes of s_i's local input data resource $s_i.In_L$, $Attr(s_i.In_L)$) within s_k's request and to forward them to s_j, $k < j \leq n$, such that s_j can help enforce s_i's information flow control policies when further composing s_{j+1}. Such an information flow control policy is called the *carry-along policy*. If the policies of s_i that are carried along the service chain to s_j result in an effective decision (true or false), then s_j does not need to back-check with s_i and, hence, the back-check process between s_i and s_j can be skipped.

On the other hand, some services may have very complicated information flow control policies. Transmitting these policies in a service chain may incur a high communication cost. Also, some policies may require protection and should not be sent out. Thus, the carry-along policy should be light-weight and not security-sensitive. The protocol should still support back-check when the information flow control policies cannot be enforced outside the policy owner's domain or when the carry-along policies do not yield an effective decision. In contrast to the carry-along policy, the policy stored and enforced at the policy owner's site due to security or size concern is called the *residential policy*. For convenience, we use $Pol_R(s)$ and $Pol_C(s)$ to represent the set of all residential policies and the set of all carry-along policies defined for service s and all the data resources that may be accessed during the computation of s, respectively. Note that, the carry-along policies must be consistent with the residential policies; that is, $Pol_C(s)$ and $Pol_R(s)$ must yield exactly the same information flow control decisions for any service s.

6.3 CARRY-ALONG POLICY PROPAGATION

In most cases, the carry-along policies sent out to remote sites may be relatively simple. But when considering a long service chain, the size of the carry-along policies may accumulate and result in a significant communication overhead. We develop a caching protocol to reduce the overhead for propagating the carry-along policies. Each service caches the carry-along policies it receives for later use. Consider the interaction between s_i and s_{i+1} in a service chain. When s_i tries to send its carry-along policies and/or the carry-along policies of its prior services to s_{i+1}, s_i first enquires s_{i+1} about whether s_{i+1} has cached some of the policy rules. Accordingly, s_i will only send the missing carry-along policy rules to s_{i+1}. We design a tree-based representation for carry-along policies to facilitate efficient caching and matching of the policy rules.

Generally, a policy can be viewed as a set of policy rules combined with logical operations. Thus, we can express each policy as a tree in which policy rules are included in leaf nodes, while non-leaf nodes contain logical operators. We formally define the policy tree model in Definition 6.1.

Definition 6.1. A policy tree is used to represent a carry-along policy, in which each non-leaf node is labeled by a logical operator, *AND*, *OR*, or *NOT*, and each leaf node is a policy rule. A policy rule is a triple (*RID*, *VN*, *RS*) where, *RID* is the rule identifier, which uniquely identifies a policy rule in a security domain, *VN* is the version number, and *RS* is the string representation of the body of the policy rule, namely the *rule string*. □

A non-leaf node with an *AND* label indicates that all of its sub trees must be satisfied. An *OR* label indicates that at least one of the sub trees should be satisfied. An *NOT* label reverses the

decision. Figure 6.1 is an example policy tree in which three policy rules are combined with an *OR* operator. In this case, satisfying any of these policy rules will result in a positive decision.

```
<CompletePolicyTree>
  <Operator value = “or”>
    <Rule id = “01” version = “03” decision = “permit”>
      <Operator value = “greaterEqualTo”>
        <AttributeName value = “clearanceLevel”/>
        <AttributeName value = “securityClass”/>
      </Operator>
    </Rule>
    <Rule id = “02” version = “01” decision = “permit”>
      <Operator value = “equalTo”>
        <AttributeName value = “transformationFactor”/>
        <AttributeValue value = “NR” type = “string”/>
      </Operator>
    </Rule>
    <Rule id = “03” version = “05” decision = “permit”>
      <Operator value = “and”>
        <Operator value = “equalTo”>
          <AttributeName value = “transformationFactor”/>
          <AttributeValue value = “LR” type = “string”/>
        </Operator>
        <Operator value = “equalTo”>
          <AttributeName value = “securityClass”/>
          <AttributeValue value = “C” type = “string”/>
        </Operator>
      </Operator>
    </Rule>
  </Operator>
</CompletePolicyTree>
```

Figure 6.1. Example complete policy tree.

In some systems, negotiation may be performed to obtain sufficient information for decision making. Here, we assume that the information flow control decision is purely based on the given policies. Many policy specification languages (e.g. XACML [OAS05]) support the concept of

policy set. A policy set is essentially a set of policies combined together with the logical operators *AND*, *OR*, or *NOT*. Thus, the policy set can also be expressed as a policy tree.

Consider the example policy tree in Figure 6.1. As all its leaf nodes contain the complete policy rules, we call it the *complete policy tree* (CPT). In the cases where the user does not care about the details of one specific policy rule, we can eliminate its rule string and use its rule identifier and version number to represent the entire rule. When all the policy rules in the tree are eliminated (only represented by its rule identifier and version number), we call it the *no-rule policy tree* (NPT, Figure 6.2). If only a part of the policy rules in the tree contain the actual rule strings, then it is a *partial policy tree* (PPT, Figure 6.3).

```
<NoRulePolicyTree>
   <Operator value = “or”>
      <Rule id = “01” version = “03” decision = “permit”/>
      <Rule id = “02” version = “01” decision = “permit”/>
      <Rule id = “03” version = “05” decision = “permit”/>
   </Operator>
</NoRulePolicyTree>
```

Figure 6.2. Example no-rule policy tree.

```
<PartialPolicyTree>
   <Operator value = “or”>
      <Rule id = “01” version = “03” decision = “permit”>
         <Operator value = “greaterEqualTo”>
            <AttributeName value = “clearanceLevel”/>
            <AttributeName value = “securityClass”/>
         </Operator>
      </Rule>
      <Rule id = “02” version = “01” decision = “permit”/>
      <Rule id = “03” version = “05” decision = “permit”/>
   </Operator>
</PartialPolicyTree>
```

Figure 6.3. Example partial policy tree.

The carry-along policy propagation mechanism works in the following way. The first step is the enquiry of the missing policy rules. Each service s_i first converts all the received carry-along policies (in PPT forms) into their NPT forms and sends them to s_{i+1}. s_{i+1} looks up its cache for the matched policy rules (both rule identifier and version number must match), and informs s_i about missing policy rules. The second step is to convert the policy tree from NPT to PPT. On receiving s_{i+1}'s response, s_i will copy the required rule strings into the corresponding leaf nodes and convert all the carry-along policies into their PPT forms, which are sent to s_{i+1}. The last step is that s_{i+1} merge the cached policy rules into the PPT received from s_i. This produces a policy tree in its CPT form. Only the carry-along policies in their CPT forms can be enforced.

6.4 REDUNDANCY-FREE BACK-CHECK

In case that s_i's ($0 \leq i < n$) carry-along policy does not yield an effective decision at s_j, s_j will deliver s_{j+1}'s attribute certificate to s_i for back-check. Note that, although an effective decision cannot be made, some of the policy rules in s_i's carry-along policy can still be evaluated to true

or false. To avoid duplicate efforts in policy evaluation, we make use of the policy tree discussed above in the back-check process to send back partial policy evaluation results. We call such a policy tree *in-evaluation policy tree* (IPT). In an IPT, if a policy rule has been fully evaluated, we replace it with its decision (true or false). If a policy rule cannot be fully evaluated remotely, we simply include its rule identifier and version number. If all the children of a node *x* in the IPT have effective decisions, then the logical operator of *x* will be performed on the decisions of the children of *x* and the result will become the label of *x*. Also, the children of *x* are deleted from the policy tree. This procedure continues recursively until the policy tree cannot be further simplified. Figure 6.4 is an example IPT developed from the CPT in Figure 6.1. In this IPT, rule "01" cannot be evaluated fully and, thus, no effective decision can be derived. Figure 6.5 provides a more complex IPT in which two policy rules need to be evaluated by back-check.

```
<InEvaluationPolicyTree>
   <Operator value = "or">
      <Rule id = "01" version = "03" decision = "permit"/>
      <AttributeValue value = "false" type = "boolean"/>
      <AttributeValue value = "false" type = "boolean"/>
   </Operator>
</InEvaluationPolicyTree>
```

Figure 6.4. Example in-evaluation policy tree.

```
<InEvaluationPolicyTree>
   <Operator value = “or”>
      <Operator value = “or”>
         <AttributeValue value = “false” type = “boolean”/>
         <Rule id = “03” version = “12” decision = “permit”/>
         <AttributeValue value = “false” type = “boolean”/>
      </Operator>
      <AttributeValue value = “true” type = “boolean”/>
      <Operator value = “or”>
         <AttributeValue value = “false” type = “boolean”/>
         <Operator value = “and”>
            <AttributeValue value = “true” type = “boolean”/>
            <Rule id = “04” version = “02” decision = “permit”/>
         </Operator>
      </Operator>
   </Operator>
</InEvaluationPolicyTree>
```

Figure 6.5. Example in-evaluation policy tree.

The redundancy-free back-check mechanism works in the following way. Consider the situation that s_j back-checks with s_i, for some $i < j$. With the carry-along policy propagation mechanism, s_j will have s_i's carry-along policy in its CPT form. The first step is to convert s_i's carry-along policy into its IPT form. For each policy rule which can be fully evaluated, we label the corresponding leaf node with its decision. For each policy rule which cannot derive an effective decision, we keep its rule identifier and version number in the node. Then we recursively simplify the policy tree as discussed above. Service s_j then sends a back-check request with the IPT to s_i. Service s_i loads in the corresponding policy rules for the rule identifiers labeled in the leaf nodes of the IPT and evaluates them. The IPT can be fully evaluated to a Boolean value, which is the final decision. The decision is then sent with the back-check response to s_j.

6.5 MEDIATOR-FREE COMPOSITION PROTOCOL

In the mediator-free service composition process, each composed concrete service, rather than a service composer, selects the concrete service to instantiate the next abstract service in the service chain. The end user s_0 first generates an abstract service chain $<s_0, as_1, \ldots, as_n, s_{n+1}>$ based on the desired functionality. During the composition, each most recently composed concrete service, s_j, $0 \leq j < n$, retrieves a set of concrete services that can instantiate the next abstract service as_{j+1}, $\{s_{j+1}^k \mid \text{for all } k, 1 \leq k \leq |as_{j+1}|\}$ ($|as_{j+1}|$ denotes the number of candidate concrete services that can instantiate as_{j+1}), from the UDDI registry. Note that, we use as_i to represent the i^{th} abstract service in the abstract service chain, s_i^k, $1 \leq k \leq |as_{j+1}|$, to represent the k^{th} candidate concrete service that can instantiate as_i, and s_i to represent the current selection for the concrete service to instantiate as_i. On considering a candidate concrete service s_{j+1}^k, for some k, s_j generates a composition request, which includes the desired abstract service chain, $<s_0, as_1, \ldots, as_n, s_{n+1}>$, the composed partial concrete service chain, $<s_0, s_1, \ldots, s_j>$, the next abstract service to be instantiated, as_{j+1}, and the attribute certificates of $s_0, \ldots, s_j$, $s_0.ac, \ldots, s_j.ac$, and sends it to s_{j+1}^k. On receiving s_j's composition request, s_{j+1}^k evaluates the attributes of $s_0, \ldots, s_j$, $Attr(s_0), \ldots, Attr(s_j)$, against s_{j+1}^k's writer IFC policy. If $s_0, \ldots, s_j$ are all valid according s_{j+1}^k's write IFC policy, s_{j+1}^k sends its attribute certificate, $s_{j+1}^k.ac$, to s_j. On receiving s_{j+1}^k's attribute certificate, s_j evaluates s_{j+1}^k's attributes, $Attr(s_{j+1}^k)$, against s_j's reader IFC policy, and forwards $s_{j+1}^k.ac$ back to all its prior services, $s_0, \ldots, s_{j-1}$, for back-check. If all the information flow control decisions are positive, s_j sends a confirmation message to s_{j+1}^k (s_{j+1} is set to s_{j+1}^k), and s_{j+1} continues to instantiate the next abstract service as_{j+2}. Such a composition process continues until it hits the end user s_{n+1}.

To achieve efficient service selection and composition, we consider that the selection process of each service includes two phases. In the first phase (Section 6.5.1), each service s_j, $0 \leq j \leq n$, uses the information gathered from historical composition transactions to help evaluate all the candidate services for as_{j+1}, s_{j+1}^k, for all k, $1 \leq k \leq |as_{j+1}|$. Specifically, s_j computes the fitness for each candidate service s_{j+1}^k, $fit(s_{j+1}^k)$, using the historical validation results (i.e. $LL_R(x, y)$ and $LL_W(x, y)$), and selects the top candidate service which is to be validated in the second phase.

In the second phase (Section 6.5.2), the top candidate selected by the first phase, s_{j+1}^k, for some k, $1 \leq k \leq |as_{j+1}|$, is validated. On one hand, s_{j+1}^k validates each of its prior concrete services s_i, $0 \leq i \leq j$, by evaluating the attributes of s_i, $Attr(s_i)$, against s_{j+1}^k's writer IFC policy. On the other hand, s_j validates s_{j+1}^k by evaluating the attributes of s_{j+1}^k, $Attr(s_{j+1}^k)$, against s_j's reader IFC policy and helps its prior services, s_0, …, s_{j-1}, validate s_{j+1}^k by either enforcing s_0's, …, s_{j-1}'s carry-along policies or back-checking.

In case that s_j cannot find a valid concrete service for instantiating as_{j+1} to form a valid concrete service chain, s_j initiates an adjustment phase (Section 6.5.3). First, s_j identifies a set of concrete services s_i, for some i, $1 \leq i \leq j$, that may be replaced to generate a valid concrete service chain. s_j sends an adjustment request to the prior service of each such s_i, i.e. s_{i-1}, and s_{i-1} instantiates as_i to another candidate service s_i^k, where $s_i^k \neq s_i$. In case that the adjustment made by some s_{i-1} results in a valid service chain, the adjustment result will be propagated to all the services, i.e. s_0, …, s_j, and s_j will take over the composition task. If some s_{i-1} considers that the likelihood of generating a valid concrete service chain is low, s_{i-1} can directly generate a failure result and propagate it to all the services, and the composition process stops.

6.5.1 First Phase Analysis

When considering a concrete service to instantiate the next abstract service in the service chain as_{j+1}, each service s_j needs to consider, for each candidate service, s_{j+1}^k, for some k, how likely the candidate service chain $<s_0, s_1, \ldots, s_j, s_{j+1}^k>$ would be valid, i.e. the fitness of service chain $<s_0, s_1, \ldots, s_j, s_{j+1}^k>$. Recall that, $fit(<s_0, s_1, \ldots, s_j, s_{j+1}^k>) = \prod_{0 \le p < q \le j} LL(s_p, s_q) \cdot \prod_{0 \le p \le j} LL(s_p, s_{j+1}^k)$. Note that $LL(x, y) = LL_R(x, y) \cdot LL_W(x, y)$. As $\prod_{0 \le p < q \le j} LL(s_p, s_q)$ is equal for all candidate services s_{j+1}^k, for all k, each service s_j only needs to consider $\prod_{0 \le p \le j} LL(s_p, s_{j+1}^k)$. We define the fitness of a concrete service s_{j+1}^k, given a set of composed concrete services S, e.g. $S = \{s_0, s_1, \ldots, s_j\}$, denoted by $fit(s_{j+1}^k|S)$, as follows.

Definition 6.2. Given a set of composed concrete services S, the fitness of a candidate service for instantiating an abstract service as_j, s_j^k, $fit(s_j^k|S)$, measures the likelihood that the selection of s_j^k will constitute a valid service chain containing all the concrete services in S. $fit(s_j^k|S) = \prod_{s \in S} LL(s, s_j^k)$, where $LL(x, y) = LL_R(x, y) \cdot LL_W(x, y)$, for all x, y. □

We use the same mechanism as in Section 5.2.1 (i.e. using the information gathered from historical composition transactions) to compute $LL_R(x, y)$ and $LL_W(x, y)$. We consider that each security domain maintains a database to store the validation results of historical composition transactions that involved the services in the domain. Moreover, some domains may exchange their historical validation results, either offline or when they have insufficient information to evaluate their candidate services.

In the first phase, each service will rank all candidate services for next abstract service in the service chain based on their fitness and select the top candidate for the second phase analysis. The detailed protocol for the first-phase analysis is given in Figure 6.6.

Input: $C_{j+1} = \{s_{j+1}^1, s_{j+1}^2, \ldots\}$, Output: s_{j+1}.
1. For each candidate service in C_{j+1}, s_{j+1}^k, $1 \le k \le |C_{j+1}|$,
   a) For all i, $0 \le i \le j$,
      i. Compute $LL_R(s_i, s_{j+1}^k)$ and $LL_W(s_i, s_{j+1}^k)$.
      ii. $LL(s_i, s_{j+1}^k) := LL_R(s_i, s_{j+1}^k) \cdot LL_W(s_i, s_{j+1}^k)$.
   b) $fit(s_{j+1}^k | \{s_0, \ldots, s_j\}) := \prod_{0 \le p \le j-1} LL(s_p, s_j^k)$.
2. Sort the elements in C_{j+1} in the descending order of their fitness.
3. $s_{j+1} := s_{j+1}^*$, where s_{j+1}^* is the element in C_{j+1} with greatest fitness.

Figure 6.6. First-phase protocol.

6.5.2 Second Phase Analysis

In the second phase, s_j validates the top candidate generated by the first phase, say s_{j+1}^k, for some k. s_{j+1}^k evaluates all the prior services, $s_0, \ldots, s_j$, against its writer IFC policy, $Pol_W(s_{j+1}^k)$. Also, s_j evaluates s_{j+1}^k against its reader IFC policy, $Pol_R(s_j)$, and helps its prior services, $s_0, \ldots, s_{j-1}$, evaluate s_{j+1}^k, by enforcing s_0's, …, s_{j-1}'s carry-along policies or by back-checking.

Each service s_j, $0 \le j \le n$, upon composing s_{j+1}^k, needs to verify whether s_{j+1}^k is valid according to the reader IFC policies of services s_i, $0 \le i \le j$, where s_i is such a service that its local input data $s_i.In_L$ is derivable from the input data of s_{j+1}^k, $s_j.Out_F$. For each of such s_i, s_j need also ensure that s_i is valid according to s_{j+1}^k's writer IFC policies. To determine all such s_i, s_j needs to find the most adjacent prior service of s_{j+1}^k, say s_h, whose transformation factor is NR (h is called the *nearest break point* of s_{j+1}^k, $nbp(s_{j+1}^k)$). For all the services prior to s_h, i.e. s_i, $0 \le i < h$, the mutual

validation between s_i and s_{j+1}^k can be fully skipped (also, s_i's carry-along policies need not be further propagated to the subsequent services of s_j). The second phase protocol is presented in Figure 6.7.

1. On composing s_{j+1}^k, s_j computes the nearest break point $h := nbp(s_{j+1}^k)$, sets $OwnerSet(s_j.Out_F) := \{s_h, \ldots, s_j\}$, and sends the composition request, its attribute certificate, $s_j.ac$, and attribute certificates of its prior services, $s_h.ac, \ldots, s_{j-1}.ac$ to s_{j+1}^k.
2. s_{j+1}^k computes the nearest break point $h = nbp(s_{j+1}^k)$, sets $OwnerSet(s_{j+1}^k.Out_L) := \{s_h, \ldots, s_j\}$, and for each service s_l in $OwnerSet(s_{j+1}^k.Out_L)$,
   a) s_{j+1}^k evaluates $Attr(s_l)$, $Attr(s_{j+1}^k.Out_L)$, and $max\{tf(s_{l+1}), \ldots, tf(s_{j+1}^k)\}$ against $Pol_W(s_{j+1}^k)$.
3. s_{j+1}^k sends the decision of step 2 to s_j.
4. If s_j receives *false* from s_{j+1}^k, s_j selects another candidate service and go to 1; otherwise, continue.
5. s_j evaluates $Attr(s_{j+1}^k)$, $Attr(s_j.In_L)$, and $tf(s_j.F)$ against $Pol_R(s_j)$.
6. For all services s_l in $OwnerSet(s_j.Out_F) - \{s_i\}$,
   a) If $CPT(s_l)$ exists, s_j evaluates $Attr(s_{j+1}^k)$, $Attr(s_l.In_L)$, and $max\{tf(s_l), \ldots, tf(s_j)\}$ against $CPT(s_l)$.
   b) If $CPT(s_l)$ does not exist or does not yield an effective decision, s_j converts $CPT(s_l)$ to $IPT(s_l)$, determines the *rid*s of the policy rules which cannot derive an effective decision, and sends $s_{j+1}^k.ac$ and these *rid*s to s_l for back-check, and waits for s_l's decision.
7. If the decision of step 5 or 6 is *false*, s_j selects another candidate service and go to 1; otherwise, continue.
8. For all s_l in $OwnerSet(s_j.Out_F)$, s_i generates $NPT(s_l)$ and sends them to s_{j+1}^k.
9. s_{j+1}^k looks up its cache for the matched policy rules and replies s_j with the missing policy rules.
10. For all s_l in $OwnerSet(s_j.Out_F)$, s_j generates $PPT(s_l)$ and sends them to s_{j+1}^k.
11. For all s_l in $OwnerSet(s_j.Out_F)$, s_{j+1}^k converts $PPT(s_l)$ to their CPT forms $CPT(s_l)$ by inserting the cached carry-along policies.

Figure 6.7. Second-phase protocol.

6.5.3 Adjustment Phase

In case that an abstract service, say as_{j+1}, cannot be instantiated with any of its candidate concrete services to form a valid service chain, the composition process enters the adjustment phase in which, some composed services s_i, for some i, $1 \leq i \leq j$, may be replaced by other candidates of as_i, i.e. s_i^k, for some k, $s_i^k \neq s_i$.

The first step is that s_j determines all the concrete services, s_i, $1 \leq i \leq j$, that may be replaced to form a valid service chain. For each s_i, if $valid(s_i, s_{j+1}^k) = false$, for some k, i.e. either s_i violates s_{j+1}^k's writer policy or s_{j+1}^k violates s_i's reader policy, then replacing s_i may potentially result in a valid concrete service chain. For performance consideration (e.g. in case that all s_i, $1 \leq i \leq j$, may be replaced, replacing all such s_i may incur high overhead), we consider that the end user s_0 decides a threshold σ, $0 \leq \sigma \leq 1$, such that at most $\lceil \sigma \cdot j \rceil$ abstract services will be selected for re-instantiation. Note that, if $valid(s_i, s_{j+1}^k) = false$, for all k, $1 \leq k \leq |as_{j+1}|$, then s_i must be replaced. Let x be the total number of concrete services s_i, $1 \leq i \leq j$, such that $valid(s_i, s_{j+1}^k) = false$, for all k. Then, the number of abstract services to be re-instantiated $m = max\{\lceil \sigma \cdot j \rceil, x\}$. In order to maximize the probability of finding a valid concrete service chain, we intend to first replace the concrete service s_i which has most access control violations with the candidates for as_{j+1}, i.e. minimizing $avg_k\{valid(s_i, s_{j+1}^k)\}$, for all k, $1 \leq k \leq |as_{j+1}|$. We consider that s_j computes the fitness (for adjustment) for each concrete service s_i, $1 \leq i \leq j$, given a set of candidate concrete services S, $fit_A(s_i|S) = 1/(1+avg_k\{valid(s_i, s_{j+1}^k)\})$, for all k, $1 \leq k \leq |as_{j+1}|$, selects the top-m services, and generates a service chain Ch containing these m services in the descending order of their fitness. Then, we generate a length-m service chain Ch^* where, the l^{th} service, $1 \leq x \leq m$, in Ch^* is the prior service of the l^{th} service in Ch.

The second step is that s_j sends an adjustment request to each service included in the service chain generated by the first step. On receiving the adjustment request, the first service in the service chain will start the adjustment phase immediately, while all other services will wait for the adjustment result of its prior service in the chain.

Consider the following example to illustrate the adjustment process. Suppose that s_j, $j \geq 5$, generates an adjustment request containing the service chain $<s_2, s_4, s_0>$ (they are the services in the service chain to be adjusted). On receiving the adjustment request, s_2 will evaluate all the candidates for as_3, replace current s_3 with the most promising candidate service s_3', $s_3' \neq s_3$, and validate the newly composed service chain $<s_0, \ldots, s_3', \ldots, s_j>$. In case that the validation result is *true*, s_2 returns the adjustment result to s_j, and s_j takes over the composition task (adjustment is successful). If the result is *false*, s_2 tries another candidate. If s_2 cannot find any candidate to form a valid service chain, then s_2 informs s_4 about the failure. In this case, s_4 selects another candidate service for as_5 and requests s_2 to perform the adjustment again. If s_4 cannot find a candidate to form a valid service chain, s_4 informs s_0 about the failure. s_0 will select another candidate for as_1 and request s_4 to perform the adjustment again. If there are no candidate services for as_1 that can generate a valid concrete service chain (all the possible combinations for (s_3, s_5, s_1) are explored), s_0 records all the invalid concrete service chains, and propagates them together with a failure result to all the involved services. Note that, for performance consideration, s_0 may define another threshold τ, $0 \leq \tau \leq 1$, such that when the total number of invalid compositions of (s_3, s_5, s_1) exceeds $\lceil \tau \cdot |as_3| \cdot |as_5| \cdot |as_1| \rceil$, the adjustment phase will be terminated, and the composition process fails.

During the adjustment, each service s_{i-1} needs to evaluate all the candidate services for instantiating as_i, and select the most promising candidate, say s_i^k, for some k, for policy evaluation. The selection of s_i^k needs to maximize the likelihood of composing a valid service chain. Hence, s_{i-1} computes the fitness of each candidate service s_i^k, for all k, $1 \le k \le |as_i|$, given the set of composed concrete services S, i.e. $fit(s_i^k|S)$, where $S = \{s_0, s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_j\}$. The adjustment phase protocol is presented Figure 6.8.

Input: σ, τ, Output: *ch*.
1. $m' := 0$.
2. For each service s_i, $1 \le i \le j$,
   a) s_j computes $v(s_i) := \sum_k valid(s_i, s_{j+1}^k)$, for all k, $1 \le k \le |as_{j+1}|$.
   b) If $v(s_i) = 0$, $m' := m'+1$.
3. $m := max\{\lceil \sigma \cdot j \rceil, m'\}$.
4. Select m services from $\{s_1, \ldots, s_j\}$ with greatest $v(s_i)$, and generate a service chain *Ch* with the prior services of these top-m services .
5. $s_p :=$ REMOVETAIL(*Ch*).
6. $ch :=$ ADJUST(s_p, as_{p+1}, *Ch*, $\varnothing$).
7. Return *ch*.

ADJUST(s_p, as_{p+1}, *Ch*, *InvalidSet*)
1. If $|InvalidSet| > \tau \cdot \prod_{as \in Ch}|as|$, then return *nil*.
2. $s_q :=$ REMOVETAIL(*Ch*).
3. For each candidate service of as_{q+1}, s_{q+1}^k in S_{q+1},
   a) Compute $fit(s_{q+1}^k|\{s_0, \ldots, s_j\} - Ch - \{s_p\})$.
4. Sort S_{q+1} in the descending order of $fit(s_{q+1}^k|\{s_0, \ldots, s_j\} - Ch - \{s_p\})$.
5. If $Ch = \varnothing$ then,
   a) Compute $valid(<s_0, \ldots, s_{q+1}, \ldots, s_j>)$.
   b) If $valid(<s_0, \ldots, s_{q+1}, \ldots, s_j>) = true$, then return $<s_0, \ldots, s_{q+1}, \ldots, s_j>$, otherwise $InvalidSet := InvalidSet \cup \{<s_0, \ldots, s_{q+1}, \ldots, s_j>\}$ and return *nil*.
6. $ch :=$ ADJUST(s_q, as_{q+1}, *Ch*, *InvalidSet*).
7. If $ch \ne nil$, then return *ch*.
8. Remove s_{q+1} from S_{q+1}, and set $s_{q+1} := s_{q+1}^k$ with greatest fitness.
9. Compute $valid(<s_0, \ldots, s_{q+1}, \ldots, s_j>)$.
10. If $valid(<s_0, \ldots, s_{q+1}, \ldots, s_j>) = false$, then go to 8.
11. If $valid(<s_0, \ldots, s_{q+1}, \ldots, s_j>) = true$, then go to 6.

Figure 6.8. Adjustment phase protocol.

CHAPTER 7

RUN-TIME INFORMATION FLOW CONTROL WITH DEPENDENCY ANALYSIS

As discussed in Section 1.1.2, the action-level access control cannot guarantee the desired data resource level protection and, hence, a data resource level access control is necessary. However, the conventional data resource level access control models are insufficient for securing the composite services. In a composite service, there may be many data shared among different component services or written into the data storage of some domains. These data may be dynamically computed from many source data (i.e. the data resources stored in some domains) from different domains. As in conventional data resource level control, the access control policies are usually defined over preexisting data resources, it is very difficult to define access control policies over the dynamically computed data in these models.

Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. The output data that service s_i sends to s_{i+1}, $s_i.Out_F$, may be computed from the local input data resources of s_i and s_i's prior services, $s_{i-1}, \ldots, s_0$, i.e. $s_i.In_L, \ldots, s_0.In_L$. Hence, on receiving $s_i.Out_F$, s_{i+1} is the potential reader of $s_i.In_L, \ldots, s_0.In_L$ and it may be necessary to evaluate s_{i+1}'s attributes, $Attr(s_{i+1})$, against the reader IFC policies of $s_i.In_L, s_{i-1}.In_L, \ldots, s_0.In_L$. However, if s_j, $0 < j \leq i$, simply passes the data along, then s_{i+1} cannot derive the information contained in $s_j.In_L$ by receiving $s_i.Out_F$ (s_{i+1} is not the reader of $s_j.In_L$) and there is no need to evaluate s_{i+1} against the reader IFC policy of $s_j.In_L$. Also, if s_j, $0 < j \leq i$,

computes its output data solely based on its local input or has transformed its input completely, then s_{i+1} cannot derive any information contained in $s_{j-1}.In_L$, …, $s_0.In_L$ (s_{i+1} is not the reader of $s_{j-1}.In_L$, …, $s_0.In_L$) and there is no need to evaluate $s_{i+}1$ against the reader IFC policies of $s_{j-1}.In_L$, …, $s_0.In_L$. Similarly, the data that s_i writes to its local output data resource, $s_i.Out_L$, may be computed from the local input data resources of s_i and s_i's prior services, s_{i-1}, …, s_0. Hence, s_i, s_{i-1}, …, s_0 are all potential writers of $s_i.Out_L$ and it may be necessary to evaluate s_i's, …, s_0's attributes, $Attr(s_i)$, …, $Attr(s_0)$, against the writer IFC policy of $s_i.Out_L$. However, if s_j, $0 < j \leq i$, simply passes the data along (s_j is not the writer of $s_i.Out_L$), or if s_j computes its output data solely based on its local input or has transformed its input completely (s_{j-1}, …, s_0 s_{i+1} are not the writers of $s_i.Out_L$), then the writer information of $s_i.Out_L$ should be recomputed. Therefore, it is necessary to track the information flow, both inside and between the services and determine the data dependencies between locally stored data resources and dynamically computed data in order to make effective information flow control decisions.

In this chapter, we consider a fine-grained run-time information flow control model with data dependency analysis. Our model leverages existing information flow analysis techniques and expands them for data dependency analysis and information flow control in service chains. We first design a run-time information flow analysis system to derive the intra-service data dependencies. Then, we consider a dependency composition mechanism in which the intra-service dependencies are further computed into inter-service data dependencies. Based on the intra-service data dependencies, we determine the reader and writer information for the data resources used in service chains and perform both the write control to the stored data resources in each domain and the control of the information dissemination to services.

The rest of this chapter is organized as follows. Section 7.1 discusses the run-time dependency analysis system, including the derivation of both intra-service and inter-service data dependencies. Section 7.2 presents the global information flow control rules and the run-time information flow control protocol.

7.1 RUN-TIME DEPENDENCY ANALYSIS

Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. The information contained in the local input data resource of s_i, $s_i.In_L$, may be processed by $s_i, s_{i+1}, \ldots, s_{j-1}$, and computed into $s_{j-1}.Out_F$ which is delivered to the subsequent service s_j, $j > i$. To enable s_i to validate s_j on whether the information contained in $s_i.In_L$ can be released to s_j, it is necessary to determine how the data that s_j receives from s_{j-1}, $s_{j-1}.Out_F$, is dependent on the local input data of s_i, $s_i.In_L$. On the other hand, during s_j's computation, s_j may perform write operation over some of its local data resources, $s_j.Out_L$. The information written into $s_j.Out_L$ may be computed from some of s_i's local data resources, $s_i.In_L$, through a chain of intermediate services $s_i, s_{i+1}, \ldots, s_j$. To validate whether the information potentially computed from s_i's local data, $s_i.In_L$, can be written into $s_j.Out_L$, it is desired to determine how $s_j.Out_L$ is dependent on $s_i.In_L$. To derive such data dependencies, it is required to first derive the input/output dependency of each individual service, and then compute the intra-service dependencies into inter-service dependencies. In Section 7.1.1, we formally define the dependency set. In Section 7.1.2, we introduce a run-time dependency analysis system by which, the intra-service data dependencies can be derived. In Section 7.1.3, we discuss a dependency set composition mechanism which derives the inter-service data dependencies from the intra-service data dependencies.

To make the discussion in this section clearer, we consider an abstract language with the syntax given in Figure 7.1.

C ::=	*skip*	Empty statement
	| v := *op*(u_1, …, u_n)	Assignment
	| C_1; C_2	Sequential composition
	| *if con*(u_1, …, u_n) *then* C_1 *else* C_2 *end*	Conditional statement
	| *while con*(u_1, …, u_n) *do* C_1 *end*	Loop

Figure 7.1. An abstract language.

Note that we consider two types of expressions, including arithmetic/logical operation over a set of variables u_1, …, u_n, *op*(u_1, …, u_n), and condition over a set of variables *con*(u_1, …, u_n). Note that *op*() models constant.

7.1.1 Dependency Set

For a data v, where v can be an output data of a service s, $s.out_k \in s.Out$, for some k, or an internal variable of s, we consider the set of all input data and internal variables of s which "contributes" to the computation of v as the dependency set of v. The dependency set of a data/variable is formally defined in Definition 7.1.

Definition 7.1. For a service s, let u be an input data of s, $s.in_k \in s.In$, for some k, or an internal variable used by s, and let v be an output data of s, $s.out_l \in s.Out$, for some l, or an internal variable of s. v is called dependent on u, if and only if the change of the value of u may also change the value of v. The set of all data and variables that v is dependent on is called the dependency set of v, which is denoted by *Dep*(v). □

We consider two types of data dependencies in a service. In an assignment, e.g. "$v := u$", the dependency between u and v is called the *explicit dependency*. In a conditional statement or a loop, e.g. "*if u then* $v := 0$ *else* $v := 1$ *end*", the dependency between u and v is called the *implicit dependency*.

Note that we use $Dep_E(v)$ and $Dep_I(v)$ to denote the explicit and implicit dependency sets of v, respectively. Also, we use $Dep(U, v)$ to denote the set of all data in dataset U that v is dependent on. For example, we use $Dep(s.In, s.out_l)$ to represent the set of all input data of service s, $s.in_k \in s.In$, for all k, which an output data of s, $s.out_l \in s.Out$, for some l, is dependent on.

7.1.2 Run-Time Intra-Service Dependency Analysis

The problem of determining whether an output data of a service s, $s.out_l \in s.Out$, for some l, is dependent on an input data of s, $s.in_k \in s.In$, for some k, is equivalent to the problem of determining whether there is an information flow from $s.in_k$ to $s.out_l$. The equivalence can be easily proved from their definitions. That is, there is an information flow from u to v, if and only if the change of the value of u results in the change of the value of v (i.e. v is dependent on u). Hence, many static and dynamic information flow analysis mechanisms may be used to derive such data dependencies. However, most of the existing mechanisms are insufficient when they are used for dependency analysis. Static information flow analysis is impractical due to the occurrences of statically irresolvable variables in programs, such as array, pointer, database use, hash table, etc. Also, static approach examines all execution paths of a program [SHR07], while we only need to derive the dependency for the current execution path which may not be statically decided. On the other hand, dynamic information flow analysis mechanisms support only flow-

insensitive analysis. That is, it may not identify all implicit information flows (information flows caused by control flow) in a program [RUS10]. To provide a flow-sensitive run-time dependency analysis, it is necessary to analyze the non-executed code and, hence, additional mechanisms are required.

7.1.2.1 Basic Run-Time Dependency Analysis

The basic run-time intra-service dependency analysis system (RTISDA) is similar to the existing works in static information flow analysis [SAB03] but performs analysis at run-time. The system works in the following way. For each output data or internal variable of service *s*, *s.v*, it manages two dependency sets, $Dep_E(s.v)$ and $Dep_I(s.v)$, to store the explicit and implicit dependency sets of *s.v*, respectively. To identify the explicit dependencies, e.g. $v := u_1 + \ldots + u_n$, we simply include all the variables on the right-hand side of the assignment operator in the explicit dependency set of the variable on the left-hand, i.e., $Dep_E(v) = \{u_1, \ldots, u_n\}$. In order to identify the implicit dependencies, the RTISDA system manages a context stack $st(s)$ for each service *s* being analyzed. When the execution encounters the start of a conditional statement (*if*) or a loop (*while*), the set of all variables contained in the condition is pushed into the context stack as an entry. When the execution encounters the end of a conditional statement or a loop (*end*), the last entry (the top one) of the context stack is popped out. When evaluating each assignment, all the variables contained in the context stack will be included in the implicit dependency set of the variable on the left-hand side of the assignment operator. Consider the following program "*if* $u_1 > 0$ *then if* $u_2 < 0$ *then* $v_1 := 1$ *else* $v_1 := 0$ *skip end else* $v_2 := 0$ *end*". In this case, we have $Dep_I(v_1) = \{u_1, u_2\}$ and $Dep_I(v_2) = \{u_1\}$.

7.1.2.2 Flow-Sensitive Run-Time Dependency Analysis

Consider the program "*p* := 0; *q* := 1; *u* := *p*; *v* := *q*; *if w then u* := *q else skip end*; *if u* = *q then v* := *p else skip end*" to illustrate the flow-insensitivity issue of run-time dependency analysis. If *w* is true, the run-time analysis can identify the following dependencies, $Dep(u) = \{p, w, q\}$ and $Dep(v) = \{q, u, p, w\}$. However, if *w* is false, the identified dependencies are $Dep(u) = \{p\}$ and $Dep(v) = \{q\}$. However, as can be seen, if *w* is true, *u* = 1, *v* = 0, and if *w* is false, *u* = 0, *v* = 1. This implies that, the values of *u* and *v* are dependent on the value of *w*, regardless of whether *w* is true or false. Note that the dependencies between *w* and *u* and between *w* and *v*, are both implicit dependencies that should be captured regardless of the execution path. Hence, it is necessary to analyze (preferably statically) the non-executed paths of the service to complement the run-time analysis [GUE06, SHR07]. However, due to the statically irresolvable variables, static analysis is infeasible, and we consider a historical approach to provide a flow-sensitive dependency analysis. That is, we record the implicit dependencies derived from previous executions, and combine all these implicit dependencies to complement the result of next run-time analysis.

To provide a flow-sensitive run-time dependency analysis, we consider caching the implicit dependencies derived from previous executions. For each program, we insert a label before each keyword that may result in a change in the context stack (i.e. before each *if*, *while*, and *end* keyword). Moreover, we record the current number of iterations on encountering each label defined in a while-loop. Hence, each record in the cache is a tuple $(l_j^k, ni_j^k, s.v, Dep_I(s.v))$ where, l_j^k is j^{th} program label in k^{th} execution and ni_j^k is the number of loop iterations (0 for if-then statement) at j^{th} program label in k^{th} execution.

7.1.2.3 Dependency Analysis Rules

By combining all the above considerations, we present our dependency analysis rules which are followed by the RTISDA system in Figure 7.2.

DA_1.	$C(s) = \textit{if } con(s.u_1, \ldots, s.u_n) \Rightarrow st(s).push(\{s.u_1, \ldots, s.u_n\})$.
DA_2.	$C(s) = \textit{while } con(s.u_1, \ldots, s.u_n) \Rightarrow st(s).push(\{s.u_1, \ldots, s.u_n\})$.
DA_3.	$C(s) = \textit{end} \Rightarrow st(s).pop$.
DA_4.	$C(s) = s.v := op(s.u_1, \ldots, s.u_n) \wedge st(s) = \varnothing$ $\Rightarrow Dep_E(s.v) := \{s.u_1, ..., s.u_n\}, Dep_I(s.v) := \varnothing$.
DA_5.	$C(s) = s.v := op(s.u_1, \ldots, s.u_n) \wedge st(s) \neq \varnothing$ $\Rightarrow Dep_E(s.v) := \{s.u_1, ..., s.u_n\}$, $Dep_I(s.v) := Dep_I(s.v, l, ni) \cup (\cup v - Dep_E(s.v)), \forall v \in st(s.v)$.
DA_6.	$\forall s.u, s.w, \exists s.v,$ s.t. $s.v \in Dep_E(s.w) \wedge s.u \in Dep_E(s.v)$ $\Rightarrow Dep_E(s.w) := Dep_E(s.w) \cup \{s.u\}$.
DA_7.	$\forall s.u, s.w, s.u \notin Dep_E(s.w), \exists s.v,$ s.t. $(s.v \in Dep_I(s.w) \wedge s.u \in Dep_I(s.v)) \vee (s.v \in Dep_E(s.w) \wedge s.u \in Dep_I(s.v)) \vee (s.v \in Dep_I(s.w) \wedge s.u \in Dep_E(s.v))$ $\Rightarrow Dep_I(s.w) := Dep_I(s.w) \cup \{s.u\}$.

Figure 7.2. Dependency analysis rules.

Rules DA_1, DA_2, and DA_3 describe how the RTISDA manipulates the context stack of each service when encountering the start/end of a conditional statement or loop. Rules DA_4 evaluates the assignment to a variable *s.v*, which is not contained in the branch of a conditional statement or loop, to derive the explicit dependency set of *s.v*, which is the set of all operands contained in the assignment (implicit dependency set of *s.v* is empty). Rule DA_5 evaluates the assignment to a variable *s.v*, which is contained in the branch of a conditional statement or loop, to derive both the explicit and implicit dependency sets of *s.v*. The explicit dependency set is the set of all operands of the assignment. The implicit dependency set includes all the variables in the context

stack of the service and all the implicit dependent variables identified from historical executions at the same program label (each statement uses the last program label prior to the statement) and with same number of iterations. Rules DA_6 and DA_7 derive the dependencies caused by transitivity. Rule DA_6 says that, if variable *s.w* is explicitly dependent on variable *s.v* and *s.v* is in turn explicitly dependent on variable *s.u*, *s.w* is also explicitly dependent on *s.u*. Rule DA_7 says that, if variable *s.w* is explicitly dependent on variable *s.v* and *s.v* is implicitly dependent on variable *s.u*, or if variable *s.w* is implicitly dependent on variable *s.v* and *s.v* is explicitly dependent on variable *s.u*, or both dependencies are implicit, *s.w* is also implicitly dependent on *s.u* (*s.w* must not be already explicitly dependent on *s.u*).

7.1.3 Run-Time Inter-Service Dependency Composition

The run-time dependency analysis system can only be used to derive the dependency relationship of a single web service. To derive the dependency relationship between arbitrary data generated or consumed by services in a service chain, an additional dependency composition mechanism is required. Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. To derive the dependency set $Dep(s_{i-1}.In_L, s_i.out_{Fk})$, for all $s_i.out_{Fk} \in s_i.Out_F$, s_i needs to first derive $Dep(s_i.In_F, s_i.out_{Fk})$, and for each data v, $v \in Dep(s_i.In_F, s_i.out_{Fk})$, s_{i-1} needs to derive the dependency set $Dep(s_{i-1}.In_L, v)$. Then, $Dep(s_{i-1}.In_L, s_i.out_{Fk}) = \cup Dep(s_{i-1}.In_L, v)$, for all v, $v \in Dep(s_i.In_F, s_i.out_{Fk})$. Similarly, To derive the dependency set $Dep(s_i.In_F, s_{i+1}.out_{Lk})$, for all $s_{i+1}.out_{Lk}$, $s_{i+1}.out_{Lk} \in s_{i+1}.Out_L$, s_{i+1} needs to derive $Dep(s_i.Out_F, s_{i+1}.out_{Lk})$ first, and for each data $v \in Dep(s_i.Out_F, s_{i+1}.out_{Lk})$, s_i derives the dependency set $Dep(s_i.In_F, v)$. Then, $Dep(s_i.In_F, s_{i+1}.out_{Lk}) = \cup Dep(s_i.In_F, v)$, for all v, $v \in Dep(s_i.Out_F, s_{i+1}.out_{Lk})$.

Using this basic idea, we consider each service chain associated with a dependency graph to represent the dependency relationships between the data generated and consumed by services in the service chain. The dependency graph is defined as follows.

Definition 7.2. The dependency graph of a service chain, $<s_0, s_1, \ldots, s_{n+1}>$, $G_D(<s_0, s_1, \ldots, s_{n+1}>) = (V_D, E_D)$ is a directed graph where, the vertex set V_D includes all the input data resources of s_i, $s_i.in_{Lk} \in s_i.In$, for all k, and all the output data resources of s_i, $s_i.out_l \in s_i.Out$, for all l, and for any two data resources $u, v \in V_D$, $(v, u) \in E_D$ if and only if v is dependent on u. □

Note that as $s_{i+1}.in_{Fk} = s_i.out_{Fk}$, for the same k, $s_{i+1}.in_{Fk}$ are also considered to be dependent on $s_i.out_{Fk}$ and, hence, $(s_{i+1}.in_{Fk}, s_i.out_{Fk}) \in E_D$. Consider s_i and s_j, $j > i$. The dependency set $Dep(s_i.In_L, s_{j-1}.out_{Fk})$, for some k, can be derived by checking, for each data resource $s_i.in_{Ll} \in s_i.In_L$, whether there is a path from $s_{j-1}.out_{Fk}$ to $s_i.in_{Ll}$. The dependency set composition mechanism is presented as follows.

Input: $Dep(s_i.In, v)$, for all i, $0 \le i \le n+1$, for all $v \in s_i.Out$,
Output: $G_D(<s_0, s_1, \ldots, s_n, s_{n+1}>) = (V_D, E_D)$.
1. $V_D := \varnothing$, $E_D := \varnothing$.
2. For all service s_i, $0 \le i \le n+1$,
   a) For all k, $1 \le k \le |s_i.In|$, $V_D := V_D \cup s_i.in_k$.
   b) For all l, $1 \le l \le |s_i.Out|$, $V_D := V_D \cup s_i.out_j$.
   c) For all k, l, $1 \le k \le |s_i.In|$, $1 \le l \le |s_i.Out|$,
      i. If $s_i.in_k \in Dep(s_i.In, s_i.out_l)$, then $E_D := E_D \cup (s_i.out_l, s_i.in_k)$.
3. For all service s_i, $0 \le i \le n$,
   a) For all k, $1 \le k \le |s_i.Out_F|$, $E_D := E_D \cup (s_i.out_{Fk}, s_{i+1}.in_{Fk})$.

Figure 7.3. Dependency graph generator.

Input: $G_D(<s_0, s_1, \ldots, s_n, s_{n+1}>)$, $U \subseteq V_D$, $v \in V_D$,
Output: $Dep(U, v)$.
1. $Dep(U, v) := \varnothing$.
2. For all $u \in U$,
   a) If there exists a directed path in G_D from v to u, then $Dep(U, v) := Dep(U, v) \cup \{u\}$.

Figure 7.4. Dependency set composer.

7.2 RUN-TIME INFORMATION FLOW CONTROL WITH DEPENDENCY ANALYSIS

Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. The output data that s_i sends to s_{i+1}, $s_i.Out_F$, may be dependent on the local data resources of s_i and s_i's prior services, $s_{i-1}, \ldots, s_0$, i.e. $s_i.In_L, \ldots, s_0.In_L$. As $s_i.In_L, \ldots, s_0.In_L$ may include the sensitive information stored in $dom(s_i), \ldots, dom(s_0)$, when $s_i.Out_F$ is delivered to the subsequent service s_{i+1}, s_{i+1} may derive the sensitive information contained in $s_i.In_L, \ldots, s_0.In_L$ from the data it receives. On the other hand, the information that s_i writes to its local data resource, $s_i.Out_L$, x, may be dependent on the local input data resources of $s_i, \ldots, s_0$, i.e. $s_i.In_L, \ldots, s_0.In_L$. As $s_i.In_L, \ldots, s_0.In_L$ may include the spurious data stored in $dom(s_{i-1}), \ldots, dom(s_0)$, when x is written into $s_i.Out_L$, $s_i.Out_L$ may be contaminated.

To secure the sensitive data resources from undesired read/write, each service s_i defines detailed information flow control policies (discussed in Section 4.3) to control the read/write accesses to its sensitive data resources, and the system defines global information flow control rules (discussed in Section 5.1) to govern the validation process of an entire service chain. As the dependencies between the locally stored data resources and the dynamically generated data in service chains have major impacts on the information flow control decisions, we define

additional global information flow control rules to reflect this impact. Note that as the data dependencies may not be statically determined, a run-time information flow control mechanism is necessary. In Section 7.2.1, we discuss the global information flow control rules with data dependency analysis. In Section 7.2.2, we present the run-time information flow control protocol.

7.2.1 Global Information Flow Control Rules with Data Dependency

Consider the service chain $<s_0, s_1, \ldots, s_{n+1}>$. According to Section 5.1, each service s_i, $0 \leq i < n+1$, needs to verify all its subsequent services in the service chain, s_j, $i < j \leq n+1$, by evaluating s_j's attributes, $Attr(s_j)$, $s_i.In_L$'s attributes, $Attr(s_i.In_L)$, and the maximal transformation factor of the intermediate services between s_i and s_j, $max\{tf(s_i), \ldots, tf(s_{j-1})\}$, against s_i's reader information flow control policies, $Pol_R(s_i)$. Also, s_i needs to verify all its prior services in the service chain, s_j, $0 \leq j < i$, by evaluating s_j's attributes, $Attr(s_j)$, $s_i.Out_L$'s attributes, $Attr(s_i.Out_L)$, and the maximal transformation factor of the intermediate services between s_i and s_j, $max\{tf(s_j), \ldots, tf(s_i)\}$, against s_i's writer information flow control policies, $Pol_W(s_i)$.

In Section 5.1.2, we have considered two special situations regarding transformation factor under which the mutual validation between two services can be skipped. In this section, we define additional global information flow control rules by considering the data dependencies. If some input data resources that s_j, $i < j$, receives from s_{j-1}, $s_j.in_{Fk}$, for some k, is not dependent on some of s_i's local input data resources, $s_i.in_{Ll}$, for some l, then s_j is not the reader service of $s_i.in_{Ll}$ and there is no need to evaluate s_j's information against the reader IFC policy defined for $s_i.in_{Ll}$. Similarly, if the data that s_j writes to some of its local data resources, $s_j.out_{Lk}$, for some k, is not

dependent on some of the local input data resources of s_i, $i < j$, $s_i.in_{Ll}$, for some l, then s_i is not the writer service of $s_j.out_{Lk}$ and there is no need to evaluate s_i's information against the writer IFC policy defined for $s_j.out_{Lk}$. The global information flow control rules are defined in Figure 7.5.

$DGIFC_1$.	$s_i.in_{Ll} \notin Dep(s_j.in_{Fk})$, $1 \le k \le \lvert s_j.In_F \rvert$, $1 \le l \le \lvert s_i.In_L \rvert$ $\Rightarrow (allow(s_i.in_{Ll}, s_j.in_{Fk}) := true)$, $\forall i, j, 0 \le i < j \le n+1$.
$DGIFC_2$.	$s_i.in_{Ll} \notin Dep(s_j.out_{Lk})$, $1 \le k \le \lvert s_j.Out_L \rvert$, $1 \le l \le \lvert s_i.in_L \rvert$ $\Rightarrow (allow(s_i.in_{Ll}, s_j.out_{Lk}) := true)$, $\forall i, j, 0 \le i < j \le n+1$.
$DGIFC_3$.	$allow(s_i.in_{Ll}, s_j.in_{Fk}) = true$ for all k, l, $1 \le k \le \lvert s_j.In_F \rvert$, $1 \le l \le \lvert s_i.In_L \rvert$ $\Rightarrow (valid_R(s_i, s_j) := true)$, $\forall i, j, 0 \le i < j \le n+1$.
$DGIFC_4$.	$allow(s_i.in_{Ll}, s_j.out_{Lk}) = true$ for all k, l, $1 \le k \le \lvert s_j.Out_L \rvert$, $1 \le l \le \lvert s_i.in_L \rvert$ $\Rightarrow (valid_W(s_i, s_j) := true)$, $\forall i, j, 0 \le i < j \le n+1$.

Figure 7.5. Dependency-based global information flow control rules.

Note that, $allow(x, y)$ denotes whether the information flow from data x to data y is allowed (*true*).

7.2.2 Run-Time Information Flow Control Protocol

By following the global information flow control rules and specifying detailed information flow control policies, each service in a service chain can validate both the consecutive and nonconsecutive services in the service chain. However, the dependency set $Dep(s_j.in_{Fk})$, $0 < j \le n+1$, can only be derived after $s_{j-1}.Out_F$ (Note that $s_{j-1}.Out_F$ is equal to $s_j.In_F$) is generated. Hence, the prior services of s_{j-1}, s_i, $0 \le i < j-1$, cannot make an effective information flow control decision when sending out $s_i.Out_F$, from which $s_{j-1}.Out_F$ is computed. Therefore, we consider a run-time information flow control protocol with back-check and carry-along policy [SHE09].

In the back-check protocol, in order for s_i to make an effective information flow control decision on whether $s_{j-1}.Out_F$ can be delivered to s_j, each intermediate service s_k, $i < k < j-1$, needs to derive the dependency graph of the sub service chain $<s_0, \ldots, s_k>$, $G_D(<s_0, \ldots, s_k>)$, and send it to s_{k+1}. On delivering $s_{j-1}.Out_F$ to s_j, s_{j-1} needs to send the derived dependency graph $G_D(<s_0, \ldots, s_{j-1}>)$ back to s_i, and s_i derives the dependency set $Dep(s_i.In_L, s_{j-1}.out_{Fl})$, for all l, $1 \leq l \leq |s_{j-1}.Out_F|$, and validates s_j using $DGIFC_1$ and $BGIFC_1$. On the other hand, s_j, on writing its local data resource, $s_j.Out_L$, has already have the dependency graph $G_D(<s_0, \ldots, s_j>)$ and, hence, can make an effective information flow control decision.

We present the run-time information flow control protocol in Figure 7.6. Note that, we assume that the transformation factors and attributes of all services in the service chain $<s_0, s_1, \ldots, s_{n+1}>$ are exchanged at the composition time, and are known in advance here.

1. s_i receives s_{i-1}'s request, req_{i-1}, which includes a set of data $s_{i-1}.Out_F = \{s_{i-1}.out_{F1}, s_{i-1}.out_{F2}, \ldots\}$, a set of carry-along policies $\{Pol_C(s_0), \ldots, Pol_C(s_{i-1})\}$, a set of attribute sets $\{\{Attr(s_0.in_{L1}), Attr(s_0.in_{L2}), \ldots\}, \ldots, \{Attr(s_{i-1}.in_{L1}), Attr(s_{i-1}.in_{L2}), \ldots\}\}$, and the dependency graph of sub service chain $<s_0, \ldots, s_{i-1}>$, $G_D(<s_0, \ldots, s_{i-1}>)$.
2. s_i computes $s_i.Out = s_i.Out_F \vee s_i.Out_L$ from $s_i.In_F$ and $s_i.In_L$.
3. s_i derives the dependency graph of the sub service chain $<s_0, \ldots, s_i>$, $G_D(<s_0, \ldots, s_i>)$.
4. For each service s_j, $0 \leq j < i$,
   a) For each $(s_j.in_L, s_i.out_L)$, $s_j.in_L \in s_j.In_L$, $s_i.out_L \in s_i.Out_L$,
      i. s_i computes $allow(s_i.in_L, s_j.out_L)$ using $DGIFC_2$.
      ii. If $allow(s_i.in_L, s_j.out_L) \neq true$, then s_i computes $allow(s_i.in_L, s_j.out_L) := auth(Attr(s_i), Attr(s_j.out_L), max\{tf(s_j), \ldots, tf(s_i)\}, Pol_W(s_j))$.
   b) s_i computes $valid_W(s_j, s_i)$.
5. If $valid_W(s_j, s_i) = false$, for any j, send *false* to s_{i-1}; otherwise, continue.
6. For each service s_j, $0 \leq j \leq i$,
   a) For each $s_j.in_L \in s_j.In_L$,
      i. s_i computes $allow(s_j.in_L, s_{i+1}.in_F)$ using $DGIFC_1$.
      ii. If $allow(s_j.in_L, s_{i+1}.in_F) \neq true$ and $Pol_C(s_j) \neq \varnothing$, s_i computes $allow(s_j.in_L, s_{i+1}.in_F) := auth(Attr(s_{i+1}), Attr(s_j.in_L), max\{tf(s_j), \ldots, tf(s_i)\}, Pol_C(s_j))$.
      iii. If $allow(s_j.in_L, s_{i+1}.in_F) \neq true$ and $Pol_C(s_j) = \varnothing$ or $Pol_C(s_j)$ does not yield an effective decision, s_i sends the dependency graph $G_D(<s_0, \ldots, s_i>)$ and $Attr(s_{i+1})$ to s_j, and waits for s_j's information flow control decision, which is decided by s_j by evaluating $Attr(s_{i+1})$, $Attr(s_j.in_L)$, and $max\{tf(s_j), \ldots, tf(s_i)\}$ against $Pol_R(s_j)$.
   b) s_i computes $valid_R(s_j, s_{i+1})$.
7. If $valid_R(s_j, s_{i+1}) = false$, for any j, s_i sends *false* to s_{i-1}; otherwise, generates a new request req_i which includes a set of data $s_i.Out_F = \{s_i.out_{F1}, s_i.out_{F2}, \ldots\}$, a set of carry-along policies $\{Pol_C(s_0), \ldots, Pol_C(s_i)\}$, a set of attribute sets $\{\{Attr(s_0.in_{L1}), Attr(s_0.in_{L2}), \ldots\}, \ldots, \{Attr(s_i.in_{L1}), Attr(s_i.in_{L2}), \ldots\}\}$, and the dependency graph of sub service chain $<s_0, \ldots, s_i>$, $G_D(<s_0, \ldots, s_i>)$.

Figure 7.6. Run-time information flow control protocol.

CHAPTER 8

CONCLUSION AND FUTURE RESEARCH

In this thesis, we have considered various issues in secure service composition and provided a comprehensive set of solutions. First, we consider the issue of secure information flow in composite services and develop a fine-grained information flow control model. We also introduce the novel concept of transformation factor to model the computation and transformation effect of intermediate services upon the information flows between two indirectly interacting services. The transformation factor can great simplify the access control policies and protocols for indirectly interacting services.

Second, we consider the issue in integrating action-level and data resource level access control mechanisms to secure web services, which has not been carefully considered in the literature. We propose to complement action-level access control with fine-grained data resource level protection. We extend conventional data resource level access control models by using the run-time information flow tracking technique to derive the dependencies between the locally stored data in each domain and the dynamically generated data in composite services, to secure the data accesses within web services.

Third, we consider the issue of performing execution-time and composition-time access control validation and show the benefit of evaluating access control policies at service composition time. We consider a mediator-based secure composition protocol and a mediator-

free distributed composition mechanism. In the mediator-based approach, we consider a three-phase composition protocol to resolve the performance issue and the trusted service composer issue in composition-time access control. In the mediator-free approach, we consider a fully decentralized composition protocol by introducing the novel concepts of the back-check process and carry-along policy to facilitate the interactions between distributed composition entities. We have also developed an adjustment protocol to achieve more efficient distributed service composition.

There are several future research directions that depart from the secure service composition mechanism discussed in this thesis. First, our secure service composition mechanism is developed based on the attribute-based access control due to its generosity of instantiating various access control models. Our mechanism can also be coupled with other more realistic access control models, such as role-based access control [FER01], multilevel security model [BEL73], usage control [PAR04], etc., to provide better protection for the web service environment.

Second, in this thesis, we only consider the sequentially composed composite services, i.e. service chains. The mechanism can be extended to general workflows which involve parallel and conditional compositions and loops, but there are additional issues that need to be carefully considered. For instance, the sensitive information stored in each domain and/or the data generated from the sensitive information may be used for branching and, hence, the execution of the workflow (specifically, which branch is selected) may be used to derive the sensitive information. Consider the workflow which includes a conditional composition, *if* $x < 1$ *then execute*(s_1) *else execute*(s_2), where x is sensitive. By determining whether s_1 or s_2 is executed, it

is possible to determine whether x is less than 1. To secure general composite services, such information flows should be considered.

Third, we may further develop a more sophisticated model for transformation factors of services. A service may have different transformation factors for different input parameters. For example, the output data of service s_i, $s_i.Out_F$, may include the sensitive information contained in the data that s_i receives from s_{i-1}, $s_i.In_F$, but does not include any critical information in s_i's local data, $s_i.In_L$. In this case, the transformation factor of s_i regarding the information flow from $s_i.In_F$ to $s_i.Out_F$ should be HR but the transformation factor of s_i regarding the information flow from $s_i.In_L$ to $s_i.Out_F$ should be NR. Also, a service operation may have multiple execution paths and different execution paths may have different transformation factor levels. In the future, we may further investigate the issues in these research directions to develop more sophisticated secure service composition mechanisms.

REFERENCES

[AGA04] S. Agarwal and B. Sprick, "Access control for semantic web services," IEEE International Conference on Web Services, pp.770-773, 2004.

[APA05] Apache, "Axis architecture guide," http://ws.apache.org/axis/java/ architecture-guide.html, 2005.

[AHS09] M. Ahsant and J. Basney, "Workflows in dynamic and restricted delegation," IEEE International Conference on Availability, Reliability, and Security, pp. 17-24, 2009.

[ARD11] C.A. Ardagna, S.D.C.D. Vimercati, S. Paraboschi, E. Pedrini, P. Samarati, M. Verdicchio, "Expressive and deployable access control in open web service applications," IEEE Transactions on Services Computing, vol. 4, no. 2, pp. 96-109, 2011.

[ASK09] A. Askarov and A. Sabelfeld, "Tight enforcement of information-release policies for dynamic languages," Computer Security Foundations Symposium, pp. 43-59, 2009.

[AUR98] T. Aura, "On the structure of delegation networks," Proceedings of Computer Security Foundations Workshops, pp. 14-26, 1998.

[AUS92] T.M. Austin and G.S. Sohi, "Dynamic dependency analysis of ordinary programs," International Symposium on Computer Architecture, pp. 342-351, 1992.

[BAN02] O.L. Bandmann, B.S. Firozabadi, M. Dam, "Constrained delegation," IEEE Symposium on Security and Privacy, pp. 131-142, 2002.

[BAR06] M. Bartoletti, P. Degano, G.L. Ferrari, "Security issues in service composition," Lecture Notes in Computer Science, pp. 1-16, 2006.

[BAR08] M. Bartoletti, P. Degano, G.L. Ferrari, R. Zunino, "Semantics-based design for secure web services", IEEE Transactions on Software Engineering, vol. 34, no. 1, pp. 33-49, 2008.

[BEL73] D.E. Bell and L.J. LaPadula, "Secure computer systems: mathematical foundations and model," The MITRE Corporation Bedford MA Technical Report M74244, 1973.

[BER04] E. Bertino, A.C. Squicciarini, D. Mevi, "A fine-grained access control model for web services," IEEE International Conference on Services Computing, pp. 33-40, 2004.

[BER06] E. Bertino, A.C. Squicciarini, L. Martino, F. Paci, "An adaptive access control model for web services," International Journal of Web Service Research, pp. 27-60, 2006.

[BHA04] R. Bhatti, E. Bertino, A. Ghafoor, "A trust-based context-aware access control model for web-services," IEEE International Conference on Web Services, pp. 184-191, 2004.

[BON96] P.A. Bonatti, M.L. Sapino, V.S. Subrahmanian, "Merging heterogeneous security orderings," European Symposium on Research in Computer Security, pp. 183-197, 1996.

[CAR06] B. Carminati, E. Ferrari, P.C.K. Hung, "Security conscious web service composition," IEEE International Conference on Web Services, pp. 489-496, 2006.

[CHA05] G. Chafle, S. Chandra, V. Mann, M.G. Nanda, "Orchestrating composite web services under data flow constraints", IEEE International Conference on Web Services, pp. 211-218, 2005.

[COE04] M. Coetzee and J.H.P. Eloff, "Towards web service access control," Computers & Security, vol. 23, no. 7, pp. 559-570, 2004.

[CRA08a] J. Crampton and H. Khambhammettu, "Delegation in role-based access control," International Journal of Information Security, vol. 7, no. 2, pp. 123-136, 2008.

[CRA08b] J. Crampton and H. Khambhammettu, "On delegation and workflow execution models," ACM symposium on Applied computing, pp. 2137-2144, 2008.

[DAM01] E. Damiani, S.D.C.D. Vimercati, S. Paraboschi, P. Samarati, "Fine grained access control for SOAP e-services," ACM International Conference on World Wide Web, pp. 504-513, 2001.

[DAW00] S. Dawson, S. Qian, P. Samarati, "Providing security and interoperation of heterogeneous systems," Distributed and Parallel Databases, vol. 8, pp. 119-145, 2000.

[DEH06] S. Dehousse, L. Liu, C. Chi, S. Faulkner, "Delegation models in service oriented systems," IEEE International Symposium on Service-Oriented System Engineering, pp. 85-94, 2006.

[DEN03] G. Denker, L. Kagal, T. Finin, M. Paolucci, K. Sycara, K., "Security for DAML web services: annotation and matchmaking," Lecture Notes in Computer Science, vol. 2870/2003, pp. 335-350, 2003.

[DEN76] D.E. Denning, "A lattice model of secure information flow," Communications of the ACM, vol. 19, no. 5, pp. 236-243, 1976.

[DEN77] D.E. Denning and P.J. Denning, "Certification of programs for secure information flow," Communications of the ACM, vol. 20, no. 7, pp. 504-513, 1977.

[ELL99] C. Ellison, B. Frantz, B. Lampson, R Rivest, B. Thomas, T. Ylonen, "SPKI certificate theory," Internet RFC 2693, http://www.ietf.org/rfc/rfc2693.txt, 1999.

[EUZ04] J. Euzenat and P. Valtchev, "Similarity-based ontology alignment in OWL-lite," Proceedings of the European Conference on Artificial Intelligence, pp. 333-337, 2004.

[FAR02] S. Farrell and R. Housley, "An internet attribute certificate profile for authorization," Internet RFC3281, http://www.ietf.org/rfc/rfc3281.txt, 2002.

[FEN06] Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou, Youfeng Wu, "LIFT: a low-overhead practical information flow tracking system for detecting security attacks," IEEE/ACM International Symposium on Microarchitecture, 2006, pp. 135-149.

[FER01] D.F. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, R. Chandramouli, "Proposed NIST standard for role-based access control," ACM Transactions on Information and System Security, vol. 4, no. 3, 224-274, 2001.

[FIS08] J. Fischer and R. Majumdar, "A theory of role composition," Proceedings of IEEE International Conference on Web Services, pp. 320-328, 2008.

[GOG82] J.A. Goguen and J. Meseguer, "Security policies and security models," IEEE Symposium on Security and Privacy, pp. 11-20, 1982.

[GON96] Li Gong and Xiaolei Qian, "Computational issues in secure interoperation," IEEE Transactions on Software Engineering, vol. 22, no. 1, pp. 43-52, 1996.

[GUE06] G.L. Guernic, A. Banerjee, T. Jensen, D. Schmidt, "Automata-based confidentiality monitoring," Lecture Notes in Computer Science, vol. 4435/2006, pp. 75-89, 2006.

[HAN06] Jun Han, R. Kowalczyk, K.M. Khan, "Security-oriented service composition and evolution," Asia-Pacific Software Engineering Conference, pp. 71-78, 2006.

[HEB09] R.N. Hebig, C. Meinel, M. Menzel, I. Thomas, R. Warschofsky, "A web service architecture for decentralised identity- and attribute-based access control," IEEE International Conference on Web Services, pp. 551-558, 2009.

[HWA06] Hyun Sik Hwang, Hyuk Jin Ko, Kyu Il Kim, Ung Mo Kim, Dong Soon Park, "Agent-based delegation model for the secure web service in ubiquitous computing environments," IEEE International Conference on Hybrid Information Technology, vol. 1, pp. 51-57, 2006.

[IBM03] IBM, "Business process execution language for web services version 1.1", http://public.dhe.ibm.com/software/dw/specs/ws-bpel/ws-bpel.pdf, 2003.

[JOS00] R. Joshi and K.R.M. Leino, "A semantic approach to secure information flow," Science of Computer Programming, vol. 37, no. 1-3, pp. 113-138, 2000.

[KAG03] L. Kagal, T. Finin, J. Anupam, "A policy language for a pervasive computing environment," IEEE International Workshop on Policies for Distributed Systems and Networks, 2003.

[KAG04] L. Kagal, M. Paolucci, N. Srinivasan, G. Denker, T. Finin, K. Sycara, "Authorization and privacy for semantic web services," IEEE Intelligent Systems, vol. 19, no. 4, pp. 50-56, 2004.

[KHU03] S. Khurshid, C.S. PAsAreanu, W. Visser, "Generalized symbolic execution for model checking and testing," Tools and Algorithms for the Construction and Analysis of Systems, pp. 553-568, 2003.

[KIN76] J.C. King, "Symbolic execution and program testing," Communications of the ACM, vol. 19, no. 7, pp. 385-394, 1976.

[LAM06] Lap Chung Lam and Tzi-cker Chiueh, "A general dynamic information flow tracking framework for security applications," Computer Security Applications Conference, pp. 463-472, 2006.

[LAM74] B.W. Lampson, "Protection," ACM SIGOPS Operating Systems Review, vol. 8, no. 1, 1974.

[LI02] Ninghui Li, J.C. Mitchell, W.H. Winsborough, "Design of a role-based trust management framework," Proceedings of the IEEE Symposium on Security and Privacy, pp. 114-130, 2002.

[MAB08] M. Mabuchi, Y. Shinjo, A. Sato, K. Kato, "An access control model for web-services that supports delegation and creation of authority," IEEE International Conference on Networking, pp. 213-222, 2008.

[MOR88] M. Morgenstern, "Controlling logical inference in multilevel database systems," The Proceedings of 1988 IEEE Symposium on Security and Privacy, pp. 245-255, 1988.

[NAN04] M.G. Nanda, S. Chandra,V. Sarkar, "Decentralizing execution of composite web services", ACM SIGPLAN Conference on Object-oriented Programming, systems, languages, and applications, 2004.

[NAT04] F.N. Natalya, "Semantic integration: a survey of ontology-based approaches", ACM Special Interest Group on Management of Data, vol. 33, no. 4, pp. 66-70, 2004.

[OAS05] OASIS, "Extensible access control markup language (XACML) version 2.0," http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2005.

[OAS06] OASIS, "Web services security: SOAP message security 1.1," http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf, 2006.

[OAS07a] OASIS, "WS-Trust 1.3," http://docs.oasis-open.org/ws-sx/ws-trust/200512/ ws- trust-1.3-os.doc, 2007.

[OAS07b] OASIS, "WS-SecureConversation 1.3," http://docs.oasis-open.org/ws-sx/ ws-secureconversation/200512/ws-secureconversation-1.3-os.doc, 2007.

[OAS07c] OASIS, "WS-SecurityPolicy 1.2," http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.doc, 2007.

[OAS07d] OASIS, "Web services profile of XACML (WS-XACML) version 1.0," http://www.oasis-open.org/committees/download.php/24951/xacml-3.0-profile-webservices- spec-v1-wd-10-en.pdf, 2007.

[OAS09] OASIS, "Assertions and protocols for the OASIS security assertion markup language (SAML) v2.0," http://www.oasis-open.org/committees/download.php/ 35711/sstc-saml-core-errata-2.0-wd-06-diff.pdf, 2009.

[OLS06] L. Olson, M. Winslett, G. Tonti, N. Seeley, A. Uszok, J. Bradshaw, "Trust negotiation as an authorization service for web services," The Proceedings of International Conference on Data Engineering Workshops, 2006.

[PAC08] F. Paci, M. Ouzzani, M. Mecella, "Verification of access control requirements in web services choreography," IEEE International Conference on Services Computing, pp. 5-12, 2008.

[PAC11] F. Paci, M. Mecella, M. Ouzzani, E. Bertino, "ACCONV - an access control model for conversational web services," ACM Transactions on the Web, vol. 5, no. 3, 2011.

[PAR04] Jaehong Park and R.S. Sandhu, "The $UCON_{ABC}$ usage control model," ACM Transactions on Information and System Security, vol. 7, no. 1, pp. 128-174, 2004.

[PAR09] S. Paradesi, P. Doshi, S. Swaika, "Integrating behavioral trust in web service compositions," IEEE International Conference on Web Services, pp. 453-460, 2009.

[QIN06] Feng Qin, Cheng Wang, Zhenmin Li, Ho-seop Kim, Yuanyuan Zhou, Youfeng Wu, "LIFT: a low-overhead practical information flow tracking system for detecting security attacks," IEEE/ACM International Symposium on Microarchitecture, pp. 135-149, 2006.

[RHA05] R. Bhatti, A. Ghafoor, E. Bertino, J.B.D. Joshi, "X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control," ACM Transactions on Information and System Security, vol. 8, no. 2, pp. 187-227, 2005.

[RUS10] A. Russo and A. Sabelfeld, “Dynamic vs. static flow-sensitive security analysis,” Computer Security Foundations Symposium, pp. 186-199, 2010.

[SAB01] A. Sabelfeld and D. Sands, “A per model of secure information flow in sequential programs,” Higher-Order and Symbolic Computation, vol. 14, no. 1, pp. 59-91, 2001.

[SAB03] A. Sabelfeld and A.C. Myers, “Language-based information-flow security,” IEEE Journal on Selected Areas in Communications, vol. 21, no. 1, pp. 5-19, 2003.

[SAB10] A. Sabelfeld and A. Russo, “From dynamic to static and back: riding the roller coaster of information-flow control research,” Lecture Notes in Computer Science, vol. 5947/2010, pp. 352-365, 2010.

[SHA05] B. Shafiq, J.B.D. Joshi, E. Bertino, A. Ghafoor, “Secure interoperation in a multidomain environment employing RBAC policies,” IEEE Transactions on Knowledge and Data Engineering, vol. 17, no. 11, pp. 1557-1577, 2005.

[SHE05] M. Shehab, E. Bertino, A. Ghafoor, “Secure collaboration in mediator-free environments,” ACM Conference on Computer and Communications Security, pp. 58-67, 2005.

[SHE07] Wei She, B. Thuraisingham, I-Ling Yen, “Delegation-based security model for web services,” IEEE High Assurance Systems Engineering Symposium, pp. 82-91, 2007.

[SHE09a] Wei She, I-Ling Yen, B. Thuraisingham, E. Bertino, “The SCIFC model for information flow control in web service composition”, IEEE International Conference on Web Services, pp. 1-8, 2009.

[SHE09b] Wei She, I-Ling Yen, B. Thuraisingham, E. Bertino, “Effective and efficient implementation of an information flow control protocol for service composition”, IEEE International Conference on Service-Oriented Computing and Applications, pp. 1-8, 2009.

[SHE10a] Wei She, I-Ling Yen, B. Thuraisingham, “WS-Sim: a web service simulation toolset with realistic data support”, IEEE Signature Conference on Computer Software and Applications Workshop, pp.109-114, 2010.

[SHE10b] Wei She, I-Ling Yen, B. Thuraisingham, E. Bertino, “Policy-driven service composition with information flow control,” IEEE International Conference on Web Services, pp. 50-57, 2010.

[SHR07] P. Shroff, S. Smith, M. Thober, “Dynamic dependency monitoring to secure information flow,” IEEE Computer Security Foundations Symposium, pp. 203-217, 2007.

[SKO03] H. Skogsrud, B. Benatallah, F. Casati, "Model-driven trust negotiation for web services," IEEE Internet Computing, pp. 45-51, 2003.

[SRI07] M. Srivatsa, A. Iyengar, T. Mikalsen, I. Rouvellou, Jian Yin, "An access control system for web service compositions", IEEE International Conference on Web Services, pp. 1-8, 2007.

[STA03] J. Staddon, "Dynamic inference control," Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, pp. 94-100, 2003.

[SUH04] G.E. Suh, J.W. Lee, D. Zhang, S. Devadas, "Secure program execution via dynamic information flow tracking," International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 1-14, 2004.

[VOG07] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, G. Vigna, "Cross site scripting prevention with dynamic data tainting and static analysis," Network and Distributed System Security Symposium, 2007.

[VOL96] D. Volpano, G. Smith, C. Irvine, "A sound type system for secure flow analysis," Journal of Computer Security, vol. 4, no. 3, pp. 167-187, 1996.

[WAI07] J. Wainer, K. Kumar, P. Barthelmess, "DW-RBAC: A formal security model of delegation and revocation in workflow systems," Journal of Information Systems, vol. 32, no. 3, pp. 365-384, 2007.

[WAN04] Lingyu Wang, D. Wijesekera, S. Jajodia, "A logic-based framework for attribute based access control," ACM Workshop on Formal Methods on Security Engineering, 2004.

[WAN06] He Wang and S.L. Osborn, "Delegation in the role graph model," ACM Symposium on Access Control Models and Technologies, pp. 91-100, 2006.

[WAN07] Cheng Wang, Shiliang Hu, Ho-seop Kim, S.R. Nair, M. Breternitz, Zhiwei Ying, Youfeng Wu, "StarDBT: An Efficient Multi-platform Dynamic Binary Translation System," Lecture Notes in Computer Science, vol. 4697/2007, pp. 4-15, 2007.

[WEI81] M. Weiser, "Program slicing," IEEE International Conference on Software Engineering, 1981.

[WEL04] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist, "X.509 proxy certificates for dynamic delegation," 3rd Annual PKI R&D Workshop, 2004.

[WON04] R. Wonohoesodo and Z. Tari, "A role based access control for web services," IEEE International Conference on Services Computing, pp. 49-56, 2004.

[W3C04] W3C, "OWL-S: semantic markup for web services," http://www.w3.org/ Submission/OWL-S, 2004.

[W3C05] W3C, "Web services addressing," http://www.w3.org/Submission/ws-addressing/, 2005.

[W3C06] W3C, "Namespaces in XML 1.0 (second edition)," http://www.w3.org/TR/ REC-xml-names/, 2006.

[XU04] Feng Xu, Guoyan Lin, Hao Hao, Xie Li, "Role-based access control system for web services," International Conference on Computer and Information Technology, pp. 357-362, 2004.

[YEH11] Lo-Yao Yeh, Yen-Cheng Chen, Jiun-Long Huang, "ABACS: an attribute-based access control system for emergency services over vehicular ad hoc networks," IEEE Journal on Selected Areas in Communications, vol. 29, no. 3, 2011.

[YIL07] U. Yildiz and C. Godart, "Information flow control with decentralized service compositions," IEEE International Conference on Web Services, pp. 9-17, 2007.

[YOU05] C. Young, "Microsoft's rule engine scalability results - a comparison with jess and drools," http://geekswithblogs.net/cyoung/articles/54022.aspx, 2005.

[YUA05] E. Yuan and Jin Tong, "Attributed based access control (ABAC) for web services," IEEE International Conference on Web Services, 2005.

[ZHU06] Junqiang Zhu, Yu Zhou, Weiqin Tong, "Access Control on the Composition of Web Services," International Conference on Next Generation Web Services Practices, pp. 89-93, 2006.

VITA

Wei She was born in Shanghai, China. After completing high school in 1997, he entered Tsinghua University, Beijing, where he received a Bachelor of Science with a major in Computer Science in 2001 and a Master of Engineering with a major in Computer Engineering in 2004. During the following two years, he was employed as a software engineer in Computer Sciences Corporation (CSC). In 2006, he entered the Graduate School of the University of Texas at Dallas.