Towards A Security Management System For Composite Web Services

By
Haiyang Sun

A thesis submitted in fulfilment of the requirements for the Degree of Doctor of Philosophy

MACQUARIE UNIVERSITY
FACULTY OF SCIENCE

Department of Computing

August 2012

© Haiyang Sun, 2013.

Except where acknowledged in the customary manner, the material presented in this thesis is, to the best of my knowledge, original and has not been submitted in whole or part for a degree in any university.

Haiyang Sun

Dedication

to my parents, *Bingyi Sun* and *Wenrong Feng*,
and to my wife *Lanshi Wang*

Acknowledgements

This PhD thesis was made possible with the help of my dear colleagues from the Department of Computing at Macquarie University. First of all, I would like to thank my supervisor, Professor Jian Yang, for supervising my research work over the four years PhD study. Her critical questions and valuable advices enable me to make steady progress on my research and keep me on track. I also thank my associate supervisor, Dr. Weiliang Zhao, for his support and valuable guidance. His advices on the research methodology are important for me to achieve the research work. Special thanks Dr. Aries Tao Tao and Dr. Daisy Daiqin He. Discussions with them give me much inspiration on my research. Their supports and helps are greatly appreciated.

Also many thanks are given to the administration team in the Department of Computing at Macquarie University. Wonderful research environment they provide is essential for me to accomplish the thesis. I want to appreciate the Faculty of Science at Macquarie University as well. The financial supports they provided for attending the international academic conferences help me to exchange the ideas with other international researchers.

I also want to thank Dr. Blaine Xin Wang from Department of Computing at Victoria University, Dr. Huiyuan Zheng, Mr. Mahbub Hassan, and Dr. Trang Nguyen from Department of Computing at Macquarie University for reviewing my thesis and correcting typo errors.

Finally, I would like to thank my families, who support me all along with my research. The completion of the thesis would not have been possible without my father

and mother's encouragement, support and patience, and without my wife's perpetual love and understanding.

List of Publications

1. Haiyang Sun, Weiliang Zhao, Jian Yang, and Jianwen Su, "TiCoBTx-Net: A Model to Manage Temporal Consistency of Service Oriented Business Collaboration", **IEEE Transactions on Services Computing**, Volume 5, Number 2, pp.207–219,2012, IEEE Computer Society.

2. Haiyang Sun, Jian Yang, and Weiliang Zhao, "A Temporal Rule-based Verification System for Business Collaboration Reliability", **Journal of Theoretical and Applied Informatics**, Volume 16, Number 2, pp. 65-68, 2009.

3. Haiyang Sun, Jian Yang, and Lai Xu, "CoBTx-Net: A Model to Verify Reliability of Collaborative Business Transaction", **Journal of Information System Frontier**, Volume 11, Number 3, pp. 257-272, 2009, Springer.

4. Haiyang Sun, Weiliang Zhao, and Surya Nepal, "PASOAC-Net: A Petri-Net Model to Manage Authorization in Service-Based Business Process", in Proceedings of the 10th International Conference on Service Oriented Computing (**ICSOC 2012**), Shanghai, China, November 2012, Springer. (Accepted)

5. Haiyang Sun, Jian Yang, Weiliang Zhao, and Surya Nepal, "SOAC-Net: A Model to Manage Service-Based Business Process Authorization", in Proceedings of the International Conference on Service Computing (**IEEE SCC 2012**), Hawaii, USA, June 2012, IEEE Computer Society.

6. Haiyang Sun, Weiliang Zhao, Jian Yang, and Guizhi Shi, "SOAC Engine: A

System to Manage Composite Web Service Authorization", in Proceedings of the 12th International Conference on Web Information System Engineering,(**WISE 2011**), Lecture Notes in Computer Science 6997, pp.334-335, Sydney, Australia, October 2011, Springer.

7. Haiyang Sun, Weiliang Zhao, and Jian Yang, "Managing Conflict of Interest in Service Composition", in Proceedings of 18th International Conference on Cooperative Information Systems (**CoopIS 2010**), Lecture Notes in Computer Science 6426, pp.273-290, Crete, Greece, October 2010, Springer.

8. Haiyang Sun, Weiliang Zhao, and Jian Yang, "SOAC: A Conceptual Model to Manage Service Oriented Authorization", in Proceedings of the IEEE International Conference on Service Computing (**IEEE SCC 2010**), pp.546-553, Miami, USA, July 2010, IEEE Computer Society.

9. Haiyang Sun, Jian Yang, Xin Wang, and Yanchun Zhang, "A Verification Mechanism for Secured Message Processing in Business Collaboration", in Proceedings of the Joint International Conferences on Asia-Pacific Web Conference (**APWeb 2009**) and Web-Age Information Management (**WAIM 2009**), Lecture Notes in Computer Science 5446, pp.480-491, Suzhou, China, April 2009, Springer.

10. Haiyang Sun, Xin Wang, Jian Yang and Yanchun Zhang, "Authorization Policy Based Business Collaboration Reliability Verification", in Proceedings of The Sixth International Conference on Service-Oriented Computing (**ICSOC 2008**), Lecture Notes in Computer Science 5364, pp.579-584, Sydney, Australia, December 2008, Springer.

11. Haiyang Sun, and Jian Yang, "Enforcing Business Collaboration Consistency in Business Transaction Net", in Proceedings of the IEEE Joint Conference on E-Commerce Technology and Enterprise Computing, E-Commerce and E-Services (**IEEE CEC&EEE 2008**), pp.223-230, Washington DC, USA, July 2008, IEEE Computer Society.

12. Haiyang Sun, and Jian Yang, "Exploiting CoBTx-Net to Verify the Reliability of Collaborative Business Transactions", in Proceedings of the 2nd IEEE Asia-Pacific Conference on Service Computing (**IEEE APSCC 2007**), pp.415-422, Tsukuba Science City, Japan, December 2007, IEEE Computer Society.

13. Haiyang Sun, and Jian Yang, "CoBTx-Net: A Model for Reliability Verification of Collaborative Business Transaction", in Proceedings of the First International Workshop on Collaborative Business Processes (**WCBP 2007**)- in conjunction with the 5th International Conference on Business Process Management (**BPM 2007**), Lecture Notes in Computer Science 4928, pp.220-231, Brisbane, Australia, September 2007, Springer.

14. Haiyang Sun, and Jian Yang, "BTx-Net: A Token Based Dynamic Model for Supporting Consistent Collaborative Business Transactions", in Proceedings of the IEEE International Conference on Service Computing (**IEEE SCC 2007**), pp.490-497, Salt Lake city, USA, July 2007, IEEE Computer Society.

Abstract

A web service can be operated in a distributed environment with a large number of resources with evolving contents. These resources can be various types of applications that come from different organizations, e.g., component web services. A web service that composes multiple component services (resources) is called composite web service that can provide comprehensive and value-added function to service consumer. In order to acquire the support from these resources, the composite web service must be able to satisfy authorization policies of these resources. Interacting with service consumers is imperative for the execution of a composite web service. But, a service consumer can access the specific functions of a composite service, only after it can satisfy the authorization policies of the composite web service. Execution policies are used to manage the sequence of operation invocations within a composite web service, i.e., business logic. Therefore, without coordination management on these policies, a composite web service may not be able to perform properly, particularly in a process environment.

Currently, it is still lack of an effective approach to address the issue of security management in composite web service by considering both service consumers and component services, and taking various types of authorization constraints into account to manage and coordinate the service consumer access and component service support.

In this thesis, we propose a service oriented conceptual model (SOAC) as an extension of role based access control that can facilitate the administration and management of access for service consumers as well as component services supports in composite

web services. Various types of conflict of interest are identified due to the complicated relationships among service consumers and component services. A process model, SOAC-Net, is also developed based on our designed conceptual model SOAC. The SOAC-Net is a Petri-Net based process model that represents an authorization flow, i.e., the sequence of the accesses by service consumers and the sequence of the supports from components services. A set of authorization policies, e.g., *authorization synchronization policy* and *authorization dependency policy*, are designed based on SOAC-Net to coordinate the access and the support in a process environment. Verification on improper authorization policy definition is proposed based on SOAC-Net to detect the unreliable execution of composite web service. A service oriented authorization control engine (SOAC engine) is developed as an implementation of the proposed authorization framework of composite web services.

In summary, this research throws new light on the current state of the art in authorization management under the loosely-coupled composite web service environments.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Background

In the past few decades, changes in the economic environment, such as globalization, mass customization, and new competitive pressure, have forced organizations to search for innovations and gain competitive advantages. A key factor to maintain the competitiveness is the capability to cooperate with existing partners and potential customers in a standardized way, which can significantly enhance the business partner relationships [1]. However, the difficulty of seamless communications becomes an obstacle to impede achieving the goal of standardization, where the legacy systems of each organization can be developed by C#, JAVA, or VB.NET. The various types of communication methods in different applications cause the *seamless* impossible. Web Services technology is developed to break the barrier, where the internal legacy systems

at each organization can be encapsulated by an XML-based standardized web interface [2, 3]. Through the interface, the organization can collaborate with its business partners dynamically, freely and on-demand [4].

Now, web services have accounted for a major part of the IT industry [38]. Companies can increasingly focus on their core expertise areas and use IT services to address all their peripheral needs. Through web services technology, the companies can easily outsource their peripheral needs to other organizations which are more specialized in the area. Web service technology brings a totally new business interaction paradigm to change the business world [5].

When seamless communication barriers are broken by the service technology, the single web service at each organization can be composed together to provide comprehensive and value-added service functions to service consumers, when single web service is not enough to catering for the various types of customer requirements.

1.1.1 Service Composition

The service technologies have an aim to allow effective composition of discrete services into end-to-end service aggregation [6]. The nature of web services creates the opportunity for building composite services by combining existing elementary or complex services (referred to as component services or resources of composite web service) [51]. A composite web service is operated in a distributed environment involving multiple parties with dynamic availability, and a large number of these resources/component services with evolving contents. Through the creation of alliances between the services, the composite web services are built up to provide comprehensive service functions to service consumers. When the services are composed together, the following features can be identified,

- **Loose coupling**: Since web service technology can provide seamless communications when services are composed together, their relationships are loosely coupled during interaction. Loose-coupling describes a type of distributed application components composition where each component is independent and composed

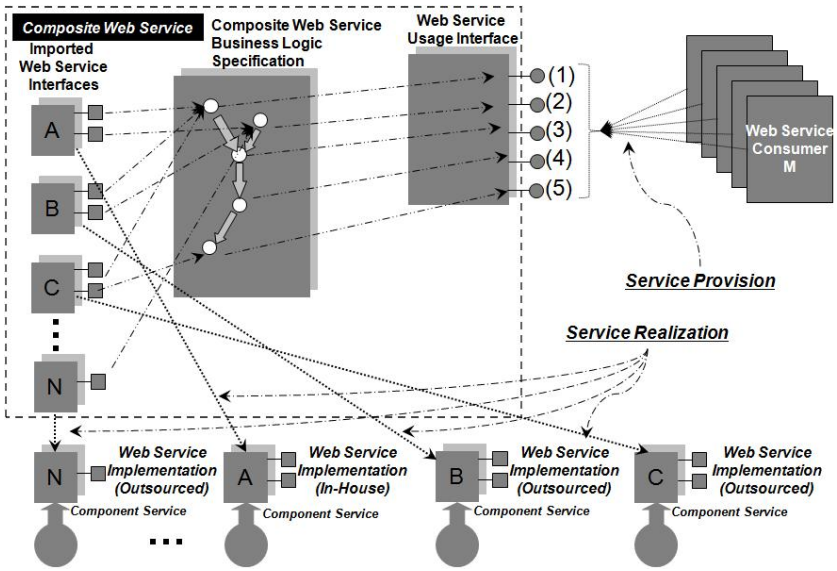

Figure 1.1: Architecture of Composite Web Service

together in multiple ways without detailed specification on how to collaborate in advance. It establishes the components collaboration in a highly dynamic fashion and on-demand basis.

- **Autonomy**: Each web service is autonomous. It does not disclose the information of the internal legacy system that it encapsulates, nor does it expose any of its internal business rules and policies used to manage its deployment. Each web service is well self-contained.

- **Business Logic**: The web services interact according to specific business logic. When web services are composed together to provide comprehensive function to service consumer, the service interactions between service consumers and the composite web service, or the interactions between the composite service and its composing component web services all need to comply with a well-defined execution sequence, i.e. a business logic.

In Fig. 1.1, we introduce an Architecture of Composite Web Service, in which various component services are composed together to provide support to the composite

web service, and the functions of the composite web service are used by multiple service consumers. In Fig. 1.1, the *composite web service* is categorized into three parts, (1) `web service usage interface`, (2) `composite web service business logic`, and (3) `imported web service interface`. In `web service usage interface`, operations are attached with the interface and are illustrated as grey circles, e.g. 5 operations can be identified in Fig. 1.1. In web service environment, the functions of the service are realized by different operations. Through these operations, *web service consumers* can access the function of the *composite web service*. `Composite web service business logic` is used to control the execution sequence of these operations. It is also a guide to a service consumer on how to use and access the operations of the composite web service. `Imported web service interface` is from the *component web services*, that support the *composite web service* to provide comprehensive functions to *service consumers*. `Imported web service interface` is the projection of component web service interface.

In Fig. 1.1, one component service can provide support to multiple operations of the composite web service. For example, component service B can provide support on both operations 1 and 2 in a composite web service. On the other hand, one operation of a composite web service can be supported by various component services. For instance, operation 1 can be supported by all component services A, B and C. The same rules will also be implemented between composite service and service consumer, where one operation within the composite web service can be accessed by multiple service consumers, and one service consumer can access multiple operations of the composite web service.

The business interactions between a service consumer and a composite web service are called **Service Provision**; while **Service Realization** is identified for the business interactions between a composite web service and component services.

Based on Fig. 1.1, the features of a service composition can be further described as:

1. **Loosely Coupled**: The service consumer and the component service can not be identified in advance. All interactions between a service consumer and a composite web service at service provision part, and all interactions between a composite

web service and a component service at service realization part are based on demand and in a dynamical fashion. For example, a component service may just be able to provide support once for a composite web service, and then becomes unavailable. The composite web service therefore will seek another component service to replace the previous one. Their relationship is not bound together.

2. **Autonomy**: Each web service, as service consumer, composite web service, or component service, bears own internal policies and rules to govern their own executions, and can not be exposed to others. For example, it is impossible for a composite web service to perceive how its component services provide the support, nor can a service consumer find how the operations of a composite web service are achieved. All types of web services are **autonomous**.

3. **Business Logic**: The `composite web service business logic` part in Fig. 1.1 presents that, both service consumers and component services need to comply with the execution sequence of a composite web service. They must follow the **business logic** to execute.

All features identified above in service composition make the security management in a composite web service *unpredictable*, *complex*, and *difficult*.

1.1.2 Service Authorization

Security is an eternal issue in web environment [99]. To protect the privacy and confidential information, security management in web environment mainly focuses on **access control**, i.e. authorization. It will restrict who can touch the sensitive information and guarantee that only the qualified entity can access the necessary data, information or knowledge, at the right time and right place [29–31].

Recently, there are three types of authorization control methods that are popularly used in industry and attract most researches in academic domain. They are Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role-based Access Control (RBAC), that are presented as follows,

- **DAC**: In DAC, users as the subject are granted permissions to access the objects based on its entity or its belonging group. User bears the full control on its granted permissions, i.e. the user can propagate its granted permissions to other users.

- **MAC**: In MAC, a central controller as an administrator deploys authorization rules that are enforced to decide if the subject can access the object. All permissions granted to the subject to work on the object are managed by the authorization rules. Since the authorization rules are centrally managed by the administrator, no user as the subject can propagate the granted permissions to others. In MAC, each subject is granted only necessary permissions.

- **RBAC**: In Role Based Access Control (RBAC) [7–9, 52], users acquire permissions through their roles rather than that they are assigned permissions directly. This greatly reduces the administrative overhead associated with individual users and permissions. For example, when a user bears 10 permissions on 50 objects (the permissions can be *Read*, *Write*, or *Update*, etc., and the objects can be databases, tables or specific business applications), and the user is leaving the company, how many steps are needed for a security administrator to handle the situation? Without RBAC, 10×50=500 steps! The administrator has to find each of the 50 objects, identify the permissions, and finally erase the 500 links on the users. However, in RBAC, a role is linked to the 10 permissions on the 50 objects. The administrator only needs 1 step to erase the links between the user and the role. Hence, we can conclude, that by adopting RBAC, the authorization management becomes efficient.

In web service paradigm, operations are used to represent the functions of each web service. Hence, the information of the operations, e.g. name and communication methods, and the information on how the operations work, e.g. the operation execution time, sequence, and the content of the message that the operation can process, become the confidential information to public [32, 33]. Not every service consumer can access the operations of a composite web service. From the component service point

of view, also not every composite web service can access the component service's operations. Each operation of the web service should be secured and protected to avoid any malicious access or misuse [34].

Hence, a successful execution of a composite web service is possible only if the following two types of authorization policies are satisfied: (a) authorization policies of component services that a composite web service must comply with in its execution; and (b) authorization policies of a composite web service that service consumers must follow to acquire the operations of the composite service. Furthermore, execution policies need to be applied to realize business logics of a composite web service through a sequence of task invocations among service consumers, component services, and operations of a composite web service. Therefore, without a proper coordination among these policies, a composite web service may not be able to execute successfully, e.g., imperative support from a specific component service could be missing, unauthorized service consumer access can occur during the execution of a composite web service, or the authorization can cause conflict of interest for a composite web service. Hence, there is a need of an effective authorization model that brings all types of policies together for a composite web service executing successfully without breaking any authorization and business rules. However, how to manage the composite web service authorization in such environment becomes an issue that has not been tackled yet. The next section will present what research problems should be solved in order to manage the composite web service authorization.

1.2 Research Problem

1.2.1 Motivating Scenario

Before elaborating the research problems in terms of composite web service authorization, let us take a motivating scenario firstly. Financial Lease is a composite web service that can provide the business equipment finance and leasing solutions. When Financial Lease receives a lease application from a customer, it firstly assesses the value

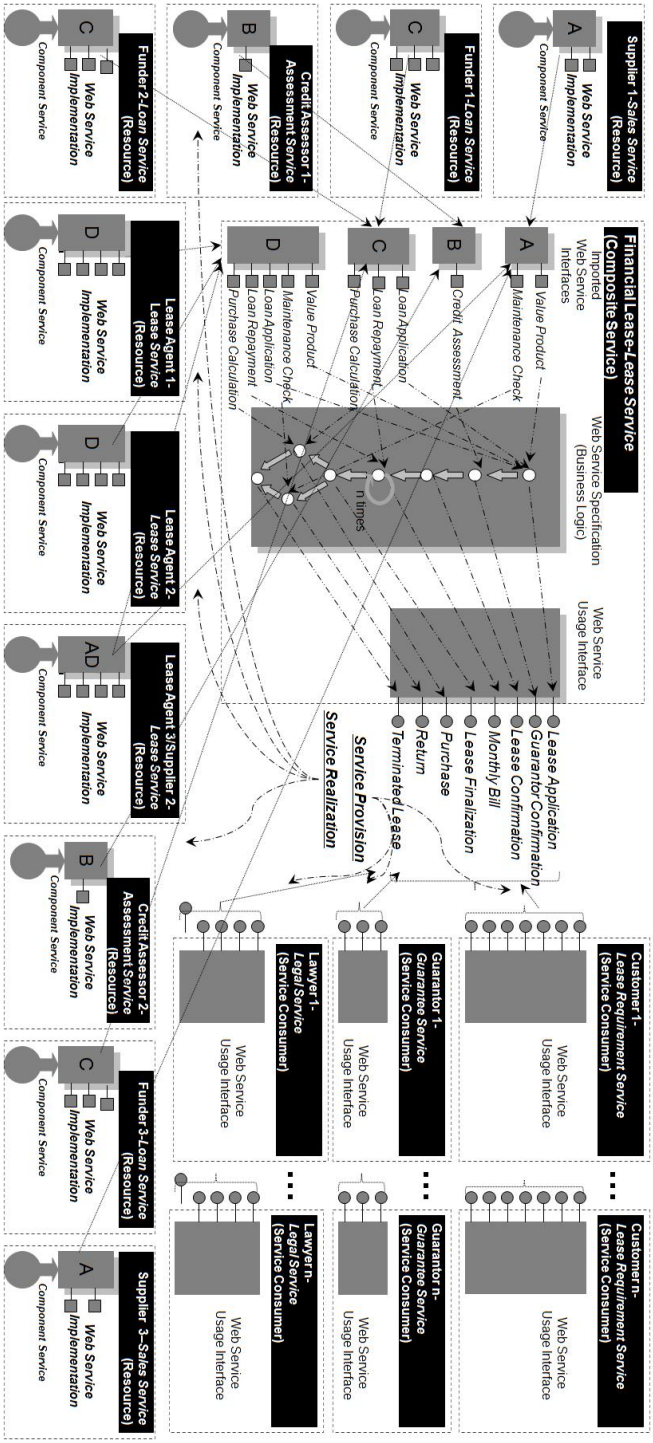

FIGURE 1.2: Motivating Scenario for Financial Lease

of the product that the customer wants to lease from a product supplier and seeks a specific funder who can provide financial support. A guarantor for the customer must be confirmed to lower the risk of bad debt. The lease application will be confirmed to customer and his/her lawyer, once the credit history check of the customer and guarantor is finished. The rental on the leased product will be paid regularly. After the agreed period, the customer or his/her lawyer can advice the Financial Lease a lease finalization notice at any time, that will enable Financial Lease to start the internal finalization process. The customer has two after-lease choices, (1) purchasing the product with a small-amount cost, or (2) returning the product. Regarding to the purchase, funder will be involved to calculate the cost to be paid by the customer; while for returning the product, supplier will carry out the maintenance check on the product. Once the product has been purchased or returned, the whole lease process is terminated.

Based on the example (See Fig. 1.2), we can observe that three types of service consumers, `Customer`, `Guarantor`, and `Lawyer`, and four types of component services, `Supplier (A)`, `Credit Assessor (B)`, `Funder (C)`, and `Lease Agent (D)`, are involved in the execution of Financial Lease. Each type of service consumer and each type of component service can include a large number of web services. For example, in Fig. 1.2, there are N web services that can be used to represent `Customer`. Moveover, in Fig. 1.2, there are at least three web services that can be used to represent the `Supplier`. On the other hand, a web service can play as multiple types of service consumers and component services. For example, in Fig. 1.2, a web service can play as both `Supplier` and `Lease Agent`.

In Fig. 1.2, the Financial Lease service is categorized into three parts as introduced in Fig. 1.1 which illustrates the Architecture of Composite Web Service at subsection 1.1.1. The three parts are (1) `web service usage interface`, (2) `composite web service business logic`, and (3) `imported web service interface`.

In `web service usage interface` of Financial Lease, there are 8 operations identified and illustrated as grey circles. They are as follows,

1. **Lease Application** is used to receive the lease application from approved web

service (as a service consumer). Note, not every web service can obtain the right to submit lease application.

2. **Guarantor Confirmation** is used to receive the information of guarantor.

3. **Lease Confirmation** is an operation used to send the result of lease application after the web service that submits lease application sends a result enquiry.

4. **Monthly Bill** is an operation accessed by the web services that are qualified to repay the rental.

5. **Lease Finalization** is used to receive the lease finalization notice.

6. **Purchase** is used to receive the purchase notification and payment information.

7. **Return** is an operation accessed by the web service that needs to return the rented product.

8. **Lease Termination** is an operation used to close the lease case.

The functions of Financial Lease service are realized by the 8 operations in `web service usage interface`. `Composite web service business logic` is used to control the execution sequence of these 8 operations. In the `composite web service business logic` of Financial Lease service at Fig. 1.2, the operations of **Lease Application**, **Guarantor Confirmation**, **Lease Confirmation** and **Monthly Bill** are required to execute sequently; while **Monthly Bill** needs to be executed repeatedly for repaying rental at each month of the leasing period. After that, **Lease Finalization** should be performed, followed by an exclusive choice between the operations of **Purchase** and **Return**. The lease finishes when the choice is determined and the operation of **Lease Termination** is reached. The `composite web service business logic` of Financial Lease service at Fig. 1.2 is also a guide to service consumer and component service on how to access and support the operations of the Financial Lease.

`Imported web service interface` in Fig. 1.2 comes from the different types of

component web services, that support the composite web service to provide comprehensive functions to service consumer. `Imported web service interface` is the projection of component web service interface. In Fig. 1.2, the different types of `imported web service interfaces` are as follows,

- `Supplier` as a type of component service is illustrated as **A** in `imported web service interface` in Fig. 1.2 to provide support on the operations of *Lease Application* and *Return*. Three web services in Fig. 1.2 can be used to represent as `Supplier`.

- `Credit Assessor` is a type of component service that needs to support the operation of *Guarantor Confirmation* as **B** in `imported web service interface` of Financial Lease service in Fig. 1.2.

- `Funder` is a type of component service that can provide support on the operations of *Lease Application*, *Monthly Bill*, and *Purchase*, as **C** in `imported web service interface` of Financial Lease service in Fig. 1.2.

- `Lease Agent` is a type of component service that can provide support on the operations of *Lease Application*, *Monthly Bill*, *Return* and *Purchase*, as **D** in `imported web service interface` of Financial Lease service in Fig. 1.2.

Until now, (1) from `web service usage interface`, we know that there are 8 operations of Financial Lease that need to be accessed by three types of service consumers. (2) From `composite web service business logic`, we know that the 8 operations of Financial Lease service are executed based on a well-defined business logic. (3) From `imported web service interface`, we know that the identified 8 operations of Financial Lease service require four types of component services to provide support. Here, how the three types of service consumers and four types of component services are able to access and support the 8 operations of composite web service is described below in details,

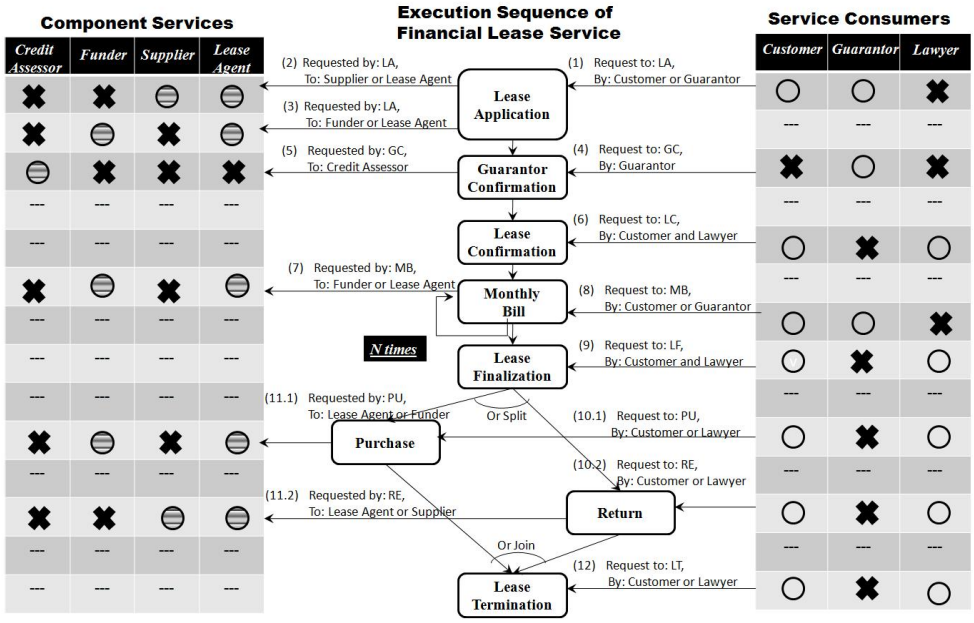

FIGURE 1.3: Execution Sequence of Financial Lease

- **Service Consumer**: (1) `Customer` should be able to access all operations. (2) *Guarantor Confirmation* can only be made by `Guarantor`. `Guarantor` can start the lease on behalf of the `customer` and help repay the rental. (3) `Lawyer` after lease finalization stage can, on behalf of its client, deal with any rest activities. Furthermore, `Lawyer` is necessary to involve in operations of *Lease Confirmation* and *Lease Finalization* with `Customer`.

- **Component Service**: (1) `Funder` is used to provide financial support. (2) `Supplier` is the product provider. (3) `Lease Agent` can provide both product and fund to Financial Lease, since the `Lease Agent` will seek its own `Funder` and `Supplier`. However, the return rate for `Funder` and for `Lease Agent` is different since `Lease Agent` will always be used to take high risk application. (4) `Credit Assessor` is used to evaluate the credit history of `Guarantor` and `Customer` as the third party.

In Fig. 1.3, we illustrate an execution sequence of Financial Lease service, where each operation of Financial Lease service is depicted as a rectangle in the process.

The three types of service consumers (`Customer, Guarantor, and Lawyer`) and the four types of component services (`Funder, Supplier, Lease Agent, and Credit Assessor`) are identified to access and support these sequenced operations. A *Service Consumer* table is used to illustrate three types of service consumers, and their associated access policies on operations. For a specific service consumer, a white circle indicates that access on specific operation is granted to the service consumer and black cross means that access is denied. Four types of component services, `Funder`, `Supplier`, `Lease Agent`, and `Credit Assessor`, are illustrated in *Component Service* table in Fig. 1.3. The circles with horizon stripes represent that this type of component service can support the specific operation; while black crosses share the same semantics in *Service Consumer* table. The sequence of interactions among service consumers, component services, and operations of Financial Lease are numbered in Fig. 1.3. We only describe the *Request* interactions in the figure, while the corresponding *Reply* interactions are not depicted to keep the diagram concisely.

We can observe from the above example that the service consumer accesses and the component service supports are not only regulated by their specific authorization policies, but also need to be restricted by the business constraints enacted during the execution of a composite web service. Otherwise, authorization issues can be raised to cease the execution of a composite web service. These business constraints can be categorized as follows,

1. Each type of service consumer and component service can only be used to access and support the operations of the Financial Lease when they are needed by the execution of an operation.

2. The `Guarantor` can repay the rental on behalf of the `Customer`, only if the lease application is also submitted by the `Guarantor`.

3. The `Lease Agent` can issue the monthly bill, only if it is the financial provider.

4. The `Lease Agent` can be used to deal with the purchase of a product, if the lease application is processed by a `Lease Agent`.

5. The `Lease Agent` can be used to deal with the return of a product, if the lease application is processed by a `Lease Agent`.

6. The `Credit Assessor` can support for assessing the credit history check, only after the guarantor information can be confirmed.

7. Operation *Monthly Bill* can be supported by `Funder` or `Lease Agent` and accessed by `Customer` or `Guarantor`. To avoid fraudulent activity, a `Guarantor` can pay the rental on behalf of a `Customer` to access the operation *Monthly Bill* only if a `Funder` issues the bill. This constraint is used when an entity can play as both `Lease Agent` and `Guarantor`. Obviously, if the entity pays the bill issued by itself, it may eventually do harm to a `Customer`'s interest.

8. If a `Lawyer` and a `Lease Agent` are together to deal with the return of a product, then the lease application must be submitted directly by a `Customer` and the product should be provided by the `Lease Agent`. The application submitted by an applicant in person will lower the risk of dispute on the product of return which is dealt with by a `Lease Agent` and a `Lawyer`, since the `Customer` can involve enough during the lease period.

The authorization of composite web services is different from traditional authorization in a close system due to the dynamic and complex relationships among service consumers and component services. The authorization of composite web service should also be managed in a process environment. In the following sections, we will present the research problems that ought to be dealt with to manage composite web service authorizations, based on the above motivating scenario.

1.2.2 Complicated Coordination of Authorization Policies

The component services are normally remote resources (the term **resource** will be used interchangeably with **component service** in the thesis), that can provide support to the operations of a composite web service. They can belong to different security

domains and have their own access policies. Granting the permissions to a service consumer on accessing the operations of a composite web service should not only confirm whether the service consumer is qualified, but also need to take access policy of the components service into account, i.e. the availability of the support from component services should be guaranteed.

However, the characteristics of these resources are different from that of objects of Role-based Access Control in close systems. In traditional authorization system, the object that needs to be accessed belongs to this authorization system. The availability of the object can be completely identified and managed by the authorization system. For example, in database domain, the object that needs to be accessed can be a table within a database, and it can be totally controlled by the authorization system in the database.

In web service environment, a composite web service needs to access the operations of component services, i.e. resources, to acquire their functional support. The component services can belong to different organizations, come from different security domains, and have different security and interest requirements. Hence, the authorization system of the composite web service can not control the object to be accessed, since the object, i.e. the operation of these resources, belongs to the component service and is managed under the scope of authorization system in a component service, not a composite web service. The authorization system of the composite web service can only identify the public resources' authorization requirements, e.g., what credential is needed to obtain the right to access the operation of a specific component service. It means that, from composite web service point of view, only the projection of a component service, i.e., the public authorization requirements, can be identified. To confirm the availability of a resource's support, a composite web service has to examine if the credential required by its component services can be prepared. This changes the confirmation on the availability of the object from a static and stable environment (within a close system, the availability of the object is seldom changed frequently), into a dynamic and instable environment (within a web service environment, the resource can change frequently).

Therefore, the authorization system of a composite web service should not only manage the access of service consumers, but also need to create a mechanism to manage the support from component services. Obviously, without the support from a specific component service, the operation that needs the support should not be accessed by the service consumer. Coordination on the authorization policies of a composite web service and those of component services is required. The authorization policies of a composite web service are used to decide if the specific service consumer is qualified for accessing specific operations of the composite web service; while the authorization policies of component web services are identified by the composite web service to decide if the support from the component services can be acquired. The complicated coordination on these two types of authorization policies changes the basic authorization question "`who can do what`?" to a more complicated one as "`who can do what under what kind of support`".

Let us take an example from the motivating scenario. The operation of **Lease Application** needs to be accessed by `Customer` or `Guarantor`, and needs to be supported by `Funder`, `Supplier`, and `Lease Agent`. Hence, when deciding if a web service can access the operation of **Lease Application** as a `Customer` or a `Guarantor`, the Financial Lease service should confirm if the one or several web services can be used to provide support on the operation of **Lease Application** as `Funder`, `Supplier` or `Lease Agent`, i.e. if the public authorization requirements of these web services can be satisfied by Financial Lease service.

Unfortunately, the existing role-based models in web service paradigm have not brought the administration of component services into the picture.

1.2.3 Dynamicity of Component Services and Service Consumer

Web services technologies facilitate the integration of the loosely-coupled distributed applications. Hence, a large amount of component web services can be used as resources to support a composite web service's operations; while the quantity of service consumers

can also be large. As illustrated in Fig. 1.2, the system of the composite web service needs to deal with not only a large number of users (service consumers) but also great amounts of resources (component services).

Moreover, the resources and service consumers are normally prone-to-change. Since component services and service consumers are both out of control of a composite web service, they can change their availability or authorization requirements at any time according to their own interests, without advanced notification to the composite web service (the relationships among different web services are loosely-coupled.). For example, if a web service as `Funder` changes its authorization policy from asking for occupational loan certificate to requiring the leasing qualification, then all the operations of financial lease service that can be supported by the service as `Funder` need to be involved with updating this authorization policy change accordingly.

However, if the changes occur frequently or happen in many web services, efficient authorization management is needed to reduce the administrative overhead in authorization of composite web services.

1.2.4 Conflict of Interest

A conflict of interest occurs when an individual is involved in multiple interests, one of which could possibly corrupt the motivation for an act in the other. Conflict of Interest is important and yet challenging in managing authorization of a composite web service, especially when the backend resources (component services) are taken into the picture. The existing role based access control approaches use the mechanism of *Separation of Duty* to deal with conflict of interest focusing on service consumers only. However, with the involvement of component services, the conflict of interest in terms of composite web service authorization can not only occur for service consumer or resource only, it can also happen between service consumers and resources, and even between pairs of service consumers and resources. Let us take several examples below,

- **Conflict of Interest by Service Consumer/s**: The Financial Lease service needs service consumers to play as `Customer`, i.e. the entity which needs the

lease, and `Guarantor`, i.e. the entity which can guarantee the Customer's lease, to access its operations. If the same service or the services with shared interest play as both `Customer` and `Guarantor`, the conflict of interest can occur between the service consumer and Financial Lease service. The interest of Financial Lease service is to use `Guarantor` to lower the risk of bad debt of `Customer` on the lease. The service consumer plays as both `Customer` and `Guarantor` can cause that the risk of bad debt of Customer' lease increases as the service consumer is guaranteing the lease application of itself. The influence of `Guarantor` on the lease is totaly missing, resulting in the damage of Financial Lease's interest.

- **Conflict of Interest by Resource/s**: Various types of resources are needed to support the operations of Financial Lease service, e.g. `Credit Assessor` and `Funder`. If the same service or the services with shared interest play as both `Funder` and `Credit Assessor`, the conflict of interest can occur between the Financial Lease service and the resources. The interest of Financial Lease service is to use the correct credit history report to lower the risk of bad debt of `Customer`'s lease. The service that plays as both `Funder` and `Credit Assessor` can modify the credit history report of `Customer` and `Guarantor` to reach the credit limit of Financial Lease service. Their goals aim to release the loan as much as possible to Financial Lease service. (Of course, the true credit history report can satisfy the credit limit of `Funder`, but not the credit limit of Financial Lease service). Hence, the Financial Lease will falsely estimate the risk of bad debt of `Customer` that can cause the interest damage.

- **Conflict of Interest by Service Consumer and Resource**: A web service or services with shared interest can play as both service consumer that requires access on the operations and resource that is needed to support the operations. However, conflict of interest can occur between Financial Lease service and the web service as both service consumer and resource. For example, when a web service plays as both `Credit Assessor` and `Customer`, the conflict of interest occurs between Financial Lease service and the web service. The interest of

Financial Lease service is to judge the lease application under a fair and correct evaluation of the credit history of the `Customer`. The web service as both `Credit Accessor` and `Customer` can modify the credit history report of itself to catering for the credit requirements of Financial Lease, which makes it generate a wrong decision on the lease release. The interest of Financial Lease service is broken by the web service as both `Credit Assessor` and `Customer`.

- **Conflict of Interest by Group of Service Consumer and Resource**: The service consumer and the resource that can access and support on the same operation can be grouped together. The group of web services that play as different groups of service consumer and resource should not cause any conflict of interest in composite web service authorization. For example, when a service consumer plays as a `Military Customer`, and the goods it requests to rent need to be supplied by part manufactory as a `Vehicle Engine Supplier`, it will violate the law if Financial Lease service also uses the same manufactory that plays as `Vehicle Engine Accessory Supplier` to supply the engine accessory to the same service consumer as a `Commercial Customer`. In this case, the group of `Military Customer` and `Vehicle Engine Supplier` and the group of `Commercial Customer` and `Vehicle Engine Accessory Supplier` are mutually exclusive. They can not be assigned simultaneously to the same group of service consumer and resource, to avoid the conflict of interest. This conflict of interest is identified to prevent the following two things happening at the same time. The first thing is to assemble the engine for military use with the engine accessory for civil use and the second thing is to rent engine and engine accessory from the same part suppliers who can juggle with the different types of products it provides. In a summary, if the manufactory as `Vehicle Engine Supplier` to provide engine to a service consumer as `Military Customer`, it should not provide engine accessory to the same service consumer that is identified as `Commercial Customer`; vise versa.

Therefore, it is necessary to specify authorization policies to detect and control the various types of conflict of interest in a composite web service. Authorization in composite services must prevent conflict of interest by service consumer, by resource, and even by service consumer and component service, and pair of them.

1.2.5 Compliance of Business Logic

The value-added function can be provided by composite service to service consumers once it composes multiple component services [101]. However, obtaining permissions is imperative for the service consumers before accessing the specific function of the composite web service; while satisfying authorization constraints of component services is the key factor, for which the composite web service can acquire their support. Furthermore, each function of a composite web service can be accessed and supported by multiple service consumers and component services. To achieve the composite service functions, a certain business logic is inherently set up within the composite service, which therefore requires these accesses of service consumers and supports of component services to comply with.

Based on the motivating scenario, we can observe that, (1) an operation of a composite service can be accessed and supported by multiple types of service consumers and component services, and (2) the inherently business logic set up within Financial Lease service requires the compliance of the access of service consumer and the supports of component services. A question that should be significantly marked is "how the service consumer and the component service can pale others to be selected to work on the specific operations during the execution process of Financial Lease service". The failure of coordinating on the large number of accesses of service consumers and supports of component services in the business process can cause various security issues.

For example, when the `Guarantor` finishes accessing the operation *Guarantor Confirmation*, its permission to access this operation should be revoked to avoid duplicated submissions of guarantor information, which may cause cheating. Let us take another example. In Fig. 1.3, *Monthly Bill* operation can be supported by `Funder` or `Lease Agent` and accessed by `Customer` or `Guarantor`. A `Guarantor` can pay the rental on

behalf of the `Customer` to access the *Monthly Bill* operation only if the bill is issued by a `Funder`, rather than a `Lease Agent`. This constraint can be used to avoid fraudulent activity, where both the `Lease agent` and the `Guarantor` are not the really lender and borrower, and cheating can be made by their privately negotiation.

Therefore, we can conclude that, the accesses of the service consumers and the supports of the component services are not only regulated by specific authorization constraints to guide what the service consumer can access and what the component service can support; but also restricted by the business authorization constraints enacted during the execution process of a composite service to maintain the security (See the 11 business authorization constraints listed in section 1.2.1). These business authorization constraints can be categorized as follows,

- **Synchronization**: The sequence of the accesses and supports by service consumers and component services needs to synchronize with the execution sequence of the operations. The service consumer and the component service should not be able to access and support the operations that have been executed in the business process, nor can they access and support the operations that will occur in the future. They can only access and support the operation when the operation is executing in the business process, and the permission to access and support on the specific operation should be revoked immediately when the operation is finished.

- **Dependency**: The access and support on an operation may depend on the other access or support on specific operation. The execution of depended access or support is a necessary precondition of the execution of the depending access or support. The security of a composite web service can be maintained if the depended access or support can be performed before the depending one.

In a summary, the accesses of three service consumers and the supports of four component services in Fig. 1.3 must be synchronously sequenced with the business logic of Financial Lease service. Within the access sequence and the support sequence, each access and support must also satisfy the dependency requirements to cater for business

security demands. Hence, an effective and efficient authorization management must be proposed to coordinate these accesses and supports during the execution of the composite web service, based on these identified authorization constraints. Unfortunately, this type of management is still missing in the domain of web service authorization.

1.2.6 Authorization Policy Verification

The access of service consumer and the support of resource are necessary key factors, for which the operation of the composite web service can be executed with its business logic. It means that, the operation within a business process can not be successfully executed if the needed access from service consumer and/or the necessary support from component service are missing. Hence, the execution of a composite web service can become unreliable, where for specific operations within the execution sequence of composite web service, the service consumers that can access the operations and the resources that are needed to provide supports on the operations are missing or can not perform their designated functions due to the *improper authorization policy definition*. The whole execution of the composite web service can even be suspended and stepped into a dead state. Verification on the authorization policy to guarantee the normal execution of the composite web service becomes a must.

For example, the operation of *Lease Confirmation* requires that at least one component service is available as a `Credit Assessor` on evaluating the credit history of `Customer` and `Guarantor`. However, within the authorization system of Financial Lease service, if no component service is defined to be able to work as a `Credit Assessor`, the whole execution of Financial Lease is suspended due to the missing of `Credit Assessor`, i.e., an improper authorization policy definition.

1.2.7 Limitation of Existing Work

Based on the research problems identified above, we can conclude that it is necessary to have a well-defined framework to manage the authorization of composite web services, which could (1) take the authorization constraints of component services (resources)

into consideration, (2) provide efficient way to manage the large quantity of service consumers and resources, (3) have the capability to identify and control a series of types of conflict of interest in the authorization of composite web services at both design time and runtime, (4) provide mechanism to manage the composite web service authorization in a process environment, and (5) ensure that the authorization policy is defined properly to facilitate the execution of a composite web service based on the well-defined business logic.

We can find that, the MAC, DAC, and traditional RBAC are all not suitable for the service-oriented authorization management. MAC and DAC are not suitable for the composite web service authorization due to their limitation on the efficiency. Without involving the mechanism of role, the administrative overhead on these two access control methods is very high. On the other hand, role-based access control (RBAC) is not suitable for composite web service authorization since it has not taken *Resource* into account. Note the *Resource* particulary can not be fully controlled in the context of service environment. Researches in [60, 61, 70] are the representative work where role-based models are implemented in web service paradigm. In these work, roles are assigned to service consumers for service authorization. However, they either do not put resource into the picture, or they simply employ an unrealistic assumption that there is a *global* coordination on the internal authorization policies of each autonomous web services (service consumer, composite web service and component service) to enforce the access control in service composition. As we introduced in previous section, each type of web service is autonomous and their relationships are loosely coupled. Global coordination on the internal authorization policies of each type of web service can not be implemented.

Other representative works are mainly focusing on managing authorization in business process. Researches in [74, 75] mainly focus on managing synchronization between the execution sequence of the operations and the access sequence of the service consumer. The support sequence of the component service is totally ignored in process-based composite service management. Furthermore, much representative research work, e.g., [76], design constraints for avoiding conflict of interest within the

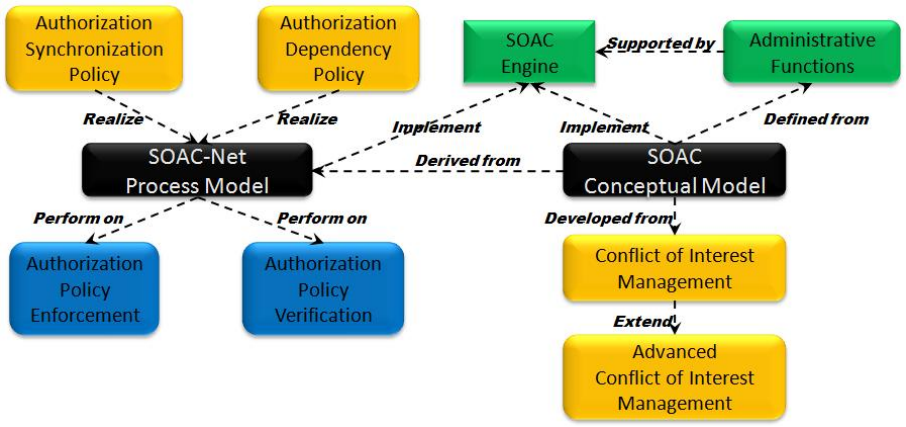

FIGURE 1.4: Research Methodology

accesses of service consumers. Dependency constraints are still missing, particularly, when they become more and more complex by involving the support of component services. Verification on these authorization policies in business process is completely not touched.

In a summary, existing work can not solve all the research problems identified in previous sections for managing composite web service authorization. A well-defined framework to manage the authorization of composite web services should be developed, that can comprehensively coordinate the access of service consumers and the support of component services. In next section, we will introduce the research methodology adopted by this thesis.

1.3 Research Methodology

We need to design a framework to manage composite web service authorization and tackle all issues identified in previous section. We adopt a research methodology by initially designing a conceptual model, named service oriented authorization control (SOAC). The reason to design conceptual model is that, (1) we need to clearly identify which elements in a composite web service are involved in authorization management, (2) clarify their relationships, and (3) examine if new elements are needed; if yes, then

new elements which can contribute to the authorization management of the composite web service are designed in the SOAC conceptual model. Based on the designed SOAC conceptual model, the research problems of "**Complicated Coordination of Authorization Constraints**" and "**Dynamicity of Component Services and Service Consumers**" are solved.

After designing the conceptual model, analysis work should be carried out based on the model to evaluate how the conceptual model can facilitate the management of composite web service authorization. As shown in Fig. 1.4, conflict of interest management can be enforced based on the SOAC conceptual model. By analyzing the relationships of different elements with the same type in SOAC, various types of conflict of interest can be identified and avoided at both design time and runtime. Moreover, conflict of interest management can also be extended by analyzing the relationships of different authorizations for one element only. By enforcing the different types of authorization policies, the issue of "**conflict of interest**" is tackled.

However, only designing a conceptual model can not explore the issue regarding to the "Compliance of Business Logic". In SOAC conceptual model, we can only answer the question of "`who can do what under what kind of support`". The question on top of "what to do" still can not be answered, i.e. "how to do". In order to answer this question, we have to derive a process model from the conceptual model to overcome the above limitation. Therefore, as shown in Fig. 1.4, a process model named SOAC-Net is developed based on the Petri-Net model. In terms of details of Petri-Net, we will introduce in Chapter 2. By introducing the process model, how to manage the composite web service authorization under an environment of business process can be solved. Authorization policies can be designed based on the process model to enforce the composite web service authorization to comply with the business logic of the composite web service. Furthermore, Petri-Net based process models provide many formal verification mechanisms to guarantee the execution of the process model. The research problems "**Compliance of Business Logic**" and "**Authorization Policies Verification**" then can be solved.

In a summary, our research methodology adopts the design ideas, (1) designing conceptual model, (2) constructing process model, and (3) based on the conceptual model and the process model, authorization policies are defined to avoid conflict of interest and to enforce and verify the authorization under the business process environment. In Fig. 1.4, we use black rectangles to represent the core models, *SOAC conceptual model* and *SOAC-Net process model*. The SOAC-Net process model is derived from the conceptual model. We use yellow rectangles to represent authorization policies. In Fig. 1.4, authorization policies are defined based on the conceptual model to avoid conflict of interest. In terms of process model, authorization policies are designed to enforce *authorization synchronization policies* and *authorization dependency policies*. These two types of authorization policies working on the process model are used to manage that the composite web service authorization can be executed properly under the business process environment. Based on the process model, authorization policies enforcement is used to examine if the authorization policies are fulfilled during the execution of a composite web service. Authorization policies verification is used to detect if the authorization policies are defined correctly and properly. We use blue rectangles to describe the two methods, *Enforcement* and *Verification*, in Fig. 1.4, which are linked to the process model. Finally, based on the conceptual model, administrative functions are developed to manage the elements and their associated relationships. All the administrative functions will be used to design the SOAC-Engine, an system that can implement all types of authorization policies based on the conceptual model and process model, to manage the composite web service authorization. We use the green rectangles in Fig. 1.4 to illustrate the two parts that are related with the system implementation.

1.4 Contribution of the Thesis

In this thesis, in order to solve the problems on managing composite web service authorization, we design the Service Oriented Authorization Control (SOAC) conceptual

model. In SOAC model, all elements involving in the composite web service authorization are identified. The relationships between the elements are illustrated. However, SOAC model is still not strong enough to solve the problems on managing composite web service under the environment of business process, since the business logic of a composite web service is not considered in SOAC. Hence, we also design a process model named as SOCA-Net, that is developed based on the Petri-Net. Then we can not only answer the question "who can do what under what kind of support", but also can tackle the issue "how to do" on top of the "what to do". Authorization policies in terms of avoiding conflict of interest are defined based on the conceptual model to prevent the authorization causing conflict. Authorization policies in terms of synchronization and dependency are defined based on the process model to ensure the authorization under the business process environment correct and proper. All above authorization polices will be enforced based on the conceptual model and the process model respectively. Moreover, authorization policies verification enacted on the process model is used to ensure that the authorization policies are defined correctly based on the process model. The research contributions of this thesis are presented in details as follows.

1.4.1 SOAC Conceptual Model

The SOAC conceptual model will handle composite web service authorization by dealing with not only a large number of service consumers, but also huge amounts of component web services, i.e. resources. SOAC conceptual model is a general framework for the authorization of composite web services as an extension of role based access control. This new conceptual model is based on the idea that the authorization of a composite web service to a service consumer should take the authorization constraints of the component services into consideration.

Therefore, in SOAC, by introducing the concept of **Resource**, the authorization policies of the component service are considered together with the authorization policies of the composite web service when judging if the permissions can be granted to a service consumer. The resource is used to support the operation of the composite web service. Following the same philosophy of role based access control (RBAC), we also identified

the element of **Service Consumer** as the entity that requires to access the operations of the composite web service. The two concepts, **Role** and **Resource Type**, are introduced in SOAC in order to reduce the administrative overhead in authorization management. Based on the characteristics of resources and service consumers, we classify resources and service consumers, map them into resource types and roles, and use resource types and roles to map with the **Operation** of a composite web service. Then the resource can provide support on the operation as specific resource type; while service consumer can access the operation as specific role.

Hence, SOAC is divided into two parts, *service provision* and *service realization* (See Fig. 1.1). In service provision part, the mappings between the elements of Service Consumer, Role, and Operation are illustrated. In service realization part, the mappings between the elements of Resource, Resource Type and Operation are presented. The mappings between the elements of Role (R), Operation (Op), and Recourse Type (ReT) integrate the service provision and service realization together. The access to the composite web service can be assigned to a service consumer if all the authorization policies of the composite web service and its resources can be satisfied. The concept **Session** is introduced at integration of service provision and service realization in SOAC. It is designed mainly for checking the conflict of interest in terms of authorization at runtime.

In summary, by introducing the concept of Role and Resource Type, the administrative overhead is dramatically reduced. When a service consumer or a resource needs to change its access requirements or support conditions on a specific operation, we just need change its belonging role or resource type. Relevant mapping between the role and the specific operation and the mapping between the resource type and the operation can remain. Moreover, by introducing the element of Resource and Resource Type, the authorization in composite web service not only considers "who can do what", but also takes "who can do what under what kind of support" into account, since granting permission to service consumer to access the specific operation will also depend on the availability of the resource that is also mapped with the specific operation as a resource type.

1.4.2 SOAC-Net Process Model

In this thesis, we design a Petri-Net based process model named SOAC-Net, which is developed based on the proposed conceptual model SOAC. SOAC-Net takes all elements and their associated relationships identified in SOAC into account. SOAC-Net is a Petri-Net based process model. The advantages of Petri-Net's graphically and mathematically founded modeling formalism with various algorithms for design and analysis make it a good candidate for modeling authorization flow in composite web services. SOAC conceptual model is lack of capability to capture the authorization policies used to manage the composite web service authorization under the environment of business process. Therefore, it is necessary to develop the process model.

SOAC-Net is used to represent an authorization flow that is the sequence of the access by service consumers and the sequence of the supports from components services. The authorization flow separated from the control flow of the operations can enforce the authorization policies regarding to the *synchronization* and *dependency* on top of the execution policy of a composite web service. This separation can facilitate the authorization management in business process environment on access control level only without delving into activity execution sequence level. The coordination on the accesses of service consumers and the supports of component services during the execution process of a composite web service is also facilitated by the enforcement of various types of authorization policies based on SOAC-Net.

SOAC-Net is divided into three parts, role-flow, resource type-flow, and constraint-flow. In SOAC-Net, a role-flow and a resource type-flow are derived from the execution sequence of the operations and associated authorization mappings, i.e. the mappings between role and operation, and the mappings between resource type and operation, that can be identified in SOAC conceptual model. These two types of flows are tightly synchronized with the execution sequence (control flow) of the operation. But they are not the same as the control flow since they have capability to manage the access sequence of role and the support sequence of resource type within one operation, where the control flow can only be used to coordinate the execution sequence between operations. Constraint-Flow is used to represent the various types of authorization

dependency policies based on the role-flow and resource type-flow.

1.4.3 Authorization Policies Enforcement

Conflict of Interest

Authorization Policies in terms of conflict of interest are designed based on the SOAC conceptual model. We categorize the authorization policies into two parts, (1) **Authorization Policies for Avoiding Conflict of Interest between Different Elements with The Same Type** and (2) **Authorization Policies for Avoiding Conflict of Interest between Different Authorizations for One Element**.

Authorization Policies for Avoiding Conflict of Interest between Different Elements with The Same Type are used to avoid the conflict of interest occurred between elements with the same type, e.g. conflict of interest between two service consumers or conflict of interest between two resources. The relationships between two elements with the same type in SOAC can be defined as **Exclusive** or **Non-exclusive**. Exclusive relationship means that two elements of SOAC, e.g., two service consumers, two roles, or two operations, are ostracized each other; while Non-exclusive relationship means that two elements of SOAC are not ostracized each other. The relationship between elements with the same type in different authorizations should be the same; Otherwise, conflict of interest will occur. For example, if Op_a and Op_b are exclusive operations, then the relationship between R_i and R_j that are mapped to the operations Op_a and Op_b respectively, should reflect the exclusive relationship of the mapped operations, i.e R_a and R_b are also exclusive roles. If R_a and R_b are assigned to service consumer SC_n and SC_m, respectively, then the relationship between the service consumers SC_n and SC_m must be matched with the relationship between the mapped roles. If service consumers SC_n and SC_m are non-exclusive with each other, then SC_n and SC_m can not be assigned roles R_i and R_j respectively at the same time since the two roles have the exclusive relationship. If they are assigned with the roles, conflict of interest occurs.

Authorization Policies for Avoiding Conflict of Interest between Different Elements with The Same Type highlights the relationships between different service consumers and/or different resources. However, in some circumstance, it is not easily to identified the relationship between two different service consumers or resources, due to the features of web service environment where each service does not want to expose its internal relationships with others. Therefore, we develop the second type of authorization policies focusing on the possible conflict of interest related with one service consumer, one resource, and one pair of service consumer and resource, i.e., **Authorization Policies for Avoiding Conflict of Interest between Different Authorizations for One Element**. Conflict of Interest that can be avoided by enforcing this type of authorization policies can occur, when one service consumer or one resource is assigned with exclusive roles or resource types. By this way, we can identify conflict of interest without needing to know the relationship between different service consumers and resources.

Each type of authorization policies to avoid conflict of interest is classified into static policies and dynamic policies, that are related with the design time conflict of interest and runtime conflict of interest. In some circumstance, it is forbidden to map the elements that can cause the conflict of interest. This is a kind of strictest way to manage composite web service authorization. All possible conflict of interest are prevented at design time. On the other hand. In some circumstance, we do not need such strictest way. A more flexible management method will be required. It is a type of trade-off between flexibility and security. Hence, we restrict that the mapping that can cause conflict of interest can be created at design time, in case of the situation where the mapping will not be activated simultaneously. Without simultaneously activating the assigned exclusive roles or exclusive resource types, the service consumers or resources can be assigned with these role and resource types at the same time, but not be able to activate them at the same time. For example, a person can be assigned as role of `Teller` and `Desktop Supervisor` in a bank branch, where `Desktop Supervisor` will verify the work of `Teller`. We can loose the security by allowing that a person can be assigned both roles at the same time, but requiring that he can not activate both

roles simultaneously. Hence, he can not verify the work made by himself, but he can also work as `Desktop Supervisor` to verify the work of other `Tellers`. In this case, the security restriction on avoiding conflict of interest is enforced at runtime.

The conflict of interest can occur in service provision part caused by the service consumer, and can happen in service realization part caused by the resource. The conflict of interest can also happen at integration part of SOAC by both service consumer and resource, and even by the groups of service consumers and resources. we can identify the four types of authorization policies at both categories-(1) *for different elements with the same type*, and (2) *for different authorizations with one element*, and at both design time and runtime. These four types of authorization policies in terms of conflict of interest are defined based on each part of the SOAC conceptual model. They are presented as follows,

- For service provision part, authorization policies can be used to avoid the conflict of interest for different service consumers with the same type, and for different authorizations with one service consumer.

- For service realization part, authorization policies can be used to prevent the conflict of interest for different resources with the same type, and for different authorizations with one resource.

- For the integration of service provision and service realization, the conflict of interest between service consumers and resources can occur, if the relationship between the service consumers and the resources is not the same as the relationship of their mapped role and resource type. Moreover, the authorization policies can also be used to avoid one web service to play as conflicted role and resource type simultaneously.

- At integration part of service provision and service realization, conflict of interest between one pair of service consumer/resource and other pairs of service consumer/resource is another new type of conflict of interest which can be identified in SOAC. A service consumer and a resource are put in one pair when the service consumer requests the access of the operation of a composite web service

and the operation needs the support of the resource. The relationships between pairs of role/resource type reflect the relationships between operations mapped to these pairs of role/resource type. If two pairs of service consumer/resource have the relationship Exclusive or Non-exclusive, the pairs of mapped roles and resource types must have the same relationship as Exclusive or Non-exclusive. This type of authorization policies can also avoid assigning two conflicted pairs of role/resource type to one pair of service consumer/resource.

Synchronization

The sequence of accesses and supports by role (a group of service consumers) and resource type (a group of component services) respectively on the operations of the composite web service must be consistent with the operation execution sequence, i.e. a role-flow and a resource type-flow must be synchronized with the control flow of operations. The reason is that, a service consumer as assigned role or a resource as mapped resource type is not able to access or support the past operations (the operations that have been accessed or supported) unless the business logic of the composite web service allows, nor can they access or support the operations that are needed in the future but not now. Without this synchronization restriction on the role-flow and resource type-flow, the service consumer and the resource can obtain the right to access and support any operation without considering the well-defined execution sequence of the operations. Such chaos and disordered authorization could do harm to the whole execution of the composite web service, where the needed service consumer or resource is not available, or the service consumer and the resource do extra and unnecessary work causing security issue. Therefore, we need to devise synchronization policies to restrict the access sequence of role and the support sequence of resource type based on the control flow of the operations in the composite web service. Authorization synchronization policy is divided into two types, (1) Role Synchronization Policy and (2) Resource Type Synchronization Policy. (See Fig. 1.5)

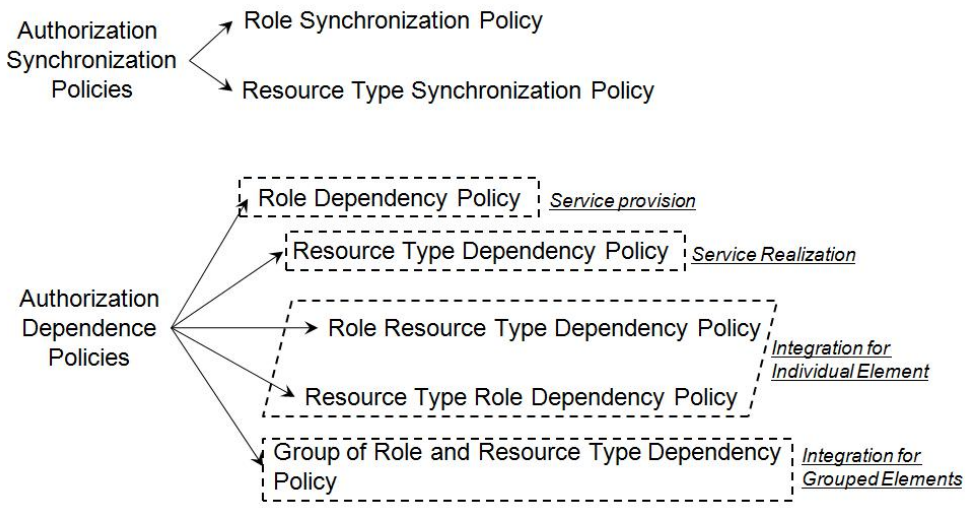

FIGURE 1.5: Authorization Policies for Synchronization and Dependency

Dependency

Authorization Dependence Policies restrict that a role or a resource type is not able to access or support the operations, until another role or another resource type has already accessed or supported specific operations. However, although the depended access or support has been made, the execution of the depending access or supports is not a must. The depended access or support is only a necessary condition for the depending access or support, but not a sufficient condition. Authorization Dependence Policies are separated into 5 categories, (1) between roles, (2) between resource types, (3) between roles and resource types, (4) between resource types and roles, and (5) between groups of roles and resource types. By defining the dependency policies, we can further restrict the access sequence of role and the support sequence of resource type on top of their synchronization with the execution sequence of the operations. (See Fig. 1.5)

1.4.4 Authorization Policies Verification

Authorization policies are enforced based on the process model SOAC-Net. However, checking whether all policies defined in SOAC-Net are correct is still an issue that is not

tackled. The improper authorization policy definition can even cause the cancelation of the execution of a composite web service due to the missing of necessary accesses and supports from service consumers and resources, respectively. Therefore, we need to provide a verification mechanism on the authorization policies based on SOAC-Net to ensure the defined authorization policies can consistently work with the execution of the operations. We call authorization flow that can consistently work with the execution sequence of the operations of composite web service as *reliable authorization flow*. However, when an authorization flow becomes unreliable, a dead state definitely occurs in SOAC-Net, i.e., the necessary role or resource type is missing that causes the service consumer or resource not being able to access and support the operations. For example, if an important operation is not mapped with any role, how can service consumer as specific role access the operation? Then how the composite web service can execute further with an operation without any access from service consumer?

Petri-Net [44] provides a set of verification mechanisms[46], and its graphically and mathematically founded modeling formalism with various algorithms for design and analysis[45] makes it a good candidate for modeling authorization flow. *Dead marking* is a property of Petri-Net based process model that can be used to detect the dead state in SOAC-Net. *Dead Marking* in Petri-Net model means a markings having no enabled binding elements. In Petri-Net model, markings are used to model the steps of the execution of the Petri-Net based process model. When a marking has no enabled binding elements, it means, that the marking can not be moved into next marking, i.e., the process mode is in a dead state. Hence, based on the property of *Dead Marking*, we develop a new property for the purpose of authorization reliability verification based on SOAC-Net, *authorization-embedded dead marking freeness*, which can verify the *Improper Authorization Policy Definition*. Formal verification approach associated with the property is also presented.

1.4.5 SOAC-Engine

The *SOAC Engine* is developed with JAVA and Oracle, and resided in local web server where the composite web service is running. It transfers the authorization messages between service consumer and the engine, and between the engine and the component web service through `Application Interface`. `Authorization Management Interface` of *SOAC Engine* is used by administrator to (1) set the elements of SOAC and their relationships, (2) identify the conflicted relationships among elements, and (3) manage the authorization policies for both composite web services and resources.

The *SOAC Engine* includes four component packages: (1) `Authorization Administrative Management` manages the elements of SOAC and their relationships, e.g., querying the assigned roles for specific service consumer. (2) `Credential Management` manages credentials for service consumers and resources. (3) `Authorization Decision Making` performs the policy compliance checking to make authorization decisions. The conflict of interest is checked in this packages. (4) The `Authorization Enforcement` controls the enforcement of authorization policies related with the composite web service and its component services.

1.5 Organization of the Thesis

In this thesis, we provide a general authorization framework to manage the composite web service authorization. Conceptual model and associated process model are developed. Based on the models, various types of authorization policies are enforced and verified to guarantee the successful composite web service authorization. The rest of this thesis is organized as follows,

Chapter1 As show in Fig. 1.6, we start in Chapter 1 by providing an introduction to the research. In this chapter we introduce the reader to the main topics of the thesis, identify the research issues, and outline the research proposal to resolve these issues.

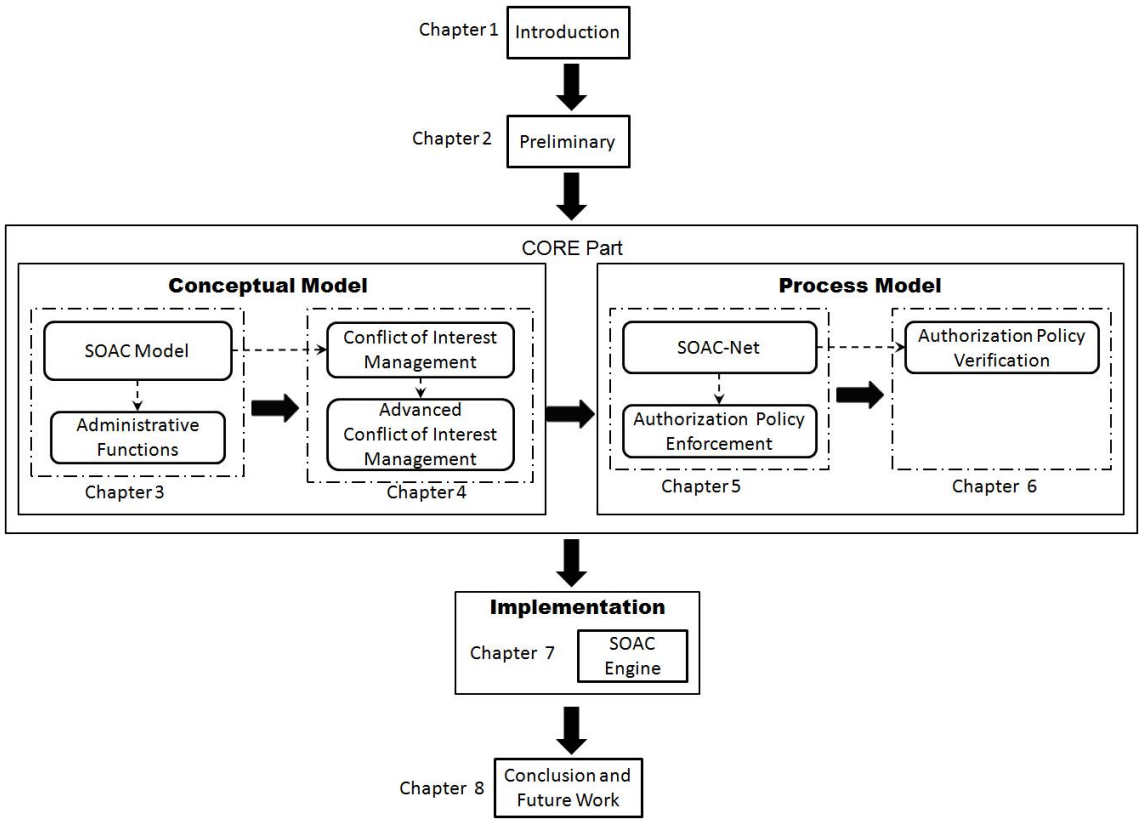

Figure 1.6: Organization of the Thesis

Chapter2 Next, in Chapter 2 we analyze related work in this area to identify current shortcomings and problems based upon which we extrapolate the requirements for our approach. We firstly review the work in Service Oriented Computing (SOC), where service composition is one of the main part in the area. We also examine the works in Business Process Management (BPM), which facilitates the management on the execution sequence of a web service composition. Secondly, we will overview the research in existing authorization model, like MAC, DAC, and RBAC. We will particularly analyze the existing family of RBAC models, where our proposed model are derived from. Finally, preliminary knowledge regarding to the Petri-Net and its associated verification methods is presented.

Chapter3 As show in Fig. 1.6, in Chapter 3, we are stepping into the core part of the thesis. In this chapter, we will propose the conceptual model of service oriented

authorization control (SOAC). Elements and their relationships in SOAC will be exposed in this chapter. Various types of administrative functions on managing these elements and relationships are also introduced in this chapter. These functions are programmed in SOAC Engine as our implementation.

Chapter4 Chapter 4 mainly focuses on managing conflict of interest in terms of authorization based on the conceptual model SOAC. We will examine the relationships between the same type of elements in deferent authorizations to avoid conflict of interest. Furthermore, a series of extended authorization policies are used to avoid conflict of interest for one element in different authorizations, e.g. one service consumer or one resource. Both types of authorization policies to avoid conflict of interest are classified into four categories at design time and runtime. These four categories are classified based on the features of a service composition, i.e. at different parts of service composition-service provision, service realization, and integration part.

Chapter5 In Chapter 5, a Petri-Net based process model SOAC-Net is developed based on the conceptual model SOAC. Authorization Synchronization Policies and Authorization Dependency Policies are proposed in this chapter. These two types of authorization polices can be enforced based on the SOAC-Net.

Chapter6 In Chapter 6, we tackle the issue of Improper Authorization Policy Definition that can cause the authorization flow unreliable. Based on the verification methods originating from the common Petri-Net, we develop the property, *authorization-embedded dead marking freeness*, to verify the reliability of authorization flow.

Chapter7 In Chapter 7, we propose a SOAC Engine as our implementation. SOAC Engine is developed based on the SOAC conceptual model and SOAC-Net process model. It can provide authorization to service consumer by considering the situation of the component web service. The authorization will be granted based on the enforcement of the authorization policies. Verification on the authorization policies is also provided by SOAC Engine for the system administrator.

Chapter8 In Chapter 8, we provide concluding remarks of this thesis and discuss directions for future research.

2

Preliminary

2.1 Introduction

Business Process Management (BPM) is a novel technology addressing how the organizations can identify, model, develop, deploy, and manage their business processes including processes that involve IT system and human interaction [10]. The Service Oriented Architecture (SOA) paradigm defines the concept of applications consisting business process as service unit. A group of service units are composed and executed together as an orchestrated sequence of messaging and event processing. Consequently, the confluence of SOA and BPM is resulting in a new process-centric paradigm that delivers the enterprise business logic in web service interaction as a combined effect of business process logic and application control logic. In this chapter, we will briefly introduce the relevant terminology and technology of Business Process Management(BPM)

as well as Service Oriented Architecture (SOA), since the principle and theme of our authorization models in service composition are derived from the convergence of these two areas.

Followed by introducing on BPM and SOA, reviewing the state of art on existing authorization management based upon a historical point of view is also necessary, especially for us to compare their advantages and limitations in the literature. Role based access control (RBAC) has become the most popular security mechanism implemented in industry products and attracts more and more researchers to focus on maturing the RBAC models. Hence, in this chapter, a family of RBAC models will be addressed in details. There are also many role-based authorization models that have been created in web service domain. However, they are not suitable for managing composite web service authorization. The advantages and the limitations of their work will also be exposed in this chapter. Authorization in business process management is another type of related work that should be stated in details in this chapter. Our proposed authorization model in this thesis is developed based on the existing role-based authorization models in both web service domain and business processes environment.

Since Petri-Net is the formalization intended to support the construction and development of our process model, its variation and corresponding verification mechanism are also presented in this chapter.

The rest of this chapter is organized as follows. Section 2.2 introduces the Business Process Management and Service oriented Architecture. A review of state of art on authorization models is presented in section 2.3. Finally, section 2.4 introduces the Petri-Net with its variation and verification methods.

2.2 Service Oriented Architecture and Business Process Management

2.2.1 Service Oriented Architecture

SOA is a novel philosophy for building distributed applications in which the elementary unit is a service. In SOA, a web service is a platform-independent, loosely-coupled, self-contained programmable web enabled application that can be described, published, discovered, coordinated and configured using XML artefact for the purpose of developing distributed inter-operable applications [13].

Under XML artefact, the service is standardized as several protocols for web service publishing and discovering, such as Web Service Description Language(WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery, and Integration (UDDI). Moreover, a series of extended web service specifications are created for the service orchestration and choreography, which state the service composition and service interaction respectively.

In a word, by using web service technologies in SOA , enterprise can flexibly solve enterprise-wide and cross-enterprise interaction challenges and thereby quickly address business challenges and opportunities [12].

Web Service Infrastructure

In order to implement the function of the web service such as platform-independent, the web service specifications require [16]:

1. A common syntax to standardized the data structure and formats,
2. A mechanism to define that a message regulated by common data format must interact with other remote sites according to the specific interaction forms through a series of binding on mapping message into a transport protocol,
3. An interface to describe service,
4. A name and directory of a service to locate where it is.

XML as a markup language can provide standard syntax for all of web service specifications through common XML schema mechanism which can define the format and structure of each specification according to own functional requirements.

Simple Object Access Protocol - SOAP is a standard for exchanging XML-formatted messages among web services [17]. It standardizes the message transferring among the organizations into a common data format, and defines PRC-style and Document-style as the interaction models for message transferring. SOAP is naturally wired with HTTP to receive and send transport protocol packets.

Web Service Description Language - WSDL is an XML format for describing network service as a set of endpoints operations on messages containing either document-oriented or procedure-oriented information [18]. WSDL separates the service endpoint into abstract definition and concrete definition. In former one, the *messages* which are the abstract description of the data being exchanged and the *port types* which are abstract collection of operations are stated. In latter one, it binds data format specification for a particular port type with the concrete protocols e.g SOAP or HTTP.

Universal Description,Discovery, and Integration - UDDI is a platform-independent, XML-based registry for organizations to list services of themselves on the Internet. It defines a set of services supporting the description and discovery of (1) businesses, organizations, and other Web services providers, (2) the Web services they make available, and (3) the technical interfaces which may be used to access those services [19]. Three inter-related components are included: *white pages* which define the address and other key points of service contact, *yellow pages* which classify the information according to the industry based upon the industrial taxonomies, and *green pages* that describe the service including the references to the specifications for webs service and pointers to the file and URL based discovery mechanism.

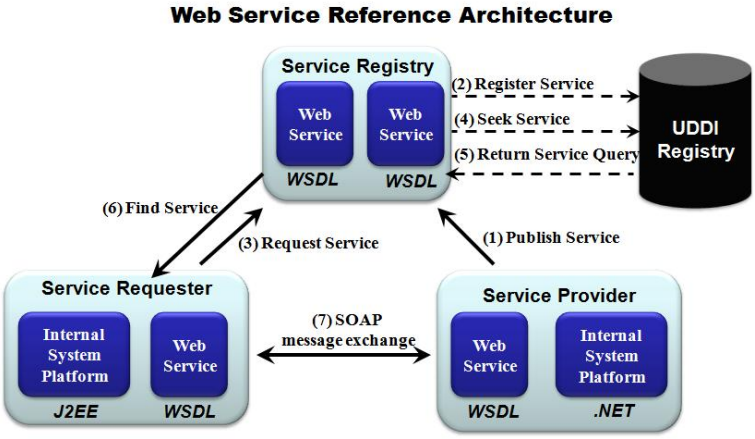

Figure 2.1: Web Service Reference Architecture

Through the three XML-based standardized protocols, the web service infrastructure can implement the platform-neutral interactions in the loosely coupled web environment, see Fig. 2.1. Three entities involve in the web service reference architecture: *service provider*, *service requester*, and *service registry*. Obviously, WSDL provides standard service interface which hides the heteronomous system platforms in each entity, e.g J2EE for service requester and .NET for service provider respectively. SOAP encapsulates the message transferred between the entities with the specific binding which is independent to the transport protocol. The service provider, to begin with, registers the service in the service registry which stores the necessary information of the service in the UDDI repository [103]. As soon as the service requester queries the service in service registry, ideal service information including the contact and location information will be returned to the service requester so that the service requester will know how and where to interact with service provider on the loosely coupled web environments.

Web Service Composition

Generally, In order to provide value-added service to the service consumer, also known as service requester in Web Service Reference Model, the organization will compose a series of component services together to provide comprehensive functions to service

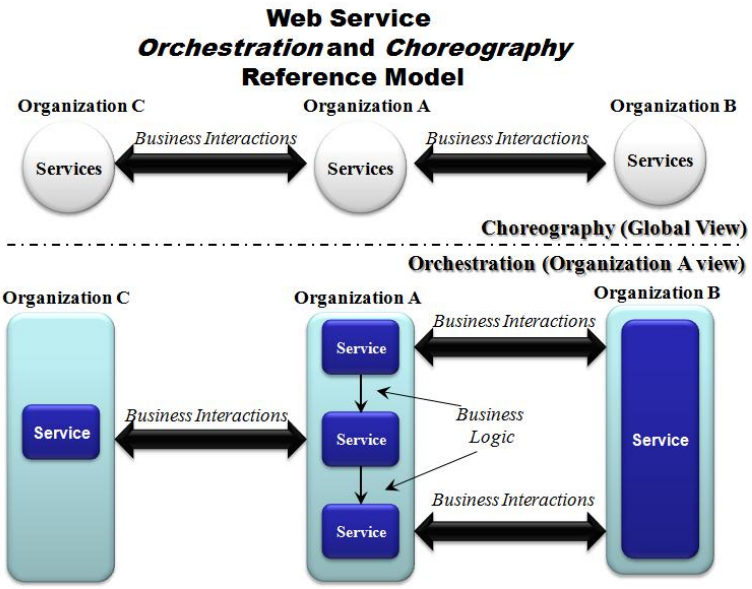

FIGURE 2.2: Web Service Orchestration and Choreography Reference Model

consumers by complex coordinations [102]. Therefore, A *composite service* is also a service that aggregates multiple component services and adopts the functions of theses component service based upon specific business logic. The term *orchestration* and *choreography* are two views on service composition. SOC as a distributed application integration paradigm is then constructed and dynamically facilitated from these two aspects [38].

Orchestration deals with the description of the interactions in which a given service can engage with other services as well as the internal steps between these interactions. Business Process Execution Language for Web Service (BPEL4WS) is the leading specification in this view of point [36]. *Choreography* captures collaborative processes involving multiple services and especially their interactions seen from a global perspective. The representative specifications in choreography point of view are Web Service Choreography Interface (WSCI) and Web Service Choreography Description Language (WS-CDL) (See Fig. 2.2).

In a word, the *orchestration* refers to an executable business process that can interact with both *internal* and *external* web services at message level which include the

business logic and task execution order [37]. *Orchestration* always represents control from one party's perspective. *Choreography* mainly focuses on collaborative behavior of the organization. In choreography, the public message exchanges between organizations are more concerned–rather than a specific business process executed in an individual organization. Choreography tracks the public message sequences among collaborative organization from a global view only.

For example, in Fig. 2.2, from global view, the web services in three organizations are interacted together. The web service choreography will only focus on these public interactions among the web services, and ignore how each web service involve in these interactions. All service interactions can be identified from web service choreography. On the other hand, we choose organization A's view to illustrate the web service orchestration in Fig. 2.2. From service orchestration point of view, each organization involving in these web service interactions in Fig. 2.2 has their own opinions on these service interaction. If we take organization B's orchestration view as example, then services and associated service interactions in organization C can not be viewed completely through organization B's orchestration view since they are not interleaved with the services in organization B. In web service orchestration, the business logic of the web services of the organization and associated service interactions are the concerns.

Our authorization models are developed from web service orchestration's point of view, i.e the composite web service's point of view, where the authorization management mechanism will be implemented to control the access and support from service consumer and component service that interact with the composite web service. How the service consumer or the component web service to interact with other web service and how to manage the related authorization issues in terms of the service consumer and component web service on the other web services are not taken into account in our authorization framework. We believe that, each web service involving in the multiple and cross organizational business interactions should bear its own authorization management systems. An central controlled authorization system developed based on web service choreography's point of view is not realistic. Here below, several representative web service protocols in web service composition are introduced in details.

Business Process Execution Language for Web Service - BPEL4WS is an XML-based executable langauge which defines a model and a grammar for illustrating the behavior of a business process based on the interactions with the services of other participants [41]. BPEL4WS is developed from web service orchestration's point of view. BPEL4WS composes a series of services within one organization as service orchestration, some of which may require interacting with other services resided in the process of organization's collaborators, and some of which may not as internal actions. BPEL4WS defines the service process from individual organization point of view that is distinguished with the service choreography which takes care of the observable behavior of each participant in the business collaboration only, treated as the business interaction. BPEL4WS has become a basic XML-based specification of service composition which can generate value-added nest service.

BPEL4WS uses *partnerLink Type*, *partnerLink*, and *Endpoint References* to describe the cross-enterprise business interactions in which the business process of each organization interacts through web service interface with the processes of other organizations. Moreover, a series of basic activities to illustrate the behavior of the business interactions and the structure activities to map the process sequences in each organization are defined in BPEL4WS. Detailed workflow patterns and communication patterns that the BPEL4WS provide can be recognized in [47]. *Data handling* and *fault handler* associated with *event handler* are encapsulated in the *Scope* to coordinate the execution of the process under a consistency manner. Message correlation is also included to ensure the reliability of message transferring in and out of the organization.

Web Service Choreography Interface - WSCI is an XML-based interface description language that describes the flow of message exchanged by web service participating in choreographed interactions with other service [40]. WSCI is a dynamic interface of web service operated with the static interface of web service-WSDL. WSDL only defines the content of the service, such as the name and type of the required message, or the *portTypes* which provide operations to deal with the messages. However,

WSCI mainly focuses on the execution order of the operations and messages of interactions defined in the WSDL. These interactions are then observed as the behavior of the participant. Another WSCI global model is introduced to accentuate the connection of the behavior of the participants and the behavior of its collaborators.

Web Service Choreography Description Language - WS-CDL is an XML-based language that describes peer-to-peer collaborations of participants by defining, from a global view point only, their common and complementary observable behavior where ordered message exchanges result in accomplishing a common business goal [39]. Through describing the components of *roleType*, *relationType*, *informationType*, *tokenType* and *channelType*, WS-CDL describes a choreography of business interactions among the participants which can represent their observable behavior only.

2.2.2 Business Process Management

A business process is a real-world activity consisting of a set of logically related tasks that produce a business outcome according to the correct business rules, when performed in the appropriate sequence [5]. Business process management is to support business process using methods, techniques, and softwares to design, enact, control and analyze operation process including human, organizations, applications, documents and other sources of information.[11] There are four stages during the lifecycle of business process: design, process configuration, process enactment, and diagnosis [35]. In design phase, based on the business requirements, the process is modeled and created. In configuration phase, the process is configured according to the system, e.g. workflow system [20, 21]. The process is performed in the process enactment phase. Finally, the diagnosis phase monitors, analyzes and optimizes the process [10].

Therefore, technology of business process management is, to a few extent, related to the workflow technology, which means that passing paper from person to person in enterprise is presented as flowchart. Workflow technology is responsible for managing the flow of work and informationalize the content of documents, thereby enabling work

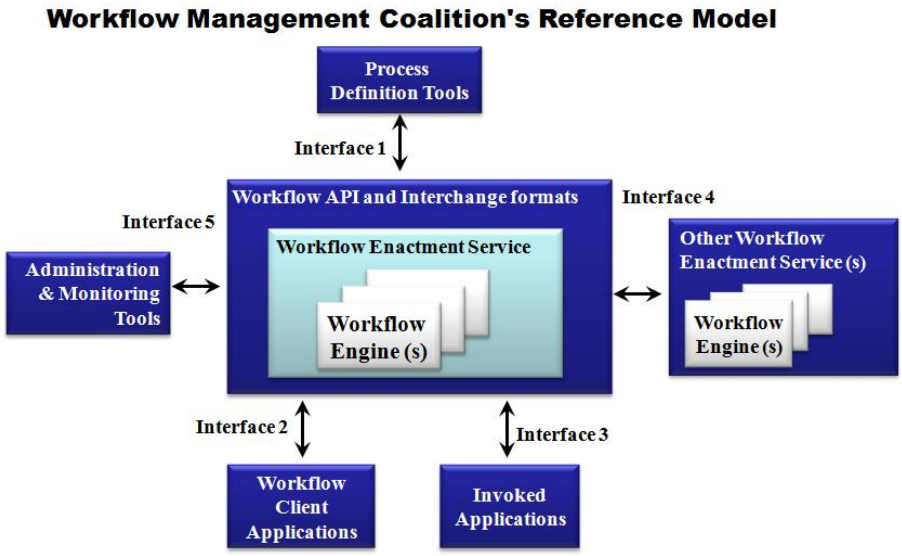

FIGURE 2.3: Workflow Management Coalition's Reference Model (© WFMC)

automated and paperless. The Workflow Management Coalition (WFMC) defines the *workflow reference model* such as Fig. 2.3 [15] which generally describes the architectures of workflow management systems and summaries the main components in the systems and the associated interfaces. The heart of the workflow system in the reference model is the *Workflow Enactment Service* [14], which ensures the right activities are carried out in the right order and by the right people. The *Process Definition Tools* illustrate the processes and the organizations used by the *Workflow Enactment Service*. Through *Workflow Client Applications*, a work item can be specified to the individual employee to perform the task, while the *Invoked Application* is the software application executed for the specific task. All of workflow tracking case control are distracted to the *Administration and Monitoring Tools*. However, BPM is a further step on workflow to accomplish the business objective. Not only inherited the flowlization, BPM also provides simulation, verification and validation of process design, and collects and interprets the real-time date. In a summary, BPM supports various phases of operational business process from design, configuration, enactment, even to diagnosis [25–28, 104].

Nowdays, with the emergence of cross-organizational business process, BPM is further conflated with the SOA technology, a forefront of enabling a desired degree of process independence [22–24]. Service Oriented Computing (SOC) is then generated as an innovative distributed computing to implement the integration of business service applications in the multiple heterogeneous system platforms and collaborate the autonomous organization as a whole.

2.3 Authorization Models

2.3.1 Role-Based Access Control

Role based access control is a very popular security mechanism implemented widely in industry products and has attracted a lot of research interest in academic domain, recently. In [52, 53], a basic Role-based Access Control (RBAC) model is presented. The security policy in RBAC does not directly grant permissions to users but to appropriate roles. In Fig. 2.4, a complete RBAC conceptual model is presented, that we will introduce in details later. The RBAC model shown in Fig. 2.4 has experienced evolvement since 1970s. As shown in Fig. 2.5, $RBAC_0$ is a basic RBAC model, where only the major elements and associated relations, e.g. User, Role, Permission, Object, and Session, are presented. $RBAC_1$ is evolved from $RBAC_0$, where role hierarchy is added on top of the basic RBAC model. In $RBAC_2$, several authorization constraints are added to restrict the mappings between elements based on $RBAC_0$, to satisfy specific business security requirements. The consolidated model $RBAC_3$ is from the combination of $RBAC_1$ and $RBAC_2$, i.e the model shown in Fig. 2.4.

Basic Model-$RBAC_0$

In basic model $RBAC_0$, most elements related with the authorization are identified as well as their associated relationships. User (U) in RBAC model represents the subject, i.e. an entity that requires the right to access. It can be human being in normal

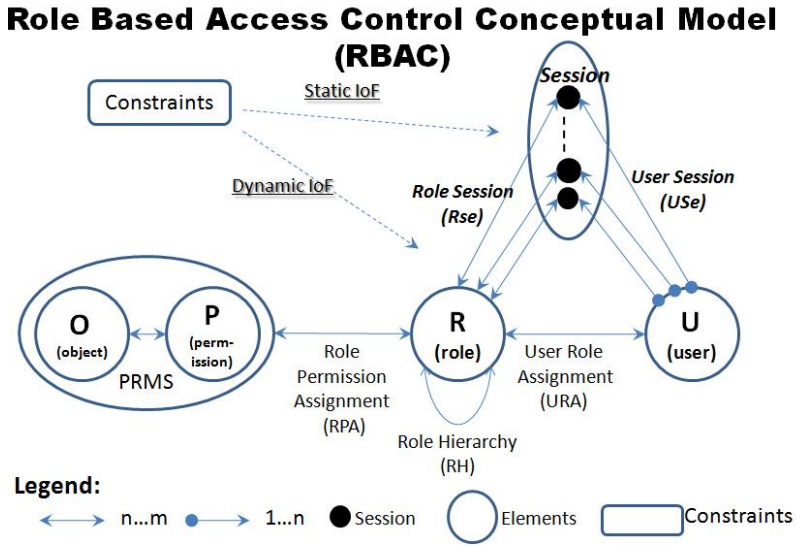

FIGURE 2.4: Role Based Access Control (RBAC) Conceptual Model)

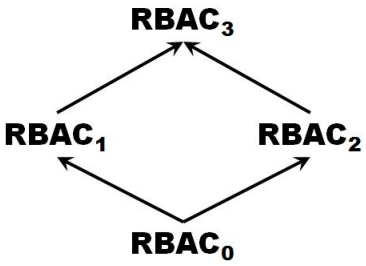

FIGURE 2.5: A Family of Role Based Access Control Models

security system in commercial company or organization. It also can be an application or even a paragraph of program code which requires the right to do something.

Role (R) in RBAC is a core idea that contributes the popularity of RBAC security mechanism. Role is a group concept that can encapsulate a type of users. It is generally a position in commercial company or organization, e.g. programmer, accountant, or HR Supervisor. In normal RBAC system, the roles can represent the structure of an organization.

Permission (P), also known as access right or privilege, is a behavior that can be operated on the object by the subject. For example, the permission in database domain can be Read, Write, Update, Open etc. These permissions can be implemented

on specific object.

Object (O) is the entity that can be accessed by the subject. The objects in database domain, for example, can be tables, columns, and even a database. Since the permission can only be used on specific object, their relationship becomes stable. Hence, sometimes, the permission and the object are mapped together as a group idea PRMS, or called permissions only. For example, the read, write and update permissions can only be used on tables in database, and open permission is only used on database. Therefore, sometimes, read-table, write-table, and update-table are grouped together to be mapped with role. If a role is mapped with the permission, e.g. read, then read-table will be assigned to the role automatically for concision. The mappings between elements will be introduced in details later in this section.

Session (Se) is created when a user is activating a subset of assigned roles. When the user stops activating the roles, the session is terminated by the system.

Now let us introduce the mapping relations between these elements. User Role Assignment (URA) is used to create the relations between user and roles. It is a many-to-many relation where a user can be mapped with multiple roles and a role can be assigned to multiple users. Role Permission Assignment (RPA) is used to create the relations between roles and PRMS (the relations between permissions and objects has been introduced already.). It is also a many-to-many relation, where a role can be mapped with multiple PRMS, i.e. with multiple permissions on specific objects, and a PRMS can be mapped to multiple roles. Through these two assignments, a user can ultimately obtain specific permissions on specific objects from the assigned roles. This greatly reduces the administrative overhead on managing user-to-permission relations directly. The efficient advantage has been introduced in section 1.1.2. Moreover, the role-permission (also known as PRMS) is stable and not easy to change. It can be pre-defined in a security system. On the other hand, the user-role relation is more dynamic and needs improvised management, e.g. to decide which role the user belong to. By introducing the concept Role in RBAC, the total authorization work is separated as determining the relation of role-permission (or PRMS) and determining the relation of user-role. Since without deep knowledge of the business, it is hard to set up the

role-permission relations as it is about determining the structure of the organization. Therefore, once relation of role-permission is set up, it is seldom to require change. However, determining which role the user should belong to is easier and may occur frequently for system administrator on conferring or revoking the relations according to the specific business policy of the company, e.g., when HR supervisor issues an assignment on George as a Sales Team Leader (STL), the system administrator just needs to create the relation of George-STL based on the assignment from HR Supervisor without the deep knowledge on what the sales team leader can do.

As shown in Fig. 2.4, User Session (USe) is a mapping between user and sessions. It is a one-to-many relation where a user can create multiple sessions, and a session can only belong to one user. Role Session (RSe) is a mapping between the session and the activated roles by users. It is a many-to-many relation where both role and session can be mapped each other with multiple across relations. By introducing the concept Session, how the user activates the assigned roles at runtime can be considered by RBAC.

Role Hierarchy-$RBAC_1$

The basic model of RBAC exposes the advantages of RBAC by introducing the concept Role and Session. However, naturally, based on the structure of the organization, there exists a relationship between roles, i.e. role hierarchy. For example, sales person and sales team leader (STL) are two roles within an organization. Obviously, the STL is the supervisor of sales permission, whose rights include the right of the sales person as well. In $RBAC_1$, the role hierarchy is presented, that is mathematically reflected as partial order, where the senior role can inherit the permissions of the junior role.

Constraints-$RBAC_2$

In $RBAC_2$, various types of constraints are introduced to restrict the mapping between the elements, particularly for the relation between role and user. The constraints can be summarized as follows,

- Separation of Duties (SoD): SoD [56, 57, 100] is used to restrict that a user can not be assigned exclusive roles to avoid conflict of interest. For example, the roles of purchase manager and account payable manager can not be assigned to one user, since cheating on purchase can occur. SoD can be enforced at both design time and rumtime. At design time, a user can not be assigned with exclusive roles. At runtime, a user can be assigned with exclusive roles, just in case that the user can not activate the exclusive roles simultaneously. The constraint used at design time is called Static SoD; while the constraint used at runtime is called dynamic SoD.

- Cardinality: This constraint is used to restrict the amount of assignment of roles to a user. For example, the role of CEO in a commercial company can only be assigned to one user.

- Binding of Duties (BoD): BoD, on the contrary of SoD, is used to restrict that if a user is assigned with specific role, then another role must also be assigned to this user, i.e. the role is bound with another role on assigning to the same user.

There are still many other constraints enacted on $RBAC_2$ to restrict the authorization. Here we just present three main constraints.

Consolidated Model-$RBAC_3$

$RBAC_3$ is consolidated from the $RBAC_1$ and $RBAC_2$. Therefore, the constraints in $RBAC_2$ can also be enforced for the role hierarchy.

2.3.2 Authorization Model in Web Service Domain

However, traditional RBAC model is only suitable for authorization management within individual organization. In RBAC, the object (also known as resource) is assumed as a constant concept, which quantity is small, and can be fixed in advance and less changed. However, in loosely-coupled web service environment, resource that is

needed to support the function spreads across-organizational boundary and is composed in highly dynamic fashion. Hence, the dynamicity of resource in service environment makes the authorization management in composite web service complicated. Research has been done in service composition security by enhancing RBAC. We shall look into some representative works in the area.

In [58], the authors propose a policy integration framework for merging heterogeneous Role Based Access Control policies of multiple domains into a global access control policy, where the conflicts that arise among the RBAC policy of individual domain can be solved by an integer programming based approach. Moreover, the authors in [59] propose a model named distributed Role Based Access Control (dRBAC) to manage authorization for systems that can span multiple administrative domains. These researches both distinguish themselves from previous traditional RBAC by being implemented in a distributed environment. However, web service environment is different from the distributed environment in that the web services are self-contained and autonomous; while the applications in distributed domain to some extent share the similar background and require an integration management from a global view. Therefore, most approaches designed for distributed authorization management are to develop a global view management based on the authorization management within individual domain. However, in web service environment, the web services are loosely coupled and belong to different organizations without sharing any background or management. It means, an authorization management from global view in web service environment is not realistic.

Therefore, most security experts become focusing on the authorization management in web service domain. The authors in [60] propose an access control model CWS-RBAC which takes the composite service into consideration. In CWS-RBAC, a global role is assigned to service consumer to gain the permission to access the composite service and a local role mapped from global role is assigned to service consumer to access the other component services. The authors in [61] propose another concept-*Role Composition* where two types of roles are composed together, global role and local role. The paper analyzes how the local role issued by the individual component service

is mapped to the global role from the composite services. In that case, if the service consumer is assigned with the global role, then it automatically bears the permissions of bound local role on the component service. However, by using such access approaches, the "role" as a concept used by specific service to manage the authorization is part of internal security policy within each web service and can not be identified by the other services. For example, the composite web service can not identify which role that it can be assigned by the component web service that it needs to access. All that the composite web service can perceive are what permissions can be granted based on what type of credential, i.e, the authorization constraints (the public part of authorization policy of each web service). Hence, the mapping of the global role issued from the composite service with the local role generated in other component services is not realistic.

The authors in [62] develope another extended RBAC model, WS-RBAC. Three new elements are introduced into the original RBAC model, named enterprise, business process and web services. The authorization constraints are described in WS-Policy [42] and WS-PolicyAttachment [43]. However, these two standards are designed for message level web services security. The security methods regarding to protect the information of the operations of web service are totally missing.

In [63, 64], the authors propose a RBAC framework to manage access control in WS-BPEL [41], named RBAC-WS-BPEL, and specify the authorization constraints on the execution activities, i.e., web service's functions. In this framework, the role is assigned to user (human) to gain the permissions on the activities. A language named BPCL to express these constraints based on XACML [65] and an algorithm to check the consistency of the access control model are developed. However, it only focuses on managing authorization of web service in terms of which staff in the organization can be granted rights to deal with the message received from specific operations of the web service in this organization. The protection on preventing the service confidential from illegal access of other web service is totally ignored. The framework can not support the composite service authorization where service consumer should be the subject to be granted the right to access the operation of composite web service, but not the human

being within the organization.

In [66, 67], the authors propose a logic framework for reasoning about the access control for WS-BPEL. The exchange of requests between business partners for supplying or declining missing credential is considered. However, how to coordinate the access control of component service with the authorization of composite web service is not taken into account. The paper only provides reasoning methods in terms of detecting if the credential that the service consumer can provide is matched with the credential that the composite service requires. They believe if the entity of the service consumer can be identified by the credential, then the access right can be granted to service consumer where the authorization management regarding to the component service is missing.

In [70, 71], the authors provide an enforcement and verification approach to guarantee that a service choreography can be successfully implemented between a set of web services (service consumer and the composite web service), based on their authorization constraints. However, the paper did not mention how to manage the access control after the authorization constraints of the composite web service are satisfied. The authorization constraints of the supporting resources are totally ignored.

In [68, 69], the authors propose an authorization specification for WS-BPEL, where the involvement of human being is also considered. However, the papers, like [63, 64], also emphasize the involvement of human being on handling the message received from specific operations of web service within one organization. The access management on which operation of the web service can be accessed by what service consumer is completely not touched.

Conflict of interest is a major concern in traditional RBAC models. In order to deal with conflict of interest, static and dynamic separation of duty mechanisms are defined in RBAC standard [54, 55]. The authors in [72] have discussed the conflict of interest in the authorization of web services. However, this research deals with the authorization of web services using the same way as those authorizations in close systems. In particular, the features of composite web services have not been taken into consideration. It is lack of existing works to identify and deal with possible types of

conflict of interest for service consumers, for component services, and between service consumers and component services in composite web services.

Although plenty of existing enhanced RBAC mechanisms and approaches have been presented which focus on managing access control in service composition, they are still insufficient in:

1. ignoring the dynamic nature of composite web services that requires resources based on-demand;
2. missing an efficient way to the administration of the resources in service-oriented authorization;
3. hard coding the roles issued from resources and composite service;
4. lack of authorization rules for preventing conflict of interest in composite web service authorization.

Therefore, an extended RBAC model is required to manage composite web service authorization by taking both authorization policies of component service and composite web service into account. Authorization constraints, e.g. separation of duties, also should be considered within the new model.

2.3.3 Authorization Model in Business Process Management

Another major domain where the RBAC authorization model is implemented a lot is business process, also known as workflow authorization management.

The authors in [73] have proposed a role based access control method through Petri-Net workflows. Role authorization rights are granted according to the state of the workflow. The access control matrices are also deployed at this stage to define the role authorization policies. However, the authorization conflicts and errors regarding to the mutually exclusive role assignments that can cause unreliable business process is missing.

In [74], the authors propose a workflow authorization model (WAM) and an authorization template (AT) to realize the synchronization of authorization flow with

workflow. However, the authorization constraints in terms of business process or workflow should not be limited on synchronization between authorization flow and workflow only. Dependency policy that is used to restrict the authorization in a workflow also should be taken into account, but is completely ignored in [74]. Moreover, the paper uses a Petri-Net based process model to describe the workflow, which can not be used in composite web service authorization. In service composition environment, the access sequence of service consumers and the support sequence of component web services generate two parallel flows that both should be synchronized with the control flow of a business process. The process model in [74] can not be used to expose the authorization constraints between the access sequence and the support sequence.

Authors in [75] extend the above WAM by using Colored Petri-Net to enforce more policies, e.g. separation of duty (SoD) and binding of duty (BoD). However, the important depending policy is still missing in [75]. Furthermore, unfortunately, the Colored Petri-Net based process model is also not suitable to describe the two authorization sequences, i.e. access sequence and support sequence, in composite web service authorization.

In [76], the authors propose a constrained workflow systems where local and global cardinality constraints as well as SoD and BoD are enforced. However, all above authorization models in workflow environment do not take resource into account, and can not be used in web service environment. The authorization dependency policies are also missing in the existing models.

In [77], the authors mainly extend the RBAC by adding an element of *Workflow* which consists of multiple tasks to enforce the separation of duties (SoD) in workflow environment. The conflicted role, conflicted permissions, conflicted users and conflicted tasks are all considered in terms of when the policy of SoD should be enforced to avoid the conflicted authorizations. However, by adopting the authorization methods proposed in [77] in composite web service environment, the question of *how the service consumer can orderly access the tasks (operations) of the workflow based on the business logic without causing any conflict* is answered. But the questions of *how the component service can orderly support the task of the workflow* and *how to coordinate the access of*

service consumer and the support of component service to avoid any conflict of interest are still missing in [77].

The authors in [78, 79] present an extended RBAC model in workflow environment named RWAM (RBAC96 Workflow Authorization Model). Both role hierarchy and constraints are considered with RWAM. The authorization flow is synchronized with the execution of the normal workflow by a temporal policy. However, the tasks within a workflow may be executed flexibly without complying with any strict time table. Hence, this method is hard to manage the synchronization between the authorization flow and the workflow, when the execution time of the tasks in workflow is undecidable.

In [80], the authors propose algorithms to check the consistency of constraints and assign users and roles to tasks that constitute the workflow in such a way that no constraints are violated. Both static and dynamic authorization constraints are expressed as clauses in a logic program. However, the reason that the work presented in [80] is still not suitable for composite web service environment is that the access sequence and support sequence are not simultaneously considered in authorization management. Their coordination on enforcing authorization constraints can be merely expressed in the normal workflow model.

In [81], the authors propose a flexile access control with dynamic checking features for handling workflow changes and exceptions. Temporal RBAC is used to synchronize the authorization flow with the execution of the workflow. Unfortunately, the dependency policy and synchronization policy are missing and not presented in [81]. Moreover, only adopting temporal rules to synchronize the authorization flow with workflow can not effectively manage the changes of the initiation time or finalization time of the tasks within a workflow.

In [82], the authors use an extension for the Business Process Modeling Notation (BPMN) to express authorizations within the workflow model, enabling the support of resource allocation pattern, e.g. SoD, Role based allocation, case handling, or history based allocation. BPMN is a standard for business process modelling that provides a graphical notation for specific business process in a business process diagram, based on a flowcharting technique very similar to activity diagrams from Unified Modeling

Language (UML). BPMN adopts traditional Role Based Access Control mechanism to restrict the authorization, where each organization involved in a business process mainly considers a question, who can access the activities of mine. However, the BPMN based authorization model in workflow environment still does not take component web service into account, and can not be used in composite web service environment. In a composite web service, in addition to thinking about how to prevent unauthorized access on the operations of composite web service, how to access the operations of a component web service should also be taken into account by the composite web service, and the coordinations on these two types of authorization should be concentrated, but is still ignored by BPMN. The authorization synchronization policies and dependency policies are also missing in the model.

In a summary, the existing authorization models in workflow environment, also known as business process, can not be used to express the composite web service authorization lies in,

1. The formal methods, e.g. Petri-Net based process model, BPMN based model or state machine based model, used to describe the business process or workflow can only express access sequence of the service consumer and the workflow. Therefore, only the synchronization between the access sequence of the service consumer and the execution sequence of the workflow can be considered. However, in composite web service authorization environment, the support sequence of the component web service should also be taken into account. Also, with the involvement of the support sequence, the synchronization should not only be between the access sequence and the execution sequence of the workflow, but also be between the support sequence and the execution sequence of the workflow. Moreover, the issue for synchronizing the access sequence with the support sequence becomes another problem that should be tackled in composite web service authorization management.

2. In addition to the authorization synchronization policy, authorization dependency policy is another one that is used to restrict the access of service consumer and

the support of component service. Unfortunately, no research work has designed any approach to express this type of authorization policy. Only popular policies of SoD, BoD and Cardinality are considered in these works. However, in binding policy, if specific access of service consumer is enabled, then the bound access of service consumer must be executed. The two accesses of service consumers are completely bound. But in dependency policy, the depending access of service consumer can be performed only after the depended access of service consumer is executed. But the execution of the depended access of service consumer does not means that the depending access of service consumer must be enabled. It is only the necessary condition that the depending access of service consumer can be enabled, but not the sufficient condition. Although it is a nuance between the policy of BoD and dependency, this type of policy still plays important role in authorization management that attention be paid.

Therefore, a process model should be developed to catch both the sequence of access and the sequence of support as authorization flow. Execution sequence of the workflow should also be described by the process model. Moreover, the authorization synchronization policy and authorization dependency policy ought to be enforced based on the new designed process model.

2.4 Petri-Net

2.4.1 Petri-Net with Variation

Petri-Net is a net theory advanced by Dr. Petri in 1962 [44]. The advantages of its graphically and mathematically founded modeling formalism with various algorithms for design and analysis [45] make it a good candidate for modeling the authorization flow.

Definition 1 *A Petri-Net is a tuple N=(P,T,F), where:*

- *P is a set of places graphically represented as circles.*

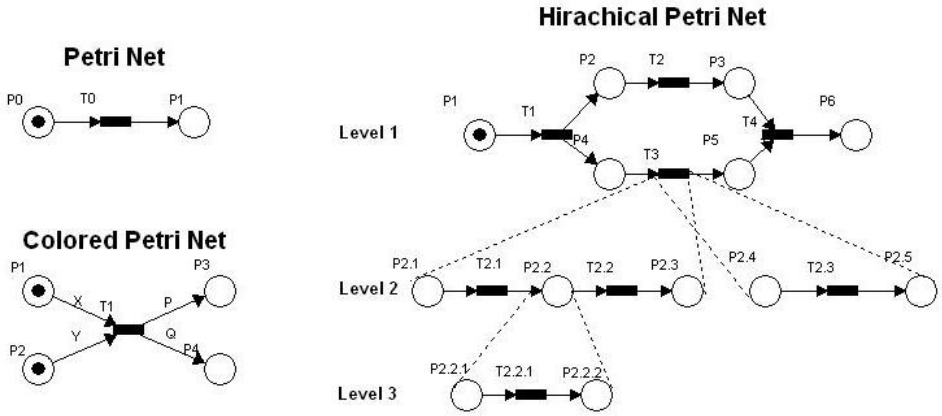

FIGURE 2.6: Various Types of Petri-Nets

- *T is a set of transitions graphically represented as dark bars.* $P \cap T = Null$.
- $F \subseteq \{P \times T\} \cup \{T \times P\}$ *is the flow relation between places and transitions.*

Marking of a Petri-Net is an allocation of tokens to the places of the net formally defined as a function $M: P \rightarrow R^{|P|}$, where $R^{|P|}$ is a $|P| \times 1$ vector with $|p|$ elements. The marking reflects the state of the Petri-Net after each firing. In a marking k, if a token in p, then $M_k(P)$=1, otherwise $M_k(p)$=0. M_0 is the initial Marking of the Petri-Net.

Definition 2 *A Marked Petri-Net is a tuple* $S=(N,M_0)$ *where N is Petri-Net,* M_0 *is the initial marking.*

Through initial marking, we observe that the Petri-Net can reach a series of markings according to the firing of transitions. A transition t is enabled under M written as $M[t>$, if $\bullet t \subseteq M$, where $\bullet t = \{y \in P | (y,x) \in F \cap x \in T\}$. A firing sequence among multiple transitions t_i (i=1..n) can be written as M[t_1>M'[t_2>M"..., where the firing sequence $\sigma = \{t_1, t_2, ...\}$.

Through investigating the marking M, we can observe a series of characteristics of Petri-Net using several analysis tools. Here we introduce two matrixes named *Incident matrix* [46] and *Transitive matrix* [50].

Definition 3 *For the Petri-Net N with n transitions and m places, the incident matrix A=[a_{ij}] is an m × n matrix [46]and its typical entry is given by*

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

where:

$$a_{ij}^{+} = \begin{cases} 1 & (i,j)\ In\ F \\ 0 & (i,j)\ Not\ In\ F \end{cases} \qquad a_{ij}^{-} = \begin{cases} 1 & (j,i)\ In\ F \\ 0 & (j,i)\ Not\ InF \end{cases}$$

$i \in T$ *and* $j \in P$

Definition 4 *a labeled place transitive matrix[50]:*

$$L_{BP} = A^{-} Diag(t_1, t_2, ..., t_n)(A^{+})^{T}$$

where A^{-}=[a_{ij}^{-}] *and* $(A^{+})^{T}$=[a_{ij}^{+}]^{T} *(T represents transpose matrix),* t_i*(i=1,2,...,n) is:*

$$|t_i| = \begin{cases} 1 & fire\ t_i \\ 0 & not\ fire\ t_i \end{cases}$$

Also we use L_{BP}^{*} to extend the original transitive matrix in which t in L_{BP} is replaced by t/d in L_{BP}^{*}, if a transition t appears d times in the same column of L_{BP}.

Colored Petri-Net (CPN) [49] in Fig. 2.6 is an extension of Petri-Net in which tokens are assigned with values. In service interactions, various types of messages can be transferred within or cross organizations. Therefore, the message types can be represented as colored tokens in CPN. Another extension of Petri-Net in Fig. 2.6 is Hierarchical Petri-Net (HPN) [48] in which different views in supporting different levels of abstraction and refinement can be specified. In this THESIS, we call it net refinement or refinement when a transition or place can be represented as one or more HPNs. Timed Petri-Net is also an extension of basic Petri-Net by adding temporal semantics [95, 96]. In the future, we will extend our Petri-Net process model proposed in this thesis by considering the temporal influence on the authorization.

2.4.2 Petri-Net with Verification

In this section, we will illustrate the property of Petri-Net based process model, that is related with our proposed authorization policy verification approaches addressed in

Chapter 6. That is ***Dead Marking Freeness***. Dead Marking Freeness means that all markings have enabled transition. Formally,

$$\forall M_k \in \mathbf{M}, \exists t \in T \qquad M \rightarrow t$$

where $\rightarrow$ is the performance of transition enabling.

The Petri-Net provides a formal approach on verifying *Dead Marking* by using transitive matrix [97, 98]. If the net is dead marking freeness, then for each marking there exists transition t_h (h=1..n) to be fired. The labeled transitive matrix L^*_{BP} illustrates the relation of ${}^\bullet t_h$ (pre-places of transition t_h) and $t_h{}^\bullet$ (post-places of transition t_h) according to the specific fired transition t_h. We set a *Reachable Marking* $M^R_k = M_{k-1} \cdot L^*_{BP}$ to indicate which transition will be fired from state K-1 to state K. Obviously, $M^R_k \neq M_k$ in that M^R_k is used to analyze which transition is fired during the process rather than illustrating the status of the whole net after k^{th} firing, although M^R_k can be easily transformed to M_k.

The *transitive matrix* L^*_{BP} [50] is an m×m matrix, where m denotes the amounts of the places in a specific Petri-Net based process model. A transition t will be the element of the *transitive matrix* L^*_{BP} at row i and column j, if place i is the pre-place of the transition and place j is the post-place of the transition. Through the *transitive matrix* L_{BP}, the structure of specific Petri-Net based process model is transferred into a form of matrix.

The *labeled transitive matrix* L^*_{BP} (m×m) is extended from the original transitive matrix L_{BP} in which t_h in L^*_{BP} is replaced by t_h/d in L^*_{BP}, if t_h appears d times in the same column of L^*_{BP}. It means that, if a transition t_h has d pre-places, then to enable the transition in time, tokens must be able to arrive all d pre-places in time and each token provides 1/d strength to support the enable of the transition t_h.

Marking M is a 1×m matrix to represent each step of the movement of the token in the process model. An element in a specific marking is equal to 1 at row 1 and column j, if there exists a token at the place j in this step of token movement; otherwise, the element is 0 in the marking. For example, marking M_1=[1, 0, 0] represents that a token is in place 1; while marking M_2=[0, 1, 0] means that a token is moved into place

2 from place 1 after one step token movement ($M_1 \rightarrow M_2$).

The token movement in the Petri-Net based process model could become true if and only if specific transition in the model could be enabled and fired in time to consume tokens from pre-place and deposit tokens in the post-place. However, if the transition can not be enabled and fired, the token movement will be ceased in this transition and is called dead marking. Here we introduce a verification method to see if a Petri-Net based process model is dead marking free at the k step of token movement (k=1 to x, where x represents how many steps are needed for the token to move from initial place to final place in the process model). Before token moves from Marking M_{K-1} to M_K, We use M_{k-1} to be computed with L^*_{BP} to get a temporary 1×m matrix M^*_k, called *Reachable Marking*. In M^*_k, if there exists an element more than or equal to 1 at column y (y could be any column), then we believe that marking M_k could be transferred from marking M_{k-1} where token is moved from specific place in M_{k-1} to place y in M_k, i.e. the Petri-Net based process model is dead marking free. The specific transition can be enabled and fired to support the marking transfer.

2.4.3 Petri-Net with Service Composition

Petri-Net is a technique used widely for business process modeling and verification. We shall look into some of the representative work in the area.

In [83, 94], the authors introduce PNML (Petri-Net Markup Language) to transfer the composite service into Petri-Net model and implement algorithm based on the model to verify the reliability of composition. However, the authors do not take into account the interactions between web services of different organizations. It means that the component service is not included within the Petri-Net model which makes it not suitable for our composite web service authorization management.

In [84], the authors provide method to transfer BPEL4WS [41] to CP-Nets for verifying web service composition. Another modified model based on Hierarchical CPN in [85] is introduced later to detect the reliability issues of web service workflow. A series of net properties are presented, such as reachability, boundness and dead transition etc. However the model based on the Hierarchical CP-Net is constructed from a centralized

global view which includes all the detailed information of participants. This assumption can not be held in the peer-to-peer loosely coupled web service environment.

In [86], the authors construct a verification framework for web service composition. *protocol conformance* as the requirement of business interaction is presented to check the correlation of the complex conversations among multiple organizations. However, verification on authorization policy is totally missing in the process model.

The authors in [87, 88] introduce a property *soundness* for verifying the reliability of WF-Net, a Petri-Net based workflow model. However without the presence of the cross-organizational service interaction, the work can not be used by individual organizations for managing service composition, which requires modeling both service consumer, component service, and their associated interactions with the composite web service.

In [89], the authors link the algebraic metamodel to the Petri-Net representation for describing the service composition. Unfortunately, the authors only present a mathematical formalism to describe the coarse-grained service composition without any introduction on the external component service support on the composite web service.

The WS-Net [90] (Web Service Net) describes the web service components in three levels to simulate the business interactions among service composition. However, in WS-Net, both service consumers and component services that are needed to interact with composite web service are set up in advance, which is not suitable for the features of service composition. In service composition, the service consumers and component service are loosely coupled with the composite web service. It is not realistic to identify which service consumer can access the composite web service and which component service can support the composite web service, in advance.

In a summary, the existing Petri-Net based process model can not be used to model and verify the authorization management in composite web service lies in,

1. No petri-net based process model can describe both the sequence of access of service consumer and the sequence of support of component web service, as well as the execution sequence of the composite web service.

2. No verification approaches have been proposed to detect the improper authorization policy definitions.

Hence, a new Petri-Net based process model is needed based upon the proposed conceptual model in this thesis. From next chapter, we will start to introduce the core part of the thesis, where the conceptual model, the process model, the authorization policy based on the two models, and the authorization policy verification method are presented in details.

3

The Conceptual Model-SOAC

3.1 Introduction

In this chapter, we will mainly focus on presenting the conceptual model of service oriented authorization control (SOAC), which can be used to manage the composite web service authorization. In web service environment, a web service can compose multiple component web services in a loosely-coupled environment. Web service technologies provide a technical foundation for seamlessly composing individual component web services into a cohesive one [51]. However, how to manage the access on the composite web service becomes a challenge in loosely-coupled environment. The research problems are identified as follows:

1. **Complicated Coordination on Authorization Policies**: Each web service, e.g. financial lease service or funder in motivating scenario in Chapter 1, bears

specific authorization policies to restrict the access on its operations. The *Financial Lease* service is a composite web service and its operations are supported by multiple component web services provided by other business partners. It is not enough to enforce the authorization policies of the *Financial Lease* service without considering the characteristics of these component services. For example, in Fig. 1.2 in Chapter 1, the operation *Lease Application* is supported by the component web services A, C and D from other organizations that can provide fund and product. Hence, without properly handling the authorization policies of the component web services, the *Financial Lease* service can not ensure if the authorization to access the specific *Financial Lease*'s operations can be supported. For instance, if the authorization to access the *Lease Application* operation has been assigned to a specific service consumer, but *Financial Lease* service fails to obtain the authorization from the other component web services, then the operation can not be enacted and the unnecessary disclosure of *Lease Application* operation occurs. This is the result of lack of coordination on the authorization policies in *Financial Lease* service and its supporting component web services. Therefore, granting the access on the *Financial Lease* service to a service consumer needs to consider not only the service consumers but also the component services. We should not only understand "`who should do what`?", but also need to know "`who should do what under what kind of support`".

2. **Dynamicity of Component Service and Service Consumer**: Web services are autonomous and interact with each other in a loosely-coupled environment. Many web services compose component services in a highly dynamic manner. For example, if a component service changes its authorization constraints from asking *Financial Lease* service for professional lease certificate to requiring loan qualification, then all the service operations in *Financial Lease* service that can be supported by the component web service need to update their identifications on the component service's authorization constraints. Moreover, there are huge amounts of web services that can provide the same or similar operations. For

example, in Fig. 1.2 in Chapter 1, component services `Funders` 1 to 3, and `Lease Agents` 1 to 3 can all provide financial support to *Financial Lease* service. Hence, if the changes occur frequently and/or happen in thousands of web services, then an efficient way to administrate these changes is needed. At this stage, the dynamicity of web service impedes the *efficiency* of service-oriented authorization management.

Therefore, efficient management on service-oriented authorization by coordinating the policies in different web services is needed. Role Based Access Control (RBAC) [52] is a widely accepted approach to restrict system access to authorized users. In RBAC, users acquire permissions through their roles rather than they are assigned permissions directly. This greatly reduces the administrative overhead associated with individual user and permission. However, web services technologies facilitate the integration of the loosely-coupled distributed applications. There may be a large amount of component web services which are used as resources to support the composite web service's operations. The quantity of service consumers can also be large. As illustrated in Fig. 1.2 in Chapter 1, the system of the composite web service (*Financial Lease Service*) needs to deal with not only a large number of users (service consumers) but also great amounts of resources (component services). Moreover, the characteristics of these resources are different from those of objects of RBAC in close systems. Hence, traditional RBAC is not suitable for the service-oriented authorization management since it has not taken the *Resource* into account. Note the *Resource* particularly can not be fully controlled by composite web service in the context of service environment.

All existing role-based models in web service paradigm have not brought the administration of resource into the picture. Actually, the quantity of resources can be very large and they can be prone-to-change, which should be considered in web service authorization. In research work [60, 61, 70], roles are assigned to service consumers for service authorization. However all these researches have not put the resource into the picture or they simply employ an unrealistic assumption that there is a *global* coordination on internal authorization policies of each autonomous web service to enforce the access control in service composition.

In this chapter, we propose an innovative conceptual model for authorization of web services, named *Service Oriented Authorization Control (SOAC)*. The conceptual model will handle the web service authorization by dealing with not only a large number of service consumers, but also huge amounts of resources. Authorization policies of the component web service and the supported composite web service are integrated together for making authorization decision to a service consumer. Furthermore, administrative functions are also presented to enforce the web service authorization from a system perspective.

The rest of chapter is organized as follows. Section 2 describes the conceptual model with the major features and detailed specifications. Section 3 discusses the administrative functions. Concluding remarks is presented in Section 4.

3.2 Conceptual Model of Service Oriented Authorization Control

In this section, we describe the conceptual model SOAC in details. *Service Oriented Authorization Control (SOAC)* is used for managing the authorization of composite web service. SOAC is divided into two parts, *service provision* and *service realization* (See Fig. 3.1). We express the SOAC conceptual model by using the notation of **Entity-Relationship** (**E-R**) **Diagram**. In Fig. 3.1, rectangles represent elements and diamonds represent relationships. The functions and relationships defined in Definitions 5 to 7 are used to describe the different parts of SOAC and are mainly used in Function Specification of SOAC in section 3.3.

3.2.1 *Service Provision* Specification

In *service provision*, a service consumer can get the authorization by fulfilling constraints of the composite service (See *Constraint* enacted between the elements of Role (**R**) and Service Consumer (**SC**) in Fig. 3.1). In Fig. 3.1, we define service consumer as the element that requires to access the composite web service's operations (**Op**).

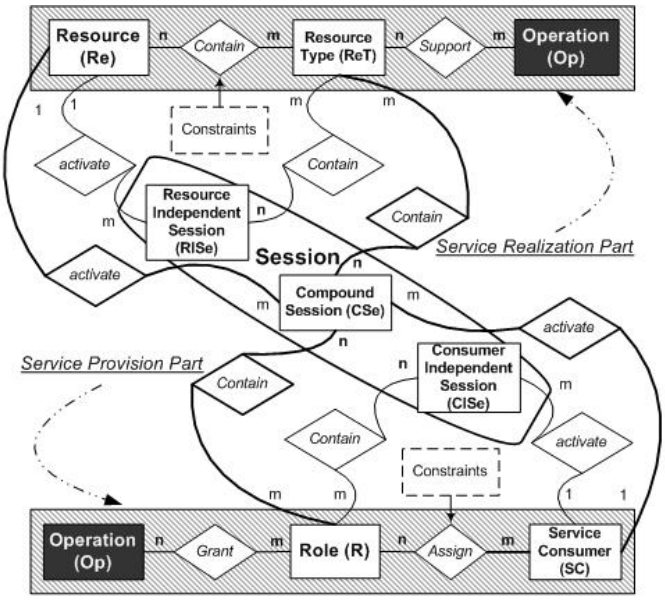

Figure 3.1: Service Oriented Authorization Control (SOAC) Conceptual Model

Since service consumers are prone to change and the quantity of consumers can be vary large, directly specifying the assignment of operations to individual service consumers needs tedious administration efforts. In SOAC, we follow the philosophy of RBAC to have the concept role to encapsulate the service consumers that can satisfy the authorization constraints of composite web services. A role will be assigned to a service consumer based on its characteristics (typically a credential that a service consumer submits to the composite web service). Each role binds with a group of operations that can be accessed. The roles guarantee that the composite web service's operations can only be accessed by the qualified service consumers. The mappings between service consumers and roles are considered in the *service provision* part of SOAC with the following formal specification.

Definition 5 *The service provision in SOAC includes:*

- $\boldsymbol{SC}$, $\boldsymbol{R}$ *,and* $\boldsymbol{Op}$ *are elements representing Service Consumer, Role, and Operation.*
- $\mathbf{SCA} \subseteq \boldsymbol{SC} \times \boldsymbol{R}$*, a many-to-many relationship to map service consumer to role assignment. Formally,* $\forall s^c \in SC, \forall r \in R, (s^c, r) \in SCA \Rightarrow s^c.credential = r.credential$,

where the credential that the service consumer submits is consistent with the credential that the role requires.

- ***assigned_sc:(r:R)*** $\rightarrow 2^{SC}$, *the mapping of role r onto a set of service consumers. Formally,* $assigned\_sc(r)=\{s^c \in SC | (s^c, r) \in SCA\}$.

- **OPA** $\subseteq \boldsymbol{Op} \times \boldsymbol{R}$, *a many-to-many relationship to map operation to role assignment.*

- ***assigned_op:(r:R)*** $\rightarrow 2^{Op}$, *the mapping of role r onto a set of operations. Formally,* $assigned\_op(r)=\{op \in Op \mid (op, r) \in OPA\}$.

3.2.2 *Service Realization* Specification

Due to the feature of **Dynamicity** of resources, it is unrealistic to specify the relationships between resources and the supported operations of composite web services individually. Resource type is defined for a set of resources by identifying their characteristics and authorization constraints (See Fig.3.1). The composite web service can bear multiple resource types that cover many resources. The resources can be accessed to support the operation if the operation is mapped with a resource type that covers these resources. Resources are linked with resource types with constraints. (See *Constraint* between the elements of Resource Type (**ReT**) and Resources (**Re**) in Fig. 3.1). The mappings between resources and resource types are the major concerns in *service realization* part of SOAC. Notes, based on its characteristics, a resource can belong to multiple resource types. The formal specification of *service realization* is as follows.

Definition 6 *The service realization in SOAC includes:*

- ***Op***, ***ReT***, *and* ***Re*** *are elements representing Operation, Resource Type, and Resource.*

- **SPA** $\subseteq$ ***Op*** $\times$ ***ReT***, *a many-to-many relationship to map operation to resource type.*

- ***assigned_ret:(ret:ReT)*** $\rightarrow 2^{Op}$, *the mapping of resource type ret onto a set of operations. Formally, assigned_ret(ret)=*$\{op \in Op | (op, ret) \in SPA\}$.

- **RTA** $\subseteq$***Re*** $\times$***ReT***, *a many-to-many relationship to map resource to resource type. Formally,* $\forall re \in Re$, $\forall ret \in ReT$, $(re, ret) \in RTA \Rightarrow$ re.constraint = ret.constraint, *where the constraint that restricts the access on the resource is consistent with the constraint that the resource type can fulfill.*

- ***assigned_re:(ret:ReT)*** $\rightarrow 2^{Re}$, *the mapping of resource type ret onto a set of resources. Formally, assigned_re(ret)=*$\{re \in Re | (re, ret) \in RTA\}$.

3.2.3 Integration of Service Provision and Service Realization

Service provision and *service realization* in SOAC must be considered together for authorization of composite web services. In Fig. 3.1, the mappings between the elements of Role (**R**), Operation (**Op**), and Recourse Type (**ReT**) integrate the *service provision* and *service realization*. (For the concision of Fig. 3.1, the element of Operation is diagrammatically separated in service realization part and service provision part.) The access to the composite web service can be assigned to a service consumer if all the constraints of the composite web service and its resources can be satisfied. In *service provision*, the service consumer is assigned a specific role for the access to the operations; while in *service realization*, the operations are mapped with resource types that cover all resources required.

In order to check conflict of interest at runtime, element **Session** is introduced at integration of *service provision* and *service realization* in SOAC (see Fig. 3.1). There are two layers of sessions, Independent Layer and Compound Layer. In independent layer, *Consumer Independent Session* (**CISe**) is used to check runtime conflict of interest in *service provision*; while *Resource Independent Session* (**RISe**) is used to check runtime conflict of interest in *service realization*.

After a service consumer starts to send a message to the composite web service for accessing its operations, the service consumer activates the assigned specific roles in a consumer independent session. Without receiving message from service consumer, the

composite web service can also activate specific resource type for the resource. In some circumstance, without knowing if the service consumer will use the assigned role to send a message, the composite service can pass some relevant information for activating the resource type to prepare the support for the composite service in advance. A resource independent session is generated in such situation.

In the compound layer, a compound session is established when the message from service consumer is transferred to the resource. In this case, specific resource type is activated by the resource as well as the role is activated by the service consumer in a compound session, i.e., the service consumer with an assigned specific role is accessing the operation of composite web service that is being supported by the resource which is included in the activated resource type. Below is the formal definition of **Session**.

Definition 7 ***Session*** *includes two layers, Independent Layer and Compound Layer. In independent layer, two sessions, Consumer Independent Session (CISe) and Resource Independent Session (RISe), are included; while in Compound Layer, a session named Compound Session (CSe) is included. They are formally presented as follows:*

- ***Consumer Independent Session (CISe)*** *is a session for the mapping of a service consumer* s^c *on a set of activated roles* $\{r_1..r_j\}$, $j \geq 1$.
- ***Resource Independent Session (RISe)*** *is a session for the mapping of a resource* re *on a set of activated resource types* $\{ret_1..ret_k\}$, $k \geq 1$.
- ***Compound Session (CSe)*** *is a sessioin for the mapping of a pair of service consumer and resource* $< s^c, re >$ *on a set of activated roles and resource types* $\{< r_1, ret_1 >..< r_j, ret_k >\}$, *(Note, the operations that the service consumer* s^c *requires to access are the same operations that the resource* re *can provide support to.) where:*
  - $r_1..r_j$, $j \geq 1$, *is a subset of roles assigned to and activated by the specific service consumer* s^c.
  - $ret_1..ret_k$, $k \geq 1$ *is a subset of resource types assigned and activated by the specific resource.*

- ***sc_cise:***$(s^c$***:SC***$)\rightarrow 2^{CISe}$*, the mapping of service consumer* s^c *onto a set of consumer independent sessions CISe.*

- ***cise_r:(***se_{ci}***:CISe)***$\rightarrow 2^R$*, the mapping of consumer independent session* se_{ci} *onto a set of roles.*

- ***re_rise:***$(re$***:Re***$)\rightarrow 2^{RISe}$*, the mapping of resource re onto a set of resource independent sessions RISe.*

- ***rise_ret:(***se_{ri}***:RISe)***$\rightarrow 2^{ReT}$*, the mapping of resource independent session* se_{ri} *onto a set of resource types.*

- ***sc_cse:***$(s^c$***:SC)***$\rightarrow 2^{CSe}$*, the mapping of service consumer* s^c *onto a set of compound sessions CSe.*

- ***cse_r*** $(se_c$***:CSe)***$\rightarrow 2^R$*, the mapping of compound session* se_c *onto a set of roles.*

- ***re_cse(***re***:Re)***$\rightarrow 2^{CSe}$*, the mapping of resource re onto a set of compound session CSe.*

- ***cse_ret(***se_c***:CSe)*** $\rightarrow 2^{ReT}$*, the mapping of compound session* se_c *onto a set of resource types.*

3.3 Function Specifications of SOAC

In this section, we present the function specifications of SOAC. These functions provide the facilities for maintaining the SOAC conceptual model components, e.g., the element sets and relations. The **Session** is also taken into consideration to manage the activations of the specific elements. Three categories of functions are presented: (1) Service Provision Administrative Operation ($\mathcal{SP} - \mathcal{AO}$); (2) Service Realization Administrative Operation ($\mathcal{SR} - \mathcal{AO}$); and (3) Session Operation ($\mathcal{SE} - \mathcal{O}$). These functions will be used as administrative functions in implementation tool in Chapter 7. The notation used to formalize the operations is a subset of the **Z** notations. The representation schema in the formal specification of the operations is:

TABLE 3.1: Service Provision Administrative Operation - $\mathcal{SP} - \mathcal{AO}$

$\mathcal{SP} - \mathcal{AO}_{ele}$
(1) `AddServiceConsumer(serviceConsumer:NAME)`
(2) `DeleteServiceConsumer(serviceConsumer:NAME)`
(3) `AddRole(role:NAME)`
(4) `DeleteRole(role:NAME)`
(5) `AddOperation(operation:NAME)`
(6) `DeleteOperation(operation:NAME)`
$\mathcal{SP} - \mathcal{AO}_{rela}$
(7) `AssignServiceConsumer(serviceConsumer,role:NAME)`
(8) `DeassignServiceConsumer(serviceConsumer,role:NAME)`
(9) `GrantOperationAccess(operation,role:NAME)`
(10) `RevokeOperationAccess(operation,role:NAME)`

$$SchemaName(Declaration) \ \triangleleft \ Predicate; ...; Predicate \ \triangleright .$$

All the data types and operations used in the formal specification have been defined in SOAC described in previous sections of this paper. *NAME* is an abstract data type used to represent the identifiers of the elements in SOAC.

3.3.1 Service Provision Administrative Operation- $\mathcal{SP} - \mathcal{AO}$

There are three elements in *service provision* part of SOAC-Service Consumer (**SC**), Role (**R**), and Operation (**Op**), and two relationships among the elements-*Assign* and *Grant* (See grey rectangle in Fig. 3.1). Hence, $\mathcal{SP} - \mathcal{AO}$ are separated into two aspects $\mathcal{SP} - \mathcal{AO}_{ele}$ and $\mathcal{SP} - \mathcal{AO}_{rela}$ to manage the elements and relationships respectively.

The operations (1)∼(6) (See Table. 3.1) are $\mathcal{SP} - \mathcal{AO}_{ele}$ used to add and delete elements in service provision part. Due to space limit, we only describe the operation (3) and (4). Let us take operation (4) as example. In operation (4), an existing role is

Table 3.2: Service Realization Administrative Operation - $\mathcal{SR} - \mathcal{AO}$

$\mathcal{SR} - \mathcal{AO}_{ele}$
(11) AddResourceType(resourceType:NAME))
(12) DeleteResourceType(resourceType:NAME)
(13) AddResource(resource:NAME)
(14) DeleteResource(resource:NAME)
$\mathcal{SR} - \mathcal{AO}_{rela}$
(15) SupportOperation(operation,resourceType:NAME)
(16) AbandonOperation(operation,resourceType:NAME)
(17) ContainResource(resource,resourceType:NAME)
(18) ExcludeResource(resource,resourceType:NAME)

deleted. If the role is activated by the specific service consumer in independent session or compound session, then the associated session should be deleted firstly. Furthermore, the mapping functions and relations associated with the deleted role also need to update to erase the effect of this role. Then the role can be deleted from the set **R**.

(3) AddRole(role:NAME)◁

role$\notin$R; R'=R$\cup${role};

assigned_sc'=assigned_sc$\cup${role$\mapsto \emptyset$};

assigned_op'=assigned_op$\cup${role$\mapsto \emptyset$};▷

(4) DeleteRole(role:NAME)◁

role$\in$R;

[$\forall$ se_{ci} $\in$CISe • role $\in$ cise_r(se_{ci})$\Rightarrow$

DeleteConsumerIndependentSession(se_{ci})];

[$\forall$ se_c $\in$CSe • role $\in$ cse_r(se_c)$\Rightarrow$

DeleteCompoundSession(se_c)];

SCA'=SCA\{sc:SC • sc$\mapsto$ role};

assigned_sc'=assigned_sc\{role↦assigned_sc(role)};

OPA'=OPA\{op:Op • op ↦ role};

assigned_op'=assigned_op\{role↦assigned_op(role)};

R'=R\{role};▷

The operations (7)∼(10) (See Table. 3.1) are $\mathcal{SP}-\mathcal{AO}_{rela}$ used to add and delete relationships in *service provision* part. For example, Operation (7) is used to create the relationships between the elements of Role (**R**) and Service Consumer (**SC**).

(7) AssignServiceConsumer(serviceConsumer, role:NAME)◁

serviceConsumer∈SC; role∈R;

(serviceConsumer↦r)∉SCA;

SCA'=SCA∪{serviceConsumer↦r};

assigned_sc'=assiged_sc\{role↦assigned_sc(role)}∪

{role↦(assigned_sc(role)∪serviceConsumer)}; ▷

(8) DeassignServiceConsumer(serviceConsumer,role:NAME)◁

serviceConsumer∈SC; role∈R;

(serviceConsumer↦role)∈SCA;

[∀se_{ci}:CISe • se_{ci}∈sc_cise(serviceConsumer)∧

role∈cise_r(se_{ci})⇒DeleteConsumerIndependentSession(se_{ci})];

[∀se_c:CSe • se_c∈sc_cse(serviceConsumer)∧

role∈cse_r(se_c)⇒DeleteCompoundSession(se_c)];

SCA'=SCA\serviceConsumer↦role;

assigned_sc'=assigned_sc\{role↦assigned_sc(role)}∪{role↦(assigned_sc(role)\

{serviceConsumer})};▷

3.3.2 Service Realization Administrative Operation- $\mathcal{SR}-\mathcal{AO}$

There are also three elements included in $\mathcal{SR}-\mathcal{AO}$- Operation (**Op**), and Resource Type (**ReT**), and Resource (**Re**). However, the element of Operation has already mentioned in previous sub section. We only introduce the left two elements in $\mathcal{SR}-\mathcal{AO}_{ele}$. Moreover, two relationships-*Support* and *Contain* are included in $\mathcal{SR}-\mathcal{AO}_{rela}$ (See white rectangle in Fig. 3.1).

Operations (11)∼(14) (See Table. 3.2) are used to create and delete elements of resource type and resource. They follow the same rules enacted in $\mathcal{SP}-\mathcal{AO}_{ele}$. Operations (15)∼(18) (See Table. 3.2) aim to create and delete the relationships-*Support* and *Contain* in *service realization* part by using the same rules in $\mathcal{SP}-\mathcal{AO}_{rela}$. Operation (11) and (12) are used to add and delete resource type respectively.

(11) AddResourceType(resourceType:NAME)◁
resourceType∉ReT;
ReT'=ReT∪{resourceType};
assigned_ret'=assigned_ret∪{resourceType↦ ∅};
assigned_re'=assigned_re∪{resourceType↦ ∅}; ▷

(12) DeleteResourceType(resourceType:NAME)◁
resourceType∈ReT
[∀ se_{ri}∈RISe • resourceType∈rise_ret(se_{ri})⇒
DeleteResourceIndependentSession(se_{ri})];
[∀ se_c∈CSe • resourceType∈cse_ret(se_c)⇒
DeleteCompoundSession(se_c)];
SPA'=SPA\{Operation:op • opeartion↦resourceType};
assigned_ret'=assigned_ret\
{resourceType↦assigned_ret(resourceType)};
RTA'=RTA\{resource:Re • resource↦resourceType};
assigned_re'=assigned_re\
{resourceType↦assigned_re(resourceType)};

ReT'=ReT\{resourceType}; ▷

3.3.3 Session Operation- $\mathcal{SE}-\mathcal{O}$

$\mathcal{SE}-\mathcal{O}$ is used to maintain the element of **Session** and associated relationships, which reflects the activation of the corresponding elements. Six commands are developed in $\mathcal{SE}-\mathcal{O}_{ele}$ based on the creation and deletion of CISe, RISe, and CSe (See Operations (19)∼(24) in Table. 3.3). The operations (25)-(38) are $\mathcal{SE}-\mathcal{O}_{rela}$ used to maintain the relationships associated with the element of session. Due to space limit, we do not list the operations in Table 3.3. Let us take the operation (21) as an example to illustrate the $\mathcal{SE}-\mathcal{O}$. Firstly, an *ars* is created which represents the activated roles and resource types included in this new compound session. The mapping functions *sc_cse(serviceConsumer)* and *re_cse(resource)* are also updated to reflect the new compound session. Finally, the *cse_r*(se_c) and *cse_ret*(se_c) are updated to point to the activated roles and resource types from the compound session se_c. Operation (22) is used to delete compound session.

(21) CreateCompoundSession(serviceConsumer,resource,operation:NAME;
ars:2^{NAME};se_c:CSe)◁
serviceConsumer∈SC; operation∈Op;
ars⊆{(role:R, resourceType:ReT)|(serviceConsumer↦role)∈SCA,
(Resource↦resourceType)∈RTA};
se_c∉CSe; CSe'=CSe∪{se_c};
sc_cse'=sc_cse\{serviceConsumer↦sc_cse(serviceConsumer)}∪
{serviceConsumer↦(sc_cse(serviceConsumer)∪{se_c})};
re_cse'=re_cse\ {resource↦re_cse(resource)}∪
{resource↦(re_cse(resource)∪{se_c})};
cse_r'=cse_r∪{se_c↦ars}; cse_ret'=cse_ret∪{se_c↦ars};▷

Table 3.3: Session Operation - $\mathcal{SE} - \mathcal{O}$

$\mathcal{SE} - \mathcal{O}_{ele}$
(19) CreateConsumerIndependentSession (serviceConsumer:NAME;ars:2^{NAME};se_{ci}:NAME)
(20) DeleteConsumerIndependentSession (serviceConsumer,se_{ci}:NAME)
(21) CreateResourceIndependentSession (resource:NAME;ars:2^{NAME};se_{ri}:NAME)
(22) DeleteResourceIndependentSession (resource,se_{ri}:NAME)
(23) CreateCompoundSession (serviceConsumer,resource,operation:NAME; ars:2^{NAME};se_c:NAME)
(24) DeleteCompoundSession (serviceConsumer,resource,operation,se_c:NAME)
$\mathcal{SE} - \mathcal{O}_{rela}$
(25) ActivateServiceConsumerCISe (serviceConsumer,se_{Ci}:NAME)
(26) ActivaterResourceRISe(resource,se_{ri}:NAME)
(27) ActivateCSe(serviceConsumer,resource,se_c:NAME)
(28) DeactivateServiceConsumerISe (serviceConsumer,se_{ci}:NAME)
(29) DeactivateResourceRISe(resource,se_{ri}:NAME)
(30) DeactivateCSe(serviceConsumer,Resource,se_c:NAME)
(31) ContainRoleCISe(role,se_{ci}:NAME)
(32) ContainRoleCSe(role, se_c:NAME)
(23) ContainResourceTypeRISe(resourceType,se_{ri}:NAME)
(34) ContainResourceTypeCSe(resourceType,se_c:NAME)
(35) ExcludeRoleCISe(role,se_{ci}:NAME)
(36) ExcludeRoleCSe(role,se_c:NAME)
(37) ExcludeResourceTypeRISe(resourceType,se_{ri}:NAME)
(38) ExcludeResourceTypeCSe(resourceType,se_c:NAME)

(22) DeleteCompoundSession(serviceConsumer,resource,
operation,se_c:NAME)◁
serviceConsumer∈SC;se_c∈CSe;
operation∈Op;
se_c∈sc_cse(serviceConsumer);
se_c∈re_cse(resource);
sc_cse'=sc_cse\{serviceConsumer↦sc_cse(serviceConsumer)}∪
{serviceConsumer↦(sc_cse(serviceConsumer)\{se_c})};
re_cse'=re_cse\ {resource↦re_cse(resource)}∪
{resource↦(re_cse(resource)\{se_c})};
cse_r'=cse_r\{se_c↦cse_r(se_c)};
cse_ret'=cse_ret\{se_c↦cse_ret(se_c)};
CSe'=CSe\{se_c}; ▷

3.4 Conclusion

Although plenty of existing enhanced RBAC mechanisms and approaches have been presented which focus on managing access control in service composition, they are still insufficient in: (1) missing the administration of the resources in service-oriented authorization and (2) ignoring the dynamicity of service environment where the composite web service composes resources based on-demand.

Research work in this chapter provides an extension of classical RBAC approach with the capability to address the authorization issues of the composite web service, brought in by the large number of dynamic service consumers and component web services. A novel conceptual model has been developed for managing the service-oriented authorization in the loosely-coupled environment of web services. Beyond the existing approaches for web services authorization, our approach considers the authorization constraints of the component web services explicitly. The merits of SOAC lie in: (1) the coordination of the authorization constraints of composite web services and component web services; and (2) the introduction of the more constant concepts

of *role* and *resource type* to represent the concepts of service consumer and resource which are dynamic and volatile. In this chapter, three categories of administrative functions of SOAC are also proposed.

In the next chapter, we will develop mechanisms to elaborate the causes and solutions regarding to conflicts of interest in web service authorization. Detailed and formalized authorization policies in terms of conflict of interest in composite web service authorization based on SOAC will also be studied.

4

Conflict of Interest

4.1 Introduction

Conflict is the struggle of persons, where two or more persons oppose each other in interaction and may cause the personal interest insufficient. It is a situation in which a person has a private or personal interest sufficient to appear to influence the objective exercise of his or her official duties as a public official, an employee or a professional. A conflict of interest occurs when an individual is involved in multiple interests, one of which could possibly corrupt the motivation for an act in the other. However, conflict of interest also happens between the owners of web services and this finally becomes the conflict of interest between services when the owners are not transparent in web service environment. Conflict of Interest is a main concern when authorizing permissions to the entity that requires access right in composite web service environment. Conflict of

Interest is important and yet challenging in managing authorization of composite web service, especially when the backend resources (component services) are taken into the picture.

A service consumer or a resource can be involved in multiple interests regarding to the authorization granted by the composite web service. For example, when a web service plays as both `Credit Assessor` and `Customer`, the conflict of interest occurs between *Financial Lease* service and the web service. The interest of Financial Lease service is to decide the lease application under a fair and correct evaluation of the credit history of the `Customer`. The web service as both `Credit Accessor` and `Customer` can modify the credit history report of itself to catering for the credit requirements of Financial Lease, which makes it generate a wrong decision on the lease. The interest of Financial Lease service is broken by the web service as both `Credit Assessor` and `Customer`. To avoid the conflict of interest, the web service can not be authorized for the lease application if itself provides the "evaluate" support. It is necessary to specify authorization rules to detect and control a various types of the conflict of interest in composite web service.

Conflict of interest in terms of authorization in composite web service can be categorized into two parts. In former one, the conflict of interest is examined between the two elements with the same type in SOAC. For example, we will check the relationship of two service consumers, two resource, or two roles. We define the relationships of two elements as *exclusive* and *non-exclusive*, and furthermore, require the same type of element in different authorizations should keep the same type of relationships. Otherwise, conflict of interest between different elements is occurred. Let us take an example in Fig. 1.2, the operations of *Lease Application* and *Guarantor Confirmation* have been defined as Exclusive relationship. Hence, the roles of `Customer` and `Guarantor` that are mapped to the operations of *Lease Application* and *Guarantor Confirmation* respectively should have the relationship of Exclusive as well. However, if two web services who share the interest are used to mapped to `Customer` and `Guarantor` respectively, then conflict of interest occurs as these two web services bear non-exclusive relationship which is not consistent with the relationship of mapped roles.

In former one, in order to check the conflict of interest, we have to expose the relationships between service consumers or between resources as exclusive or non-exclusive. However, in web service environment, due to the feature of autonomy of web service, the relationships of the web service to other services may not be easily detected or advised. For example, sometimes, it is hard to decide if two web services share the interests. Hence, in the later classification of management on conflict of interest, we focus on analyzing the conflict of interest for one service consumer and one resource on different authorizations. It can ensure that all authorizations for one service consumer or one resource, even for one group of service consumer and resource, are conflict-free.

Both types of management on conflict of interest will be presented at design time and runtime, and classified into four categories according to in which part of service composition they can occur, e.g in service provision part, in service realization part, or at integration part of service provision and service realization.

In this chapter, section 2 will present the first type of conflict of interest between different elements. Section 3 will introduce the second type of conflict of interest for one element. Conclusion will be in the last section.

4.2 Management of Conflict of Interest

In this section, we will examine the conflict of interest for different elements with the same type. Four types of **Conflicts of Interest** are identified based on SOAC. Authorization rules are defined to prevent the various types of conflict of interest in this category at both design time and run time.

The relationships between two elements with the same type in SOAC are defined as *Exclusive* $\otimes$ or *Non-exclusive* $\ominus$. *Exclusive* relationship means that two elements of SOAC, e.g., two service consumers, two roles, or two operations, are ostracized each other; while *Non-exclusive* relationship means that two elements of SOAC are not ostracized each other. The relationship between elements with the same type in different authorizations should be the same; Otherwise, conflict of interest will occur.

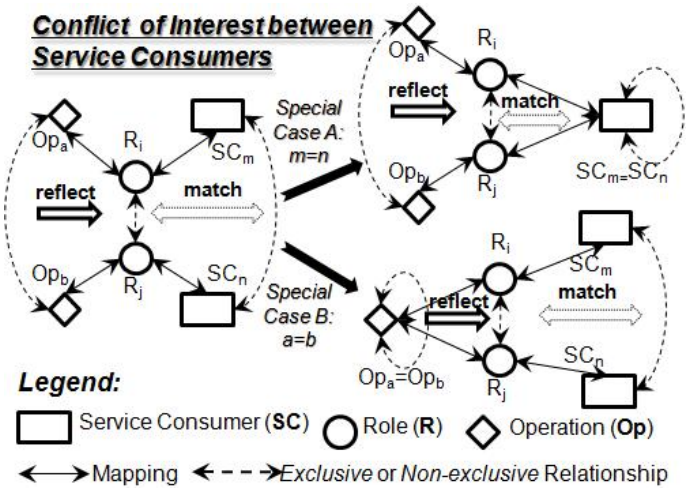

FIGURE 4.1: Conflict-Free Role Relationship Check (CF-R-RC)

4.2.1 Conflict of Interest between Service Consumers

In *service provision*, the relationship between two service consumers should be the same as the relationship between the assigned roles for these two consumers to prevent conflict of interest. In Fig. 4.1, if Op_a and Op_b are *exclusive* ($Op_a \otimes Op_b$), then the relationship between R_i and R_j that are mapped to the operations Op_a and Op_b respectively should reflect the *exclusive* relationship ($Op_a \otimes Op_b \Rightarrow R_i \otimes R_j$). The relationship between assigned roles for service consumers SC_n and SC_m must be matched with the relationship between these two consumers. If service consumers SC_n and SC_m are *non-exclusive* with each other ($SC_n \ominus SC_m$), then SC_n and SC_m can not be assigned roles R_i and R_j respectively at the same time because the roles have the *exclusive* relationship.

Two special cases are illustrated in Fig. 4.1, where (1) two service consumers become the same one in special case A, and (2) two operations become the same one in special case B. Moreover, the relationship between the element and itself can be *non-exclusive* or *exclusive* according to its situation.

For example, in motivating scenario, if we introduce a new operation named *Payment Verification* after the operation of *Monthly Bill*, that is used to verify the monthly

rental from `Customer`, the operations of *Monthly Bill* and *Payment Verification* are *exclusive* operations that need to be mapped to different roles, and such roles are recognized as *exclusive* roles as `Customer/Guarantor` and `Verifier`. If a service consumer is assigned with both `Customer/Guarantor` and `Verifier` (Special case A in Fig.4.1), the conflict of interest will occur, since `Customer/Guarantor` and `Verifier` must have relationship-*Exclusive* for accessing *exclusive* operations.

Let us take another example. "Double Check" policy is enforced in Financial Lease service on the operation of *Payment Verification*, i.e., two financial institutes are required to ensure the monthly rental from `Customer`. In order to avoid fraudulent payment assessment on rental payment, an *exclusive* relationship between the `Initial Verifier` and `Second Verifier` must be enforced. Westpac Financial Service and St. George Financial Consulting Company are two financial institutes with *non-exclusive* relationship because they belong to the same financial group. Westpac Financial Service and St. George Financial Consulting Company can not be assigned the roles `Initial Verifier` and `Second Verifier` to do payment verification for one transaction due to their *non-exclusive* relationship.

To prevent conflict of interest among service consumers, the following authorization rule named *Static Conflict-Free Role Relationship Check* (**S-CF-R-RC**) is specified as follows:

Authorization Rule 1 S-CF-R-RC: *Let* $\mathbb{SC}$ *be a set of Service Consumers. Let* $\mathbb{R}$ *be a set of Roles. There is no conflict of interest between service consumers, formally* $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$*, if there exists a subset of Role* $\mathbb{R}$ *named* $\widetilde{R_a}$*, which includes all roles that have been mapped with service consumers, and the relationships between* r_i *and roles in* $\widetilde{R_a}$ *are the same as the relationships between* sc_m *and service consumers that have been mapped to the roles in* $\widetilde{R_a}$*. Formally,* $\exists \widetilde{R_a} \subseteq \mathbb{R}\text{-}\{r_i\}$, $\forall\ r_j \in \widetilde{R_a}$, $(r_j,\ assigned\_sc(r_j)) \in SCA$, $\forall r'_j \in \mathbb{R}\text{-}\{r_i\}\text{-}\widetilde{R_a}$, $assigned\_sc(r'_j) = \emptyset$, $\mathcal{RL}(r_i, r_j) = \mathcal{RL}(sc_m, assigned\_sc(r_j))$*, where* $\mathcal{RL}$*(element, element)=*{*exclusive* ($\otimes$)*, non-exclusive* ($\ominus$)} *reflecting the exclusive, or non-exclusive relationships between elements.*

As an alternative solution, roles can be assigned without using the above authorization rule but the conflict of authorization between consumers will be checked at run time. The mappings between service consumers and roles can be stored in the system at design time. The conflict of interest between consumers is checked when the assigned roles are activated simultaneously by a specific consumer. The authorization rule named *Dynamic Conflict-Free Role Relationship Check* (**D-CF-R-RC**) is specified as follows:

Authorization Rule 2 D-CF-R-RC: *Let* $\mathbb{SC}$ *be a set of Service Consumers. Let* $\mathbb{R}$ *be a set of Roles. There is no runtime conflict of interest between service consumers, formally* $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$, *and* $r_i \in RSi(SCSi(sc_m))$ *and/or* $r_i \in RSc(SCSc(sc_m))$, *if there exists a subset of Role* $\mathbb{R}$ *named* $\widetilde{R_b}$, *which includes all roles that are being activated, and the relationships between* r_i *and all roles in* $\widetilde{R_b}$ *are the same as the relationships between* sc_m *and service consumers that are activating these roles in* $\widetilde{R_b}$. *Formally,* $\exists \widetilde{R_b} \subseteq \mathbb{R}\text{-}\{r_i\}$, $\forall r_k \in \widetilde{R_b}$, $\exists sc_n \in \mathbb{SC}$, $r_k \in RSi(SCSi(sc_n))$ *and/or* $r_k \in RSc(SCSc(sc_n))$, $\forall r'_k \in \mathbb{R}\text{-}\widetilde{R_b}\text{-}r_i$, $\forall sc'_n \in \mathbb{SC}$, $r'_k \notin RSi(SCSi(sc'_n))$, *and* $r'_k \notin RSc(SCSc(sc'_n))$, $\mathcal{RL}(r_i, r_k) = \mathcal{RL}(sc_m, sc_n)$.

4.2.2 Conflict of Interest between Resources

If two resources have the relationship *Exclusive* or *Non-exclusive*, the mapped resource types for these two resources must have the same relationship as *exclusive* or *non-exclusive* to prevent conflict of interest.

In Fig.4.2, if Op_a and Op_b have *exclusive* relationship ($Op_a \otimes Op_b$), then the relevant resource types ReT_i and ReT_j should be *exclusive* ($Op_a \otimes Op_b \Rightarrow ReT_i \otimes RetT_j$). The relationship between the resource types mapped with resources Re_k and Re_h must be the same as the relationship between these two resources. If the resources Re_k and Re_h are *non-exclusive* with each other ($Re_k \ominus Re_h$), e.g., belonging to one company group, then Re_k and Re_h can not be mapped to resource types ReT_i and ReT_j respectively at the same time, since ReT_i and ReT_j are *exclusive*. To avoid conflict of interest, two resources with relationship $\otimes$ or $\ominus$ must be included in the associated two resource

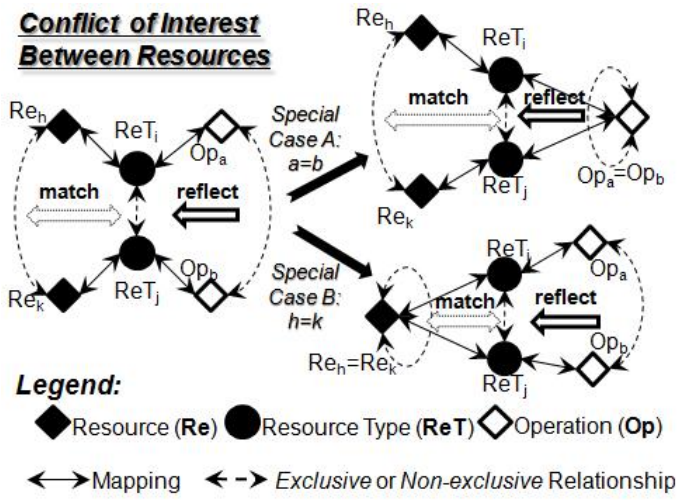

Figure 4.2: Conflict-Free Resource Type Relationship Check (CF-RT-RC)

types with the same relationship $\otimes$ or $\ominus$.

Two special cases are described in Fig.4.2, where operations (Special Case A in Fig.4.2) and resources (Special Case B in Fig.4.2) become one operation and one resource respectively. Let us take an example as special case B in Fig. 4.2. For the security reason, *Lease Application* and *Guarantor Confirmation* are *exclusive* operations in Financial Lease service, where the mapped resource type, `Funder` and `Credit Assessor`, are *exclusive*. Hence, if web service as a resource plays as both `Funder` and `Credit Assessor`, *non-exclusive* relationship exists between the resource and itself, and the resource can not be included in resource types `Funder` and `Credit Assessor` at the same time.

We devise the authorization rule named *Static Conflict-Free Resource Type Relationship Check* (**S-CF-RT-RC**) on the mapping of resources and resource types to prevent the conflict of interest between resources. Here we formally define the authorization rule at design time as follows:

Authorization Rule 3 S-CF-RT-RC: *Let* $\mathbb{R}e$ *be a set of Resources. Let* $\mathbb{R}e\mathbb{T}$ *be a set of Resource Types. There is no conflict of interest between resources, formally* $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e, (re_h, ret_i) \in RTA$*, if there exists a subset of* $\mathbb{R}e\mathbb{T}$ *named*

$\widetilde{ReT_a}$ that includes all resource types that have been mapped with resources, and the relationships between ret_i and resource types in $\widetilde{ReT_a}$ are the same as the relationships between re_h and the resources mapped with resource types in $\widetilde{ReT_a}$. Formally $\exists\widetilde{ReT_a} \subseteq \mathbb{R}e\mathbb{T}$-$\{ret_i\}$, $\forall\ ret_j \in \widetilde{ReT_a}$, $(ret_j,\ assigned\_re(ret_j)) \in RTA$, $\forall ret'_j \in \mathbb{R}e\mathbb{T}$-$\{ret_i\}$-$\widetilde{ReT_a}$, $assigned\_re(ret'_j) = \emptyset$, $\mathcal{RL}(ret_i, ret_j) = \mathcal{RL}(re_h, assigned\_re(ret_j))$.

Alternatively, the resource can be mapped to resource types without using the above authorization rule, but the conflict of interest between resources will be checked at run time. The mappings between resources and resource types can be stored in system at design time. The conflict of interest between resources are checked when the resource types are activated simultaneously by employing the resources to provide supports to the operations. Here we formally define the *Dynamic Conflict-Free Resource Type Relationship Check* (**D-CF-RT-RC**) on preventing runtime conflict of interest between resources.

Authorization Rule 4 D-CF-RT-RC: *Let $\mathbb{R}e$ be a set of Resources. Let $\mathbb{R}e\mathbb{T}$ be a set of Resource Types. There is no runtime conflict of interest between resources, formally $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e, (re_h, ret_i) \in RTA$, and $ret_i \in RTS(RES(re_h))$, if there exists a subset of $\mathbb{R}e\mathbb{T}$ named $\widetilde{ReT_b}$ includes all resource types that are being activated and the relationships between ret_i and resource types in $\widetilde{ReT_b}$ should be the same as the relationships between re_h and resources that are employed to support operations by specific resource types in $\widetilde{ReT_b}$. Formally, $\exists\widetilde{ReT_b} \subseteq \mathbb{R}e\mathbb{T}$-$\{ret_i\}$, $\forall ret_k \in \widetilde{ReT_b}$, $\exists re_l \in \mathbb{R}e$, $ret_k \in RTS(RES(re_l))$, $\forall ret'_k \in \mathbb{R}e\mathbb{T}$-$\widetilde{ReT_b}$-$\{ret_i\}$, $\forall re'_l \in \mathbb{R}e$, $ret'_k \notin RTS(RES(re'_l))$, $\mathcal{RL}(ret_i, ret_k) = \mathcal{RL}(re_h, re_l)$.*

4.2.3 Conflict of Interest between Service Consumers and Resources

Resources and service consumers can have relationship as *exclusive* or *non-exclusive* that must be the same as the relationship of mapped roles and resource types. The relationship between a resource type and a role reflects the relationship between the

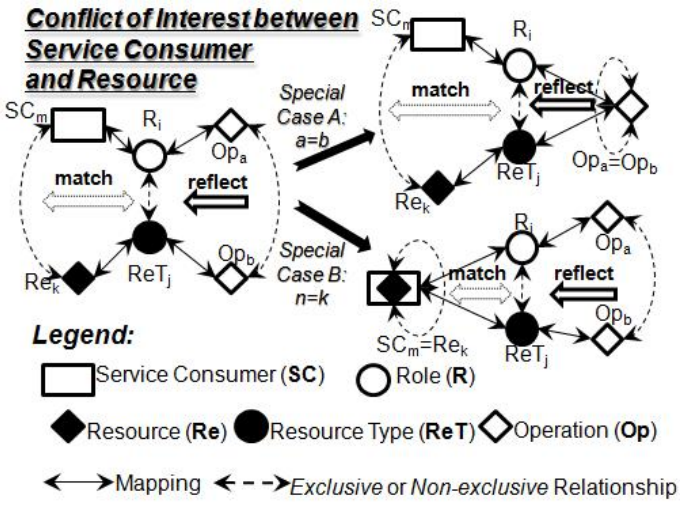

Figure 4.3: Conflict-Free Role & Resource Type Relationship Check (CF-R^{2}T-RC)

operation that the role need to access and the operation that the resource type can support. The conflict of interest between service consumers and resources can occur, if the relationship between the service consumers and the resources is not the same as the relationship of mapped role and resource type.

Two special case are also presented in Fig. 4.3, where (1) the operation that the role need to access and the operation that the resource type can support are the same one (Special Case A in Fig. 4.3), and (2) the service consumer and the resource are the same web service (Special Case B in Fig. 4.3). In special case A at Fig. 4.3, the operation that the resource type ReT_j supports is what the role R_j need to access ($Op_a = Op_b$). Their relationship is *non-exclusive* ($Op_a \ominus Op_b$). If the relationship between the mapped service consumer SC_m and resource Re_k is *exclusive* ($SC_m \otimes Re_k$), e.g., the Chinese manufactory as the resource to provide product and the USA military customer as the service consumer to rent the product, the mapping between the service consumer SC_m to the specific role R_j and the mapping between the resource Re_k to the specific resource type ReT_j can not be made simultaneously.

Let us take another example, in special case B at Fig. 4.3, a web service plays as both a service consumer and a resource($SC_m = Re_k$). Their relationship is *non-exclusive* ($SC_m \ominus Re_k$). If the operation that the web service supports as resource is

exclusive with the operation that the web service need to access as the service consumer ($Op_a \otimes Op_b$), there is a conflict of interest between the consumer and the resource. If the web service is assigned with specific role R_i to access the operation Op_a, it can not be mapped to resource type ReT_j to support operation Op_b; vise versa.

We define authorization rule named *Static Conflict-Free Role & Resource Type Relationship Check* (**S-CF-R^{2}T-RC**) to prevent the conflict of interest between the consumer and the resource. The formal specification is as follows:

Authorization Rule 5 S-CF-R^{2}T-RC: *Let $\mathbb{R}e$ be a set of Resources, and $\mathbb{SC}$ be a set of Service Consumers. Let $\mathbb{R}$ be a set of Roles, and $\mathbb{R}e\mathbb{T}$ be a set of Resource Types. There is no conflict of interest between service consumer and resource if (1) and (2) are satisfied:*

1. *service consumer sc_m and role r_i can be mapped in SCA, formally $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$, if there exists a set named $\widetilde{ReT_a}$ that is a subset of Resource Types and includes all resource types that have been mapped with specific resources; The relationships between r_i and resource types in $\widetilde{ReT_a}$ should be the same as the relationships between sc_m and the resources that are mapped with the resource types in $\widetilde{ReT_a}$. Formally, $\exists \widetilde{ReT_a} \subseteq \mathbb{R}e\mathbb{T}$, $\forall\ ret_j \in \widetilde{ReT_a}$, $(ret_j,\ assigned\_re(ret_j)) \in RTA$, $\forall ret'_j \in \mathbb{R}e\mathbb{T} - \widetilde{ReT_a}$, $assigned\_re(ret'_j) = \emptyset$, $\mathcal{RL}(ret_j, r_i) = \mathcal{RL}(assigned\_re(ret_j), sc_m)$.*

2. *resource type ret_i and resource re_h can be mapped in RTA, formally $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e, (re_h, ret_i) \in RTA$, if there exists a set R_a as a subset of Roles that includes all roles which have been assigned to specific service consumers, and the relationships between ret_i and all roles in R_a should be the same as the relationship between re_h and service consumers that are assigned as specific roles in R_a. Formally, $\exists \widetilde{R_a} \subseteq \mathbb{R}$, $\forall\ r_j \in \widetilde{R_a}$, $(r_j,\ assigned\_sc(r_j)) \in SCA$, $\forall r'_j \in \mathbb{R} - \widetilde{R_a}$, $assigned\_sc(r'_j) = \emptyset$, $\mathcal{RL}(ret_i, r_j) = \mathcal{RL}(re_h, assigned\_sc(r_j))$.*

The mappings between roles and service consumers, and the mappings between resources and resource types can be made without using the above authorization rule.

The conflict of interest between service consumers and resources will be checked at runtime. The mappings between role and service consumer, and the mapping between resource and resource type can be stored in system at design time. The conflict of interest between service consumer and resource is checked when the assigned role and resource type are activated simultaneously in the execution of the composite web service requested by the specific service consumer. The authorization rule named *Dynamic Conflict-Free Role & Resource Type Relationship Check* (**D-CF-R^{2}T-RC**) is specifies as follows:

Authorization Rule 6 D-CF-R^{2}T-RC *Let $\mathbb{R}e$ be a set of Resources, and $\mathbb{SC}$ be a set of Service Consumers. Let $\mathbb{R}$ be a set of Roles, and $\mathbb{R}e\mathbb{T}$ be a set of Resource Types. There is no runtime conflict of interest between service consumer and resource if (1) and (2) are satisfied:*

1. *service consumer sc_m can activate assigned role r_i, formally $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$, and $r_i \in RSc(SCSc(sc_m))$, if there exists a subset of Resource Types named $\widetilde{ReT_b}$ that includes all resource types that are activated (when resources mapped to these resource types are required to provide support to operations), and the relationship between r_i and all resource types in $\widetilde{ReT_b}$ should be the same as the relationship between sc_m and resources that are activating the resource types in $\widetilde{ReT_b}$. Formally, $\exists \widetilde{ReT_b} \subseteq \mathbb{R}e\mathbb{T}$, $\forall ret_g \in \widetilde{ReT_b}$, $\exists re_h \in \mathbb{R}e$, $ret_g \in RTS(RES(re_h))$, $\forall ret'_g \in \mathbb{R}e\mathbb{T} - \widetilde{ReT_b}$, $\forall re'_h \in \mathbb{R}e$, $ret'_g \notin RTS(RES(re'_h))$, $\mathcal{RL}(ret_g, r_i) = \mathcal{RL}(re_h, sc_m)$.*

2. *resource type ret_i is being activated by resource re_h, formally $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e, (re_h, ret_i) \in RTA$, and $ret_i \in RTS(RES(re_h))$, if there exists a subset of $\mathbb{R}$ named $\widetilde{R_b}$ that includes all of roles that have been activated, and the relationship between ret_i and roles in $\widetilde{R_b}$ should be the same as the relationship between re_h and service consumers that activate the roles in $\widetilde{R_b}$. Formally, $\exists \widetilde{R_b} \subseteq \mathbb{R}$, $\forall r_g \in \widetilde{R_b}$, $\exists sc_m \in \mathbb{SC}$, $r_g \in RSc(SCSc(sc_m))$, $\forall r'_g \in \mathbb{R}\text{-}\widetilde{R_b}$, $\forall sc'_m \in \mathbb{SC}$, $r'_g \notin RSc(SCSc(sc'_m))$, $\mathcal{RL}(ret_i, r_g) = \mathcal{RL}(re_h, sc_m)$.*

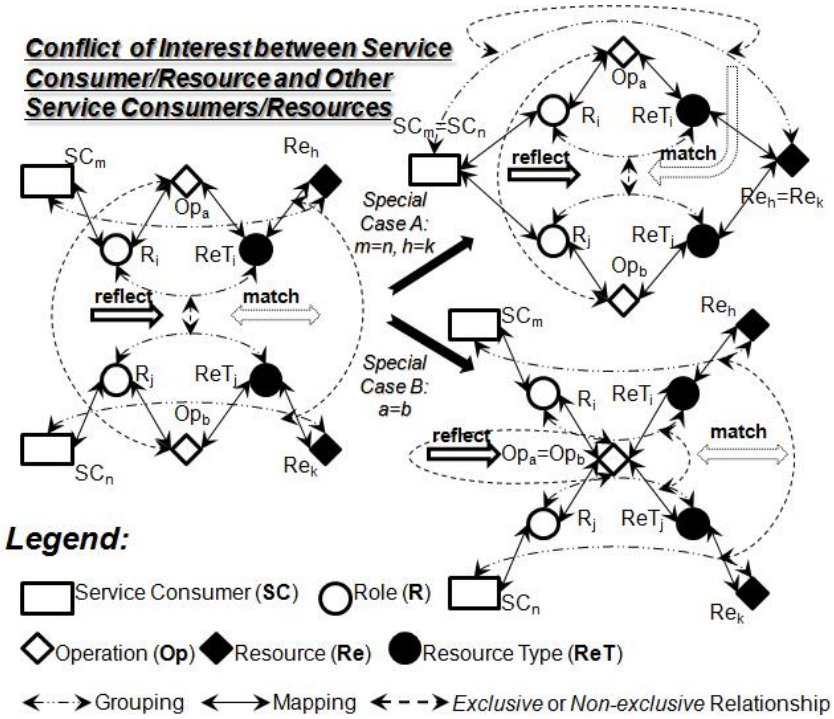

FIGURE 4.4: Conflict-Free Pair of Role & Resource Type Relationship Check (CF-PR^{2}T-RC)

Conflict of interest between one pair of service consumer/resource and other pairs of service consumer/resource is another new type of conflict of interest which can be identified in SOAC. A service consumer and a resource are put in one pair when the service consumer requests the access of the operation of a composite web service and the operation needs the support of the resource. The relationships between pairs of role/resource type reflect the relationships between operations mapped to these pairs of role/resource type. If two pairs of service consumer/resource have the relationship *Exclusive* or *Non-exclusive*, the pairs of mapped roles and resource types must have the same relationship as *Exclusive* or *Non-exclusive*

For example, in Fig. 4.4, if the operations are *exclusive* ($Op_a \otimes Op_b$), the relationship between the pairs of mapped roles and resource types must also be *exclusive* ($Op_a \otimes Op_b \Rightarrow (R_i, ReT_i) \otimes (R_j, ReT_j)$). Note, here the relationship between operations will be reflected by the relationship between the pairs of roles and resource types rather than considering the relationship between roles or resources types individually which are

discussed in previous sections. If the relationship between two pairs of service consumer and resource are *non-exclusive* ($(SC_n Re_h) \ominus (SC_m, Re_k)$), the pairs of mapped roles and resource types must also be *non-exclusive* to prevent the conflict of interest.

Two special cases are illustrated in Fig. 4.4, where (1) the pairs of service consumer and resource are the same one (Special case A in Fig. 4.4 ($SC_m = SC_n$ and $Re_h = Re_k$)), and (2) the operations in different authorizations are the same one (Special cased B in Fig. 4.4 ($Op_a = Op_b$)). Let us take an example in special case A. We separate the role of `Customer` in motivating scenario as `Commercial Customer` and `Military Customer` who will rent product for civil use and military use respectively. The resource type `Supplier` is also separated into `Vehicle Engine Supplier`, `Vehicle Engine Accessory Supplier`, and `Other Supplier`, according to the product that the supplier can provide. The operation of Lease Application is also separated into *Lease Application for Engine*, *Lease Application for Engine Accessory*, and *Lease Application for Others*.

Therefore, when a service consumer is mapped with the role `Military Customer`, and the goods it orders need to be supplied by part manufactory mapped with resource type `Vehicle Engine Supplier`, it will violate the law if Financial Lease service also uses the same manufactory that is mapped with resource type `Vehicle Engine Accessory Supplier` to supply the engine accessory to the same consumer that is mapped with role `Commercial Customer`.

In this case, the *exclusive* relationship between the operations of *Lease Application for Engine* and *Lease Application for Engine Accessory* requires that the relationship between the pair of `Military Customer` and `Vehicle Engine Supplier` and the pair of `Commercial Customer` and `Vehicle Engine Accessory Supplier` are *exclusive* also. If the two pairs of role and resource type are mapped to the same pair of service consumer and resource with a *non-exclusive* relationship, the conflict of interest occurs.

This conflict of interest is identified to prevent the following two things happening at the same time. The first thing is to assemble the engine for military use with the engine accessory for civil use and the second thing is to purchase engine and engine

accessory from the same part suppliers. We can observe that the service consumer can be mapped with both roles `Military Customer` and `Commercial Customer` without causing conflict of interest between customers, e.g., when the goods it rents are not provided by the same component service. We can also observe that the manufactory (the resource) can be mapped with both resource types `Vehicle Engine Supplier` and `Vehicle Engine Accessory Supplier` without causing conflict of interest between resources, e.g., when the products it provides are not for the same service consumer. The conflict of interest occurs when the service consumer is mapped with both roles and the resource is mapped with both resource types. In a summary, if the manufactory (a resource) as `Vehicle Engine Supplier` to provide engine to a service consumer as `Military Customer`, it should not provide engine accessory to the same service consumer that is identified as `Commercial Customer`; vise versa.

We set up authorization rule named as *Static Conflict-Free Pair of Role & Resource Type Relationship Check* (**S-CF-PR^{2}T-RC**) to prevent conflict of interest between two pairs of service consumer/resource. Here we formally define the authorization rule at design time as follows:

Authorization Rule 7 S-CF-PR^{2}T-RC *Let $\mathbb{R}e$ be a set of Resources, and $\mathbb{SC}$ be a set of Service Consumers. Let $\mathbb{R}$ be a set of Roles, and $\mathbb{R}e\mathbb{T}$ be a set of Resource Types. There is no conflict of interest between one pair of service consumer/resource and another pair of service consumer/resource, formally $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$, $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e$, $(re_h, ret_i) \in RTA$, assigned_op$(r_i) \cap$ assigned_ret$(ret_i) \neq \emptyset$, if there exists a set named $\widetilde{ReT_a}$ that is a subset of Resource Types and includes all resource types which have been mapped with specific resources, and exists a set named $\widetilde{R_a}$ that is a subset of Roles and includes all roles which have been assigned to specific service consumers; There must exist a resource type (ret_k) and a role (r_k) that map to the same operations; the relationship between (ret_i, r_i) and (ret_k, r_k) should be the same as the relationships between (re_h, sc_m) and pairs of resources and service consumers that are mapped to ret_k and r_k respectively. Formally, $\exists \widetilde{ReT_a} \subseteq \mathbb{R}e\mathbb{T}$, $\forall\ ret_j \in \widetilde{ReT_a}$, $(ret_j,$ assigned_re$(ret_j)) \in RTA$, $\forall ret'_j \in \mathbb{R}e\mathbb{T} - \widetilde{ReT_a}$, assigned_re$(ret'_j) = \emptyset$, $\exists \widetilde{R_a} \subseteq \mathbb{R}$,*

$\forall\ r_j \in \widetilde{R_a}$, $(r_j,\ assigned\_sc(r_j)) \in SCA$, $\forall r'_j \in \mathbb{R} - \widetilde{R_a}$, $assigned\_sc(r'_j) = \emptyset$, $\exists r_k \in \widetilde{R_a}, \exists ret_k \in \widetilde{ReT_a}$, $assigned\_op(r_k) \cap assigned\_ret(ret_k) \neq \emptyset$, $\mathcal{RL}((ret_i, r_i), (ret_k, r_k)) = \mathcal{RL}\ ((re_h, sc_m), (assigned\_re(ret_k), assigned\_sc(r_k)))$.

Without using the above authorization rules, the conflict of interest between pairs of service consumer and resource can be checked at runtime. The mapping between service consumer and role, and the mapping between resource type and resource are stored in system. The conflict of interest between pairs of service consumer/resource is checked when the associated roles and resource types are activated simultaneously. The authorization rule named *Dynamic Conflict-Free Pair of Role & Resource Type Relationship Check* (**D-CF-PR^{2}T-RC**) is specified as follows:

Authorization Rule 8 D-CF-PR^{2}T-RC *Let $\mathbb{R}e$ be a set of Resources, and $\mathbb{SC}$ be a set of Service Consumers. Let $\mathbb{R}$ be a set of Roles, and $\mathbb{R}e\mathbb{T}$ be a set of Resource Types. There is no run time conflict of interest between one pair of service consumer/resource and another pair of service consumer/resource, formally $\exists r_i \in \mathbb{R}, \exists sc_m \in \mathbb{SC}, (r_i, sc_m) \in SCA$, $\exists ret_i \in \mathbb{R}e\mathbb{T}, \exists re_h \in \mathbb{R}e, (re_h, ret_i) \in RTA$, $assigned\_op(r_i) \cap assigned\_ret(ret_i) \neq \emptyset$, $r_i \in RSc\ (SCSc(sc_m))$, and $ret_i \in RTS(RES(re_h))$, if there exists a subset of $\mathbb{R}e\mathbb{T}$ named $\widetilde{ReT_b}$ which includes all resource types that are being activated, and there also exists a subset of $\mathbb{R}$ named $\widetilde{R_b}$ which includes all roles that are being activated. There must exist a resource type ret_k belonging to $\widetilde{ReT_b}$ and a role r_k belonging to $\widetilde{R_b}$ that are supporting and accessing the same operations respectively; The relationship between (ret_i, r_i) and (ret_k, r_k) should be the same as the relationship between (re_h, sc_m) and pairs of resources and service consumers that are activating ret_k and r_k respectively. Formally $\exists \widetilde{ReT_b} \subseteq \mathbb{R}e\mathbb{T}$, $\forall ret_x \in \widetilde{ReT_b}$, $\exists re_y \in \mathbb{R}e$, $ret_x \in RTS(RES(re_y))$, $\forall ret'_x \in \mathbb{R}e\mathbb{T} - \widetilde{ReT_b}$, $\forall re'_y \in \mathbb{R}e$, $ret'_x \notin RTS(RES(re'_y))$, $\exists \widetilde{R_b} \subseteq \mathbb{R}$, $\forall r_x \in \widetilde{R_b}$, $\exists sc_y \in \mathbb{SC}$, $r_x \in RSc(SCSc(sc_y))$, $\forall r'_x \in \widetilde{R_b}$, $\forall sc'_y \in \mathbb{SC}$, $r'_x \notin RSc(SCSc(sc'_y))$, $\exists r_k \in \widetilde{R_b}$,$\exists ret_k \in \widetilde{ReT_b}$, $assigned\_op(r_k) \cap assigned\_ret(ret_k) \neq \emptyset$, $\exists sc_n \in \mathbb{SC}$, $r_k \in RSc(SCSc(sc_n))$, $\exists re_g \in \mathbb{R}e$, $ret_k \in RTS(RES(re_g))$, $\mathcal{RL}((ret_i, r_i), (ret_k, r_k)) = \mathcal{RL}((re_h, sc_m), (re_g, sc_n))$.*

4.3 Advanced Management of Conflict of Interest

In this section, the conflict of interest for one element in SOAC will be checked and prevented. The four types of **Conflicts of Interest** are identified based on SOAC from one service consumer's and/or one resource's perspective. Each type of conflict of interest can be detected and controlled at design time or at runtime. If the conflict of interest is dealt at design time, it is referred to as Static Conflict of Interest (S-CoI). If the conflict of interest is dealt at runtime, it is referred to as Dynamic Conflict of Interest (D-CoI).

For *service provision*, S-CoI or D-CoI can be used to avoid the conflicted authorizations to one service consumer. For *service realization*, S-CoI or D-CoI can be used to prevent the conflicted authorizations to one resource. For the integration of *service provision* and *service realization*, S-CoI or D-CoI can be used to avoid assigning this web service to one conflicted pair of role and resource type, when the web service is a service consumer and a resource simultaneously. They can also avoid assigning two conflicted pairs of role/resource type to one pair of service consumer and resource.

4.3.1 Conflict of Interest for One Service Consumer

In *service provision*, a service consumer is assigned with specific roles to gain the permissions to access the relevant operations. Hence, in SOAC, a service consumer can be mapped with multiple roles and the roles are mapped with various operations. The mapping between a Role and an Operation is much more stable than that between a Role and a Service Consumer. The reason is that the service consumer is normally unknown in advance and changing dynamically. For instance, the service consumer can terminate the lease application and restart a new lease application at any time. Mappings between role and operation are not changed such frequently. Hence, the relationship between different roles can be used as a reflection of the relationship between different operations that are mapped with those roles respectively. This relationship can be conflicted or non-conflicted. To avoiding the conflict of interest in terms of one consumer, we can prevent the service consumer from mapping with conflicted roles

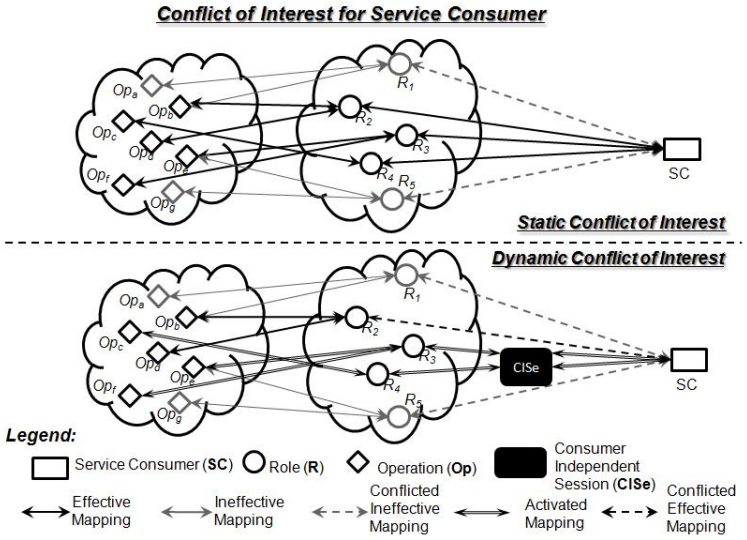

Figure 4.5: Conflict of Interest for One Service Consumer

simultaneously.

As an example, the amount of rental on specific goods varies based on the purpose of customer on how to deal with the product after leasing period. Hence, Operations *Purchase* and *Return* should be avoided to both be accessible by one service consumer. Due to the mappings between operations and roles are stable (not prone to changing), the relationships between roles can reflect the relationships between their mapped operations. Based on the exclusive relationship between the operations of *Purchase* and *Return*, we should define the role `Customer_Purchase` and `Customer_Return` that are mapped to operations *Purchase* and *Return* respectively as conflicted roles. The static conflict of interest (S-CoI) in service provision can be avoided if we do not enable the mappings between one service consumer and the two conflicted roles simultaneously.

In Fig. 4.5, we use white diamonds to represent operations and white circles to represent roles. A white rectangle represents a service consumer. If a role is assigned to the service consumer, the mappings between the role and the operations and the mapping between the role and the service consumer become effective. At the upper part of Fig. 4.5, the service consumer is not mapped to R_1 and R_5. The mapping between R_1 and Op_a and other mappings related to R_1 and R_5 are ineffective represented with

grey lines. In the diagram, we assume that there is conflict of interest if operations Op_a and Op_c are accessed by the same service consumer. The relationship between R_1 and R_4 that are mapped to the operations Op_a and Op_c respectively have the conflict of interest. R_3 and R_5 have the conflict of interest due to the similar assumption on the relationship between Op_g and Op_f. When a service consumer SC is effectively mapped with R_2, R_3, and R_4 as the upper part of Fig. 4.5, the mappings between the service consumer and the roles R_1 and R_5 are conflicted ineffective mappings (grey dash lines). Conflicted ineffective mappings mean that even the service consumer bears the credential to be qualified for specific role, the mappings between the service consumer and the role still can not be effective, since it can cause conflict of interest for the service consumer. In a word, the creation of a mapping should not cause the service consumer to be assigned with two conflicted roles. The conflicted ineffective mappings can be changed to ineffective mappings. For example, R_1 and R_4 are conflicted roles. when the mapping between R_4 and service consumer becomes ineffective, the mappings between R_1 and service consumer is changed from conflicted ineffective mapping to ineffective mapping. It means, at this time, if the service consumer can satisfy the authorization constraint for R_1, the mapping between the service consumer and R_1 can become effective which will not cause conflicted roles R_1 and R_4 assigned to the service consumer simultaneously.

We provide following formal definition to introduce the authorization rule that can be implemented to avoid conflict of interest in service provision.

Definition 8 *Type(e)={**R**, **Op**, **SC**, **ReT**, **Re**} is a function to return the type of the element in SOAC, where e represents a set of elements.*

For example, Type(R_i)=**R** means that the type of the set of elements R_i in SOAC is role.

Definition 9 *Conflict(e_a,e_b)={TRUE, FALSE}, is a boolean function to illustrate if two sets of elements (e_a, e_b) with same type (role and resource type can be treated as same type to compare) are conflicted or not.*

Definition 10 *$Map^s(e_a, e_b)$={e_Map, C_ie_Map, nC_ie_Map} is a function to illustrate the type of mapping between two elements with different types at design time, where e_a, and e_b can be two elements with different types, i.e., Type(e_a)≠Type(e_b). e_Map represents effective mapping, C_ie_Map represents Conflicted Ineffective Mapping, and nC_ie_Map represents Non-Conflicted Ineffective Mapping.*

We define an authorization rule named *Static Conflict-Free Role Authorization* (**S-CF-RA**) to restrict the conflict of interest regarding to the mappings between one service consumer and its associated roles. The main idea of S-CF-RA is that if a mapping between a role and a service consumer becomes effective, then the role must not be conflicted with other roles that have been assigned to this service consumer. The formal specification is as follows,

Authorization Rule 9 S-CF-RA: *Let $\boldsymbol{SC}$ be a set of Service Consumers. Let $\boldsymbol{R}$ be a set of Roles. There is no conflict of interest for service consumer sc_m ($sc_m \in \boldsymbol{SC}$) to be mapped with r_i ($r_i \in \boldsymbol{R}$), i.e. the mapping between sc_m and r_i can be effective, if there exists a subset of Roles named $\widetilde{R_a}$ ($|\widetilde{R_a}| \geq 0$), which includes all roles that have been mapped with the service consumer sc_m, and the relationships between these roles in the set of $\widetilde{R_a}+r_i$ are not conflicted. Formally, $Map^s(sc_m, r_i)$=e_Map, if $\widetilde{R_a} = assigned\_sc(sc_m)$, $\forall \widetilde{R_c} \subseteq \widetilde{R_a}$, $Conflict(\widetilde{R_a}, \widetilde{R_c}+r_i)$=FALSE.*

As an alternative solution, the mappings between the service consumer and roles can be stored in the system at design time. The conflict of interest will be avoided at runtime by controlling the simultaneous activations of assigned roles for the specific service consumer. In the lower part of Fig. 4.5, we can observe that, only R_3 and R_4 are activated by the service consumer SC, where their mappings to the service consumer are changed from effective mappings to activated mappings. R_2 is not activated in that circumstance since its activation may cause conflict of interest for the service consumer SC at runtime. In this case, the mapping between SC and R_2 can not be activated. The following specification describes the types of mappings at runtime.

Definition 11 *$Map^d(e_a, e_b)$={C_e_Map, nC_e_Map, C_ie_Map, nC_ie_Map, a_map} is a function to illustrate the types of mapping between two elements with different types at runtime, where ELE_a and ELE_b are two elements with different types, i.e., Type(e_a)≠Type(e_b). C_e_Map represents Conflicted Effective Mapping; while nC_e_Map represents Non-Conflicted Effective Mapping. C_ie_Map represents Conflicted Ineffective Mapping and nC_ie_Map represents Non-Conflicted Ineffective Mapping. a_Map means Activated Mapping.*

When the assigned role is activated by the service consumer, a consumer independent session (**CISe**) is introduced. We can restrict the mappings among the service consumer, CISe, and roles, to avoid the conflict of interest for this service consumer who has the session at runtime. The authorization rule named *Dynamic Conflict-Free Role Authorization* (**D-CF-RA**) is specified as follows,

Authorization Rule 10 ***D-CF-RA:*** *Let **SC** be a set of Service Consumers. Let **R** be a set of Roles, and CIS be a consumer independent session. There is no conflict of interest for service consumer sc_m ($sc_m \in \boldsymbol{SC}$) to activate r_i ($r_i \in \boldsymbol{R}$), i.e. the mappings among sc_m, CIS, and r_i is activated mapping, if there exists a subset of Role named $\widetilde{R_a}$ ($|\widetilde{R_a}| \geq 0$), which includes all roles that have been activated by the service consumer sc_m in session CIS, and the relationships between these roles in the set of $\widetilde{R_a}+r_i$ are not conflicted. Formally, $Map^d(sc_m, CIS)$=a_Map and $Map^d(CIS, r_i)$=a_Map, if $\widetilde{R_a} = cise\_r(sc\_cise(sc_m))$, $\forall \widetilde{R_c} \subseteq \widetilde{R_a}$, $Conflict(\widetilde{R_a}, \widetilde{R_c}+r_i)$=FALSE.*

4.3.2 Conflict of Interest for One Resource

In service realization, the mappings between operations and resource types are much more stable than the mappings between resource types and resources, due to the same reason when considering the mappings among service consumers, roles, and operations. The relationships between resource types actually reflect the relationships between the mapped operations. The static conflict of interest (S-CoI) in service realization considers the mappings between one resource and multiple resource types at design time, where one resource can not be simultaneously mapped with two conflicted resource

types. In Fig.4.6, black diamonds represent resources, and black circles represent resource types. The upper part of Fig.4.6 shows the conflict of interest in service realization at design time; while the lower part shows runtime conflict of interest (that will be discussed later). At the upper part, $Op_f \otimes Op_b \Rightarrow ReT_2 \otimes ReT_5$, i.e., the relationship between ReT_2 and ReT_5 is conflicted since the mapped operations Op_f and Op_b are conflicted. Hence, when the resource *re* is effectively mapped ReT_3, ReT_4, and ReT_5, we can observe from the diagram that the mapping between resource *re* and ReT_2 is conflicted ineffective mapping; while the mapping between Resource *re* and ReT_1 is ineffective mapping. The difference between these two mappings is, that the mapping between *re* and ReT_1 can become effective mapping at any time; while the mapping between ReT_2 and *re* can not be changed until its conflicted resource type ReT_5 is not effectively mapped to *re*.

Let us take an example for S-CoI in service realization. For avoiding price cheat, *Lease Application* and *Monthly Bill* are *mutually exclusive* (conflicted) operations in Financial Lease, which require involving different suppliers, `Value Supplier` who values the price of the product that the customer wants to rent and `Maintenance Supplier` who provides the machine and maintenance service. The resource types for the `Value Supplier` and `Maintenance Supplier` are conflicted with each other. If one resource is mapped to both `value Supplier` and `Maintenance Supplier`, static conflict of interest occurs in service realization.

We devise the authorization rule named *Static Conflict-Free Resource Type Authorization* (**S-CF-RTA**) for the mappings between resources and resource types to prevent the conflict of interest at design time. Here we formally specify the authorization rule as follows,

Authorization Rule 11 S-CF-RTA: *Let* $\boldsymbol{ReT}$ *be a set of Resource Types. Let* $\boldsymbol{Re}$ *be a set of Resources. There is no conflict of interest for resource* re_n *(*$re_n \in \boldsymbol{Re}$*) to be mapped with* ret_j *(*$ret_j \in \boldsymbol{ReT}$*), i.e. the mapping between* re_n *and* ret_j *can be effective, if there exists a subset of Resource Type named* $\widetilde{ReT_a}$ *(*$|\widetilde{ReT_a}| \geq 0$*), which includes all resource types that have been mapped with the resource* re_n*, and the relationships*

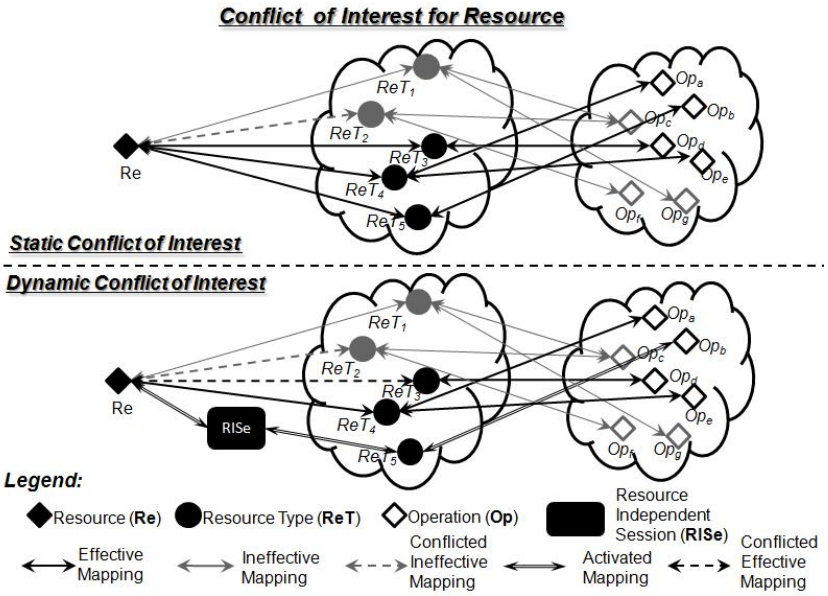

FIGURE 4.6: Conflict of Interest for One Resource

between these resource types in the set of $\widetilde{ReT_a}+ret_i$ *are not conflicted. Formally,* $Map^s(re_n, ret_j)$=*e_Map, if* $\widetilde{ReT_a} = assigned\_re(re_n)$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\widetilde{ReT_a}, \widetilde{ReT_c}+ret_j)$=*FALSE.*

Alternatively, the mappings between resources and resource types can be stored in system at design time. The conflict of interest for the resource are checked when the resource types are activated simultaneously by employing the resource to provide support to the operations. In the lower part of Fig. 4.6, the resource is activating ReT_5 to provide support to Op_b. Because the ReT_3 and ReT_5 are conflicted with each other at runtime, the resource can not activate ReT_3 in the system simultaneously with ReT_5. The authorization rule of the *Dynamic Conflict-Free Resource Type Authorization* (**D-CF-RTA**) is defined as follows,

Authorization Rule 12 ***D-CF-RTA:*** *Let* ***ReT*** *be a set of Resource Types. Let* ***Re*** *be a set of Resources, and RIS be a resource independency session. There is no conflict of interest for resource* re_n ($re_n \in$***Re***) *to activate* ret_j ($ret_j \in$***ReT***), *i.e. the mapping among* re_n, *RIS, and* ret_j *are activated mapping, if there exists a subset of*

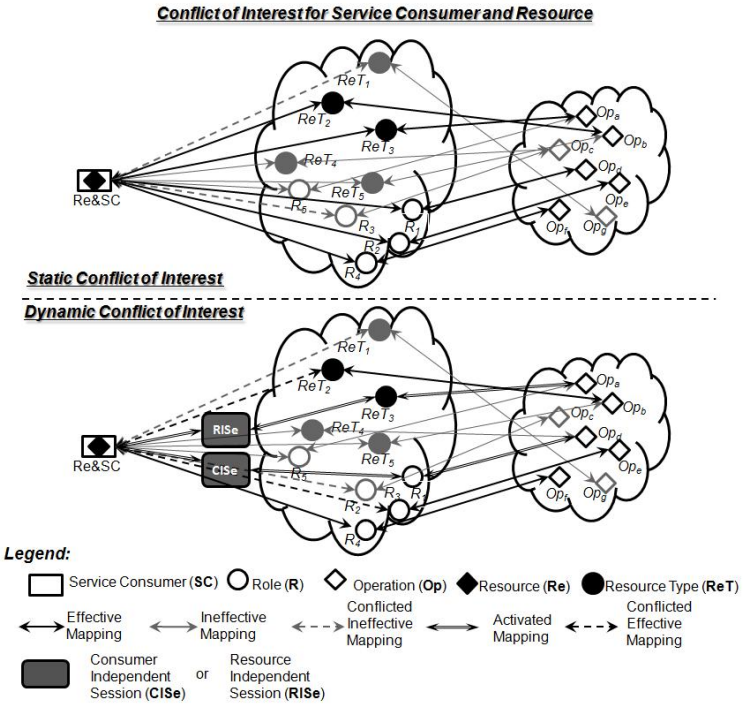

FIGURE 4.7: Conflict of Interest for One Service Consumer and Resource

Resource Type named $\widetilde{ReT_a}$ *(*$|\widetilde{ReT_a}| \geq 0$*), which includes all resource types that have been activated by the resource* re_n*, and the relationships between these resource types in the set of* $\widetilde{ReT_a}$*+*ret_i *are not conflicted. Formally,* $Map^d(re_n, RIS)$*=a_Map and* $Map^d(RIS, ret_j)$*=a_Map if* $\widetilde{ReT_a} = rise\_ret(re\_rise(re_n))$*,* $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$*,* $Conflict$ *(*$\widetilde{ReT_a}$*,* $\widetilde{ReT_c}$*+*ret_j*)=FALSE.*

4.3.3 Conflict of Interest for One Pair of Service Consumer and Resource

One web service can play as both a resource and a service consumer simultaneously. A new kind of conflict of interest specifies the situation when the role and the resource type are mapped with the same web service. For example, the qualified credit organization can be assigned a resource type `Fixed_Asset_Evaluation_Organization` to evaluate the credit history of service customers with fixed asset. If a credit organization with fixed asset submits a lease application to Financial Lease and it is

qualified to provide fixed asset evaluation as well, then the conflict of interest can occur when the credit organization is assigned as both `Customer_with_Fixed_Asset` and `Fixed_Asset_Evaluation_Organization`. The credit organization should not have a chance to evaluate a lease application of itself.

At the upper part of Fig. 4.7, a web service is assigned as R_1, R_2, and R_4; while it works as resource types ReT_2 and ReT_3 simultaneously. The mappings between the web service and these elements are effective mappings. However, the mapping between the service and ReT_1 is conflicted ineffective mapping, since ReT_1 has the conflict of interest with R_4, and R_4 is currently effectively mapped with the service. The same circumstance occurs due to the conflict between R_3 and ReT_3. It makes the mapping between R_3 and the web service conflicted ineffectively, as ReT_3 is currently effectively mapped with the service.

We devise the authorization rule named *Static Conflict-Free Role and Resource Type Authorization* (**S-CF-R^2TA**) to prevent the conflict of interest in terms of the web service that plays as conflicted role and resource types simultaneously at design time. Here we formally define the authorization rule as follows,

Authorization Rule 13 S-CF-R^2TA: *Let* $\boldsymbol{SC}$ *be a set of Service Consumers and* $\boldsymbol{R}$ *be a set of Roles. Let* $\boldsymbol{ReT}$ *be a set of Resource Types and* $\boldsymbol{Re}$ *be a set of Resources. There is no conflict of interest for resource* re_n *(*$re_n \in \boldsymbol{Re}$*) and service consumer* sc_m *(*$sc_m \in \boldsymbol{SC}$*)* $(re_n = sc_m)$ *to be mapped with* ret_j *(*$ret_j \in \boldsymbol{ReT}$*) or* r_i *(*$r_i \in \boldsymbol{R}$*), if there exists a subset of Resource Type named* $\widetilde{ReT_a}$ *(*$|\widetilde{ReT_a}| \geq 0$*) which includes all resource types that have been mapped with the resource* re_n*, there exists a subset of Role named* $\widetilde{R_a}$ *(*$|\widetilde{R_a}| \geq 0$*) which includes all roles that have been mapped with the service consumer* sc_m*, and the relationships between these resource types and roles in the set of* $\widetilde{ReT_a}+\widetilde{R_a}+ret_i$ *or the set of* $\widetilde{ReT_a}+\widetilde{R_a}+r_i$ *are not conflicted. Formally,*

- $Map^s(re_n, ret_j)$=e_Map *if* $\widetilde{R_a}$ =assigned_sc(sc_m) $(re_n = sc_m)$, $\widetilde{ReT_a}$=assigned_re$(re_n))$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\widetilde{ReT_a}+\widetilde{R_a}, \widetilde{ReT_c}+\widetilde{R_c}+ret_j)$=FALSE, *or*
- $Map^s(sc_m, r_i)$=e_Map, *if* $\widetilde{R_a}$ =assigned_sc(sc_m) $(re_n = sc_m)$, $\widetilde{ReT_a}$=assigned_re$(re_n))$ $(re_n = sc_m)$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\widetilde{ReT_a}+\widetilde{R_a}, \widetilde{ReT_c}+\widetilde{R_c}+r_i)$=FALSE.

The mappings between a web service and assigned roles, and the mappings between the same web service and the assigned resource types can be made at design time without enforcing the above authorization rule. The conflict of interest for a web service that can play both as a service consumer and a resource simultaneously will be checked at runtime, when the assigned role and resource type are activated at the same time by the web service. For example, a web service can be simultaneously assigned the resource types of `Fixed_Asset_Evaluation_Institute` and the role of `Customer_with_Fixed_Asset` without causing any conflict of interest. However, the conflict of interest can occur only after the two assigned role and resource types are activated by the web service at the same time, where its lease application can be evaluated by itself. At the lower part of Fig. 4.7, ReT_2 is assigned to the web service. But it can not be activated since it conflicts with R_1 that is being activated by the web service. R_2 can not be activated in the lower part of Fig. 4.7 due to the same reason between R_2 and ReT_3. The authorization rule named *Dynamic Conflict-Free Role and Resource Type Authorization* (**D-CF-R^2TA**) is specified as follows,

Authorization Rule 14 D-CF-R^2TA: *Let* $\boldsymbol{SC}$ *be a set of Service Consumers and* $\boldsymbol{R}$ *be a set of Roles. Let* $\boldsymbol{ReT}$ *be a set of Resource Types and* $\boldsymbol{Re}$ *be a set of Resources. There is no conflict of interest for resource* re_n *(*$re_n \in \boldsymbol{Re}$*) and service consumer* sc_m *(*$sc_m \in \boldsymbol{SC}$*)* $(re_n = sc_m)$ *to activate* ret_j *(*$ret_j \in \boldsymbol{ReT}$*) or* r_i *(*$r_i \in \boldsymbol{R}$*), if there exists a subset of Resource Type named* $\widetilde{ReT_a}$ *(*$|\widetilde{ReT_a}| \geq 0$*) which includes all resource types that have been activated by resource* re_n*, there exists a subset of Role named* $\widetilde{R_a}$ *(*$|\widetilde{R_a}| \geq 0$*) which includes all roles that have been activated by service consumer* sc_m*, and the relationships between these resource types and roles in the set of* $\widetilde{ReT_a}+\widetilde{R_a}+ret_j$ *or the set of* $\widetilde{ReT_a}+\widetilde{R_a}+r_i$ *are not conflicted. Formally,*

- $Map^d(re_n, ret_j)$=*a_Map if* $\widetilde{R_a}$ =$cise\_r(sc\_cise(sc_m))$ $(re_n = sc_m)$,
  $\widetilde{ReT_a}$=$rise\_ret(re\_rise(re_n))$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$,
  $Conflict(\widetilde{ReT_a}+\widetilde{R_a}, \widetilde{ReT_c}+\widetilde{R_c}+ret_j)$=*FALSE*,

- $Map^d(sc_m, r_i)$=*a_Map, if* $\widetilde{R_a}$ =$cise\_r(sc\_cise(sc_m))$ $(re_n = sc_m)$,
  $\widetilde{ReT_a}$=$rise\_ret(re\_rise(re_n))$ $(re_n = sc_m)$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$,

$Conflict(\widetilde{ReT_a}+\widetilde{R_a},\ \widetilde{ReT_c}+\widetilde{R_c}+r_i)$=*FALSE*.

There is another type of conflict of interest identified in this paper that is related with pair of service consumer/resource, where the service consumer and the resource are played by different web services. A service consumer and a resource can be put into one pair when the service consumer requests the access of an operation and the operation needs the support from the resource. However, the conflict of interest can occur among differen authorizations in terms of the pair of specific service consumer and resource. Let us take an example. When a service consumer is mapped with the role `Military Customer`, and the goods it requests to rent need to be supplied by part manufactory mapped with resource type `Vehicle Engine Supplier`, it will violate the law if Financial Lease also use the same manufactory that is mapped with resource type `Vehicle Engine Accessory Supplier` to supply the engine accessory to the same consumer that is mapped with role `Commercial Customer`. In this case, the pair of `Military Customer` and `Vehicle Engine Supplier` and the pair of `Commercial Customer` and `Vehicle Engine Accessory Supplier` are mutually exclusive. They can not be assigned simultaneously to the same pair of service consumer and the manufactory which works as a resource. If the two pairs of roles and resource types are mapped to the same pair of service consumer and resource, the conflict of interest occurs. This type of conflict of interest is identified to prevent the following two things happening at the same time. The first thing is to assemble the engine for military use with the engine accessory for civil use and the second thing is to rent engine and engine accessory from the same part suppliers. We can observe that the service consumer can be mapped with both roles `Military Customer` and `Commercial Customer` without causing conflict of interest if it rents vehicles. We can also observe that the manufactory can be mapped with both resource types `Vehicle Engine Supplier` and `Vehicle Engine Accessory Supplier` without causing conflict of interest if they are rented by `Commercial Customer` only. The conflict of interest occurs when the service consumer is mapped with both roles and the resource is mapped with both resource types. In a summary, if the manufactory as `Vehicle Engine Supplier` to provide engine to a service consumer as `Military Customer`, it should not provide engine accessory to the

same service consumer that is identified as `Commercial Customer`; vise versa.

In the upper part of Fig. 4.8, the service consumer is mapped with R_3 and R_4. R_1 and R_5 are conflicted with R_3 and R_4, so R_1 and R_5 can not be effectively mapped with the service consumer. The R_2 and the service consumer are not mapped currently. The resource is effectively mapped to resource type ReT_1, ReT_3 and ReT_4. ReT_2 is conflicted with ReT_1, so ReT_2 can not be effectively mapped with the resource. ReT_5 and the resource are not mapped currently. The pairs of role/resource type ReT_4/R_2 and ReT_1/R_1 are conflicted pairs, compared with the effective pairs of ReT_3/R_3 and ReT_3/R_4 respectively. If any pair of them is mapped with the service customer and the resource, it will cause the conflict of interest (See grey dash line with circles at both sides at the upper part of Fig. 4.8).

We design the authorization rule named Static Conflict-Free Pair of Role and Resource Type Authorization(S-CF-PR^2TA) to prevent the conflict of interest for one pair of service consumer/resource. Here we formally define the authorization rule at design time as follows,

Authorization Rule 15 ***S-CF-PR^2TA****: Let* $\boldsymbol{SC}$ *be a set of Service Consumers and* $\boldsymbol{R}$ *be a set of Roles. Let* $\boldsymbol{ReT}$ *be a set of Resource Types and* $\boldsymbol{Re}$ *be a set of Resources. There is no conflict of interest for resource* re_n *(*$re_n \in \boldsymbol{Re}$*) and service consumer* sc_m *(*$sc_m \in \boldsymbol{SC}$*) (*$re_n \neq sc_m$*) to be mapped with* ret_j *(*$ret_j \in \boldsymbol{ReT}$*) and* r_i *(*$r_i \in \boldsymbol{R}$*), if there exists a subset of Resource Type named* $\widetilde{ReT_a}$ *(*$|\widetilde{ReT_a}| \geq 0$*) which includes all resource types that have been mapped with the resource* re_n*, there exists a subset of Role named* $\widetilde{R_a}$ *(*$|\widetilde{R_a}| \geq 0$*) which includes all roles that have been mapped with the service consumer* sc_m*, and the relationships between the pairs of these resource types and roles in the set of* $\{<\widetilde{ReT_a + ret_j}, \widetilde{R_a + r_i}>\}$ *are not conflicted. Formally,*

- $Map^s(re_n, ret_j)$=*e_Map, if* $\widetilde{R_a}$ =*assigned_sc*(sc_m)), $\widetilde{ReT_a}$=*assigned_re*(re_n)) ($re_n \neq sc_m$), $\forall \widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall \widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\{<\widetilde{ReT_a}, \widetilde{R_a}>\}, \{<\widetilde{ReT_c + ret_j}, \widetilde{R_c}>\})$=*FALSE,*
- $Map^s(sc_m, r_i)$=*e_Map, if* $\widetilde{R_a}$ =*assigned_sc*(sc_m)), $\widetilde{ReT_a}$=*assigned_re*(re_n)) ($re_n \neq sc_m$), $\forall \widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall \widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\{<\widetilde{ReT_a}, \widetilde{R_a}>\}, \{<\widetilde{ReT_c}, \widetilde{R_c + r_i}>\})$=*FALSE.*

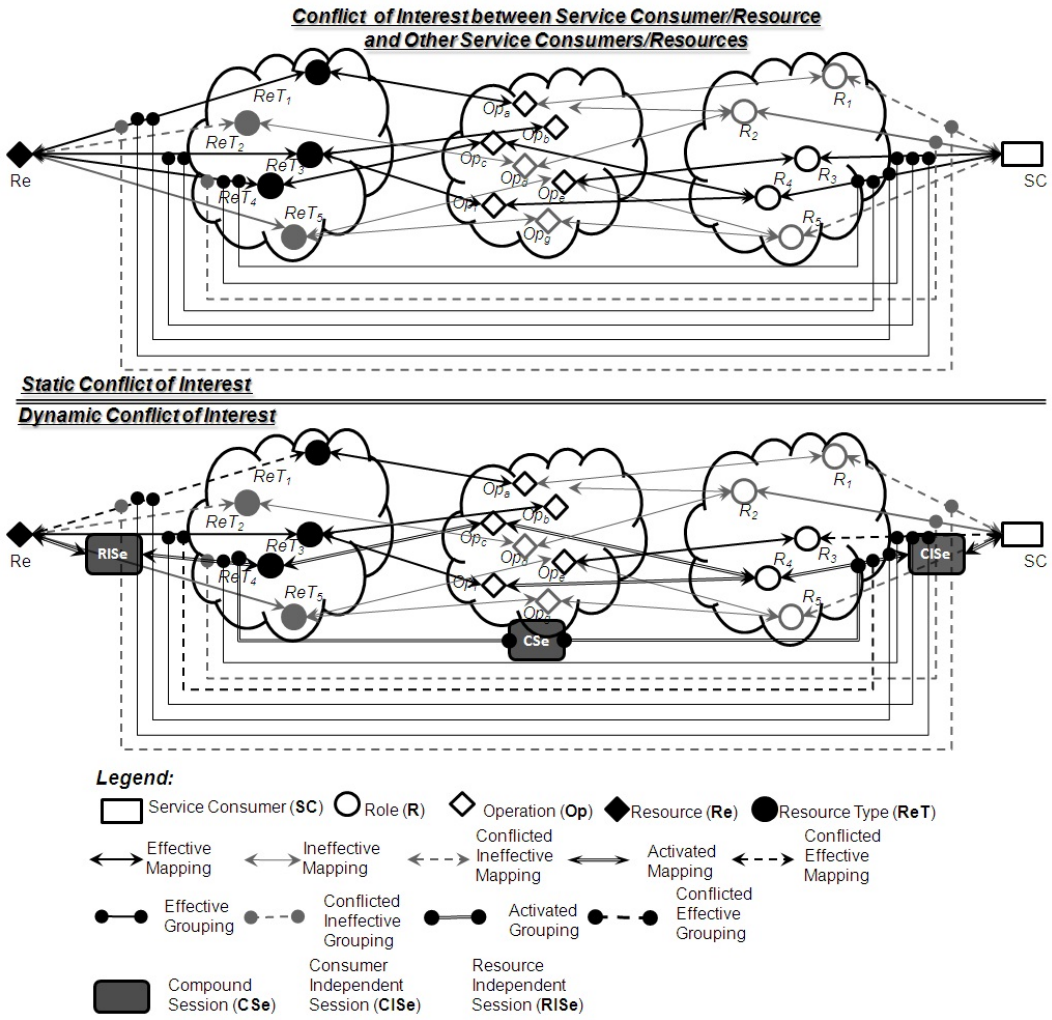

FIGURE 4.8: Conflict of Interest for One Pair of Service Consumer and Resource

The conflict of interest for one pair of service consumer and resource can also be checked at runtime, when the associated roles and resource types are activated simultaneously. At the lower part of Fig. 4.8, the service consumer and the resource activate R_4 and ReT_4 respectively. The pair R_4/ReT_3 can not be activated because it has the conflict of interest with the pair R_4/ReT_4. The authorization rule named Dynamic Conflict-Free Pair of Role and Resource Type Authorization (**D-CF-P**R^2**TA**) is used to restrict this type of conflict of interest at runtime and is specified as follows,

Authorization Rule 16 ***D-CF-P***R^2 ***TA****: Let* ***SC*** *be a set of Service Consumers and* ***R*** *be a set of Roles. Let* ***ReT*** *be a set of Resource Types and* ***Re*** *be a set of Resources.*

Authorization Policies for Avoiding Conflict of Interest between Different Elements with Same Type

Policy	Service Provision	Service Realization	Service Integration (I)	Service Integration (II)
Static (Design Time)	S-CF-R-RC	S-CF-RT-RC	S-CF-R^{2}T-RC	S-CF-PR^{2}T-RC
Dynamic (Runtime)	D-CF-R-RC	D-CF-RT-RC	D-CF-R^{2}T-RC	D-CF-PR^{2}T-RC

Authorization Policies for Avoiding Conflict of Interest between Different Authorizations for One Element

Policy	Service Provision	Service Realization	Service Integration (I)	Service Integration (II)
Static (Design Time)	S-CF-R-AC	S-CF-RT-AC	S-CF-R^{2}T-AC	S-CF-PR^{2}T-AC
Dynamic (Runtime)	D-CF-R-AC	D-CF-RT-AC	D-CF-R^{2}T-AC	D-CF-PR^{2}T-AC

Figure 4.9: Authorization Policies for Avoiding Conflict of Interest

There is no conflict of interest for resource re_n ($re_n \in \boldsymbol{Re}$) *and service consumer* sc_m ($sc_m \in \boldsymbol{SC}$) ($re_n \neq sc_m$) *to activate* ret_j ($ret_j \in \boldsymbol{ReT}$) *and* r_i ($r_i \in \boldsymbol{R}$), *if there exists a subset of Resource Type named* $\widetilde{ReT_a}$ ($|\widetilde{ReT_a}| \geq 0$) *which includes all resource types that have been activated by the resource* re_n, *there exists a subset of Role named* $\widetilde{R_a}$ ($|\widetilde{R_a}| \geq 0$) *which includes all roles that have been activated by the service consumer* sc_m, *and the relationships between the pairs of these resource types and roles in the set of* $\{<\widetilde{ReT_a + ret_j}, \widetilde{R_a + r_i}>\}$ *are not conflicted. Formally,*

- $Map^d(re_n,\ ret_j)$=a_Map *if* $\widetilde{R_a}$ =$cse\_r(sc\_cse(sc_m))$, $\widetilde{ReT_a}$=$cse\_ret(re\_cse(re_n))$ ($re_n \neq sc_m$), $sc\_cse(sc_m)$=$re\_cse(re_n)$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\{<\widetilde{ReT_a}, \widetilde{R_a}>\}, \{<\widetilde{ReT_c + ret_j}, \widetilde{R_c}>\})$=$FALSE$,

- $Map^d(sc_m,\ r_i)$=a_Map, *if* $\widetilde{R_a}$ =$cse\_r(sc\_cse(sc_m))$, $\widetilde{ReT_a}$=$cse\_ret(re\_cse(re_n))$ ($re_n \neq sc_m$), $sc\_cse(sc_m)$=$re\_cse(re_n)$, $\forall\widetilde{R_c} \subseteq \widetilde{R_a}$, $\forall\widetilde{ReT_c} \subseteq \widetilde{ReT_a}$, $Conflict(\{<\widetilde{ReT_a}, \widetilde{R_a}>\}, \{<\widetilde{ReT_c}, \widetilde{R_c + r_i}>\})$=$FALSE$.

4.4 Conclusion

This chapter addresses the conflict of interest issue regarding to both service consumers and component services in composite web services. In Fig. 4.9, the two categories of conflict of interest are presented. Four types of conflict of interest at each category are identified for different elements in SOAC and for one element in SOAC. Authorization policies at both design time and run time to deal with various types of conflict of interest are also provided.

However, the operations of the composite web service will be executed according a well-defined business logic, i.e. a workflow. How to manage the composite web service authorization in the workflow environment is still not tackled. In next chapter, we will introduce a process model SOAC-Net developed based on the Petri-Net. Authorization policies based on the SOAC-Net are designed and enforced to ensure the composite web service authorization correct and proper.

5

Process Model- SOAC-NET

5.1 Introduction

In this chapter, we will present a Petri-Net based process model named SOAC-Net which is developed based on the SOAC conceptual model. A composite web service in modern organization can be operated in a distributed environment involving multiple parties with dynamic availability and a large number of resources with evolving contents. These resources can be various types of applications that come from different organizations, e.g., component web services. Therefore, these resources can belong to different security domains, and have different security and interest requirements. In order to acquire the support from these resources, a composite web service must be able to satisfy these resources' authorization policies. On the other hand, interacting with service consumer is imperative for the execution of a composite web service. But

the service consumer can access the specific operations of a composite web service, only after it can satisfy the authorization policies of the composite service. Execution policies are used to manage the sequence of operation invocations within a composite web service, i.e., business logic. Therefore, without coordination on these policies, a composite web service may not be able to perform properly. For example, supposing an operation in a composite web service can only be invoked once, if no proper policies coordination is put in place, multiple invocations by the same service consumer can occur. Supposing operation A and operation B are a pair of sequenced operations in a composite web service. Multiple service consumers can invoke operation A; while operation B can only be accessed by one service consumer. A constraint is set up in the composite web service as the service consumer who can invoke operation B must have already invoked operation A. Now if operation A is accessed by a service consumer who is not granted permission to invoke operation B, then no service consumer is allowed to access operation B. So composite web service is discontinued. Such dependency between service consumer accesses also need to be identified between resource supports, and even between service consumer access and resource support which makes policy coordination complex.

Traditional workflow uses control flow to strictly constrain the sequence of the occurrence of the activities. However, in reality, particular in web service environment, the operations of composite web services, service consumers, and component web service are interacted in a highly loosely-coupled environment. This causes that the collaboration among these web services is dynamic and flexible without easy controlling. Still strictly constraining the sequence of interactions is not realistic. Hence, when traditional control flow is losing the capability in directing the execution of a hard-coded process in web service environment, authorization sequence becomes a necessary control to restrict the access of specific operations in composite web service. How to manage the service consumer accesses and the resource supports in a composite web service then becomes an issue that is not tackled yet.

Based on the motivating scenario in section 1.2.1, we can observe that, (1) an operation within a composite web service can be accessed and supported by multiple types

of service consumers and resources, and (2) the inherently business logic (execution policy) set up within the composite web service *Financial Lease* requires the compliance of the service consumer accesses and the resource supports. Hence, an effective authorization management should be performed on coordinating the service consumer accesses and the resource supports in a process environment. Otherwise, authorization issues can be raised to cease process execution.

Restricting when the service consumer accesses and the resource supports can/can't be used on specific operation during the execution of a composite web service is an initial step to maintain the composite web service running properly. For example, supposing the `Guarantor` finishes accessing the operation *Guarantor Confirmation*, the permission to access this operation should be revoked immediately from the `Guarantor` to avoid repeated submissions of guarantor information. Furthermore, although multiple service consumers or resources can access or support the same operations, not all of them can become effective which may suspend the execution of composite web service. In Fig. 1.3, *Monthly Bill* operation can be supported by `Funder` or `Lease Agent` and accessed by `Customer` or `Guarantor`. To avoid fraudulent activity, a `Guarantor` can pay the rental on behalf of the `Customer` to access the *Monthly Bill* operation only if the bill is issued by a `Funder`, rather than a `Lease Agent`, i.e., during the execution of operation *Monthly Bill*, the invocation by `Guarantor` depends on the `Funder` support. This constraint is used when an entity can play as both `Lease Agent` and `Guarantor`. Obviously, if the entity pays the bill issued by itself, it may eventually do harm to `Customer`'s benifit. `Funder` as a financial provider can not play as a `Guarantor`. Hence, the `Guarantor` can pay the bill issued from `Funder` only. In a summary, `Guarantor` becomes ineffective if the operation *Monthly Bill* is supported by `Lease Agent`. Supposing `Customer` also can not be used due to another dependency restriction, the BP will be suspended since both users, `Customer` and `Guarantor`, become unavailable to invoke operation *Monthly Bill*.

Therefore, we can conclude that, the service consumers accesses and the resource supports are not only regulated by specific authorization policies to guide what the user can access and what the resource can support, but also restricted by the business

constraints enacted during the execution of business process to maintain the security in a process environment. These business constraints can be categorized as follows,

- **Synchronization**: The sequence of the service consumers accesses and the sequence of resource supports should both synchronize with the execution sequence of the operations in a composite web service. When an operation is ready to execute in a process instance, the relevant service consumers and resources that can access and support the operation should be invoked. Once the operation finishes, the permissions assigned to service consumers and resource to access and support the operation should be revoked immediately. The service consumer and resource that can access and support the next operation in the process instance then should be ready. In above example, when the operation *Guarantor Confirmation* starts, the authorization to access the operation should be granted to `Guarantor`. When the operation finishes, the authorization of `Guarantor` access on the operation should be revoked immediately.

- **Dependency**: A service consumer access or a resource support on specific operation in a composite web service may depend on another achieved service consumer access or resource support. In the above example, the access of the `Guarantor` on the operation of *Monthly Bill* depends on the support of the `Funder` on the same operation. If the `Funder` is used to support the *Monthly Bill*, then the `Guarantor` can be used to access the operation. Otherwise, the access of `Guarantor` on the operation of *Monthly Bill* should be restricted.

In a summary, to maintain the security during the execution of a composite web service as motivating scenario, the accesses of three service consumers, and the supports of four resources should be synchronously sequenced with the composite web service *Financial Lease*. Within the sequence of service consumer accesses and the sequence of resource supports, each access and support must also satisfy the dependency requirements to cater for business security demands. Existing works, e.g., [74, 75], mainly focus on managing synchronization between the execution sequence of the operations

and the sequence of service consumer accesses. They do not take the sequence of resources supports into account. Furthermore, much representative research works, e.g., [76], design constraints for avoiding conflict of interest within the service consumer accesses. Dependency constraint is still missing. Particularly, dependency constraints can not only be enabled between service consumer accesses, but can also be executed between resource supports, and even between service consumer access and resource support, which become more complex with the involvement of resource support management.

Hence, an authorization management based on the execution sequence of a composite web service should be developed to maintain the security in a process environment. In this chapter, two types of authorization policies, (1) Authorization Synchronization Policies and (2) Authorization Dependence Policies, are included to state the above business security demands. A process model SOAC-Net incorporated with SOAC is designed to represent authorization flow. An authorization flow is the sequence of service consumer accesses and the sequence of resources supports with associated authorization constraints, that is different from the control flow within normal workflow model, e.g., workflow-net [87]. The authorization flow can reflect the control flow of business process by enforcing synchronization policy. It can also enforce the dependency policy on top of the execution policy of the composite web service, which facilitates the authorization management on access control level only without delving into task execution sequence level.

The rest of this chapter is organized as follows. In section 2, both authorization synchronization policies and authorization dependency policies based on SOAC-Net are formally defined. In section 3, we will examine the SOAC-Net in depth, including its structure and execution policies. An example of execution of SOAC-Net will also be presented in this section. Conclusion will be presented in the last section.

5.2 Authorization Policies

It is still not enough for managing composite web service authorization, if we only know which service consumer can access the operation and which resource can be used to support the operation. We should also design policy to control how they can access and support the operations under a specific business logic. We define two types of Authorization Policies to control the access sequence of service consumers and the support sequence of resources, (1) *Authorization Synchronization Policies* and (2) *Authorization Dependence Policies*.

5.2.1 Authorization Synchronization Policies

The sequence of accesses and supports by role (a group of service consumers) and resource type (a group of component services) respectively on the operations of a composite web service must be consistent with the operation execution sequence, i.e. a role-flow and a resource type-flow must be synchronized with the control flow of operations. The reason is that, a service consumer as assigned role or a resource as mapped resource type can not be able to access or support the past operations (the operations that have been accessed or supported) unless the business logic of the composite web service allows, nor can they access or support the operations that are needed in the future but not now. Without this synchronization restriction on the role-flow and resource type-flow, the service consumer and the resource can obtain the right to access and support any operation without considering the well-defined execution sequence of the operations. Such chaos and disordered authorization could do harm to the whole execution of the composite web service, where the needed service consumer or resource is not available, or the service consumer and the resource do extra and unnecessary work causing security issue. Therefore, we need to devise synchronization policies to restrict the access sequence of role and the support sequence of resource type based on the control flow of the operations in a composite web service. Authorization synchronization policy is divided into two types, (1) Role Synchronization Policy and (2) Resource Type Synchronization Policy.

Constraint 1 ***Role Synchronization Policy****: Let* $\boldsymbol{OP}$ *be a set of Operations, and* $\boldsymbol{R}$ *be a set of roles. Let* $\boldsymbol{F}$ *be a set of execution sequences between operations,* $Op \rightarrow Op \in \boldsymbol{F}$*. (Notes,* $\rightarrow$ *represents the execution sequence of elements that will not repeatedly stated in the other constraints.) The role-flow is consistent with control-flow, if there exist operations* op_i *and* op_j*, whose execution sequence can be recorded as* $op_i \rightarrow op_j \in \boldsymbol{F}$*, then all roles that are mapped to* op_i *and all roles that are mapped to* op_j *must access the operations* op_i *and* op_j *based on the same execution sequence* $op_i \rightarrow op_j \in \boldsymbol{F}$*.*

In role synchronization policy, by setting up the sequence of role access to be consistent with the execution sequence of the mapped operations, the synchronization between the role-flow and the control-flow of operations is achieved. The role can not access the operation unless it starts to execute. When the execution of the operation finishes, the role access on the operation is revoked. For example, the `Guarantor` can start to access the operation *Guarantor Confirmation* when the operation can start to execute. When the operation *Guarantor Confirmation* finishes, the permission assigned to `Guarantor` to access the operation should be revoked immediately.

Constraint 2 ***Resource Type Synchronization Policy****: Let* $\boldsymbol{OP}$ *be a set of Operations, and* $\boldsymbol{ReT}$ *be a set of resource types. Let* $\boldsymbol{F}$ *be a set of execution sequences between operations,* $Op \rightarrow Op \in \boldsymbol{F}$*. The resource type-flow is consistent with control-flow, if there exist operations* op_i *and* op_j*, whose execution sequence can be recorded as* $op_i \rightarrow op_j \in \boldsymbol{F}$*, then all resource types that are mapped to* op_i *and all resource types that are mapped to* op_j *must support the operations* op_i *and* op_j *based on the same execution sequence* $op_i \rightarrow op_j \in \boldsymbol{F}$*.*

Resource type synchronization policy is used to synchronize the sequence of resource type supports with the execution sequence of operations. It requires that the resource type can support the operation only when it is the operation's turn to execute. When the execution of the operation finishes, the support of the resource type on the operation is revoked. For example, when the operation of *Lease Application* starts, the `Funder` should be available in terms of securing financial support amount. After the operation

finishes, the support from `Funder` should be revoked immediately to avoid the variation of the financial support amount made by the `Funder`.

5.2.2 Authorization Dependence Policies

Authorization Dependence Policies restrict that a role or a resource type is not able to access or support the operations, until the other role or the other resource type has already accessed or supported specific operations. Authorization Dependence Policies are separated into 5 categories, (1) between roles, (2) between resource types, (3) between roles and resource types, (4) between resource types and roles, and (5) between groups of roles and resource types (See Fig. 3.1). Here below, the 5 types of dependency policies are defined as follows,

Constraint 3 ***Role Dependency Policy*** **($\mathcal{C}_{r\to r}$):** *Let* ***OP*** *be a set of Operations, and* ***R*** *be a set of roles.* $OPA \subseteq \boldsymbol{R} \times \boldsymbol{OP}$ *is a set of relations of assigned roles on operations. A role dependency policy* $\mathcal{C}_{r\to r}$ *can be written as* $opa_a \to opa_b$*, where* opa_a *and* $opa_b \in OPA$*, and* opa_b *depends on* opa_a*, i.e. without* opa_a*,* opa_b *can not be used in role-flow.*

Role dependency policy ($\mathcal{C}_{r\to r}$) is used to restrict that the role access on specific operation depends on another role access in role-flow. This is different from role synchronization policy which is used to restrict the sequence of role accesses consistent with the execution sequence of operations. Formally, when $r_i \times op_i = opa_a$, and $r_j \times op_j = opa_b$, only after opa_a has been used in role-flow, then opa_b can be used. For example, in the motivating scenario at Chapter 1, the access of the `Guarantor` on the operation of *Monthly Bill* depends on the access of the `Guarantor` on the operation of *Lease Application*. It means that the `Guarantor` can repay the rental on behalf of `Customer` only if the *Lease Application* is also submitted by the `Guarantor`. A role dependency policy can be written as `Guarantor`$\times$***Lease Application***$\to$`Guarantor`$\times$***Monthly Bill***.

Constraint 4 ***Resource Type Dependency Policy*** **($\mathcal{C}_{ret\to ret}$):** *Let* ***OP*** *be a set of Operations, and* ***ReT*** *be a set of resource types.* $SPA \subseteq \boldsymbol{ReT} \times \boldsymbol{OP}$ *is a set of relations*

of assigned resource types on operations. A resource types dependency policy $\mathcal{C}_{ret \to ret}$ can be written as $spa_a \to spa_b$, where spa_a and $spa_b \in SPA$, and spa_b depends on spa_a, i.e. without spa_a, spa_b can not be used in resource type-flow.

Resource type dependency policy ($\mathcal{C}_{ret \to ret}$) bears the same semantics like role dependency policy ($\mathcal{C}_{r \to r}$). It is used to restrict the support of a resource type on a specific operation to depend on another support of resource type within resource type-flow. This is also different from resource type synchronization policy which is used to restrict the sequence of resource type support consistent with the execution sequence of the operations. Formally, when $ret_i \times op_i = spa_a$, and $ret_j \times op_j = spa_b$, only spa_a has been used in resource type-flow, then spa_b can be used. For example, in the motivating example at Chapter 1, the support of `Lease Agent` on the operation of *Monthly Bill* depends on the support of the `Lease Agent` on the operation of *Lease Application*. It means that `Lease Agent` can be used to issue bill only the `Lease Agent` is used to support the lease application. A resource type dependency policy can be written as `Lease Agent`$\times$***Lease Application***$\to$`Lease Agent`$\times$***Monthly Bill***.

Let us take another example. In the motivating scenario, the `Lease Agent` can be used to deal with the purchase of a product or the return of a product, if the lease application is processed by a `Lease Agent`. These two resource type dependency policies can be written as `Lease Agent`$\times$***Lease Application***$\to$`Lease Agent`$\times$***Return***, and `Lease Agent`$\times$***Lease Application***$\to$`Lease Agent`$\times$***Purchase***. Dependency policies can not only be used within role-flow and resource type-flow, but also be used between role-flow and resource type-flow. The role access on specific operation can depend on the support of a resource type, and vise versa.

Constraint 5 ***Role Resource Type Dependency Policy*** **($\mathcal{C}_{r \to ret}$):** *Let* ***OP*** *be a set of Operations,* ***R*** *be a set of roles, and* ***ReT*** *be a set of resource types.* $\boldsymbol{R} \times \boldsymbol{OP} \subseteq OPA$ *is a set of relations of assigned roles on operations, and* $\boldsymbol{ReT} \times \boldsymbol{OP} \subseteq SPA$ *is a set of relations of assigned resource types on operations. A role resource type dependency policy $\mathcal{C}_{r \to ret}$ can be written as $opa_a \to spa_a$, where $opa_a \in OPA$ and $spa_a \in SPA$, and spa_a depends on opa_a, i.e. without opa_a, spa_a can not be used in resource type-flow.*

Role resource type dependency policy ($\mathcal{C}_{r\rightarrow ret}$) is used to restrict a support of a resource type on specific operation to depend on a role access in role-flow. When $ret_i \times op_i = spa_a$, and $r_j \times op_j = opa_a$, only after the role r_j has been used to access the specific operation op_j, the resource type ret_i can then support the operation op_i. For example, in the motivating scenario at Chapter 1, the support of `Credit Assessor` on the operation of *Lease Confirmation* depends on the access of the `Guarantor` on the operation of the *Guarantor Confirmation*. It means that, the `Credit Assessor` can evaluate the credit history of `Customer` and `Guarantor` only after the `Guarantor` has confirmed its information. A role resource type dependency policy can be written as `Guarantor`$\times$***Guarantor Confirmation***$\rightarrow$`Credit Assessor`$\times$***Lease Confirmation***.

Constraint 6 ***Resource Type Role Dependency Policy*** **($\mathcal{C}_{ret\rightarrow r}$):** *Let* $\boldsymbol{OP}$ *be a set of Operations,* $\boldsymbol{R}$ *be a set of roles, and* $\boldsymbol{ReT}$ *be a set of resource types.* $\boldsymbol{R}\times\boldsymbol{OP}\subseteq OPA$ *is a set of relations of assigned roles on operations, and* $\boldsymbol{ReT}\times\boldsymbol{OP}\subseteq SPA$ *is a set of relations of assigned resource types on operations. A resource type role dependency policy* $\mathcal{C}_{ret\rightarrow r}$ *can be written as* $spa_a \rightarrow opa_a$*, where* $opa_a \in OPA$ *and* $spa_a \in SPA$*, and* opa_a *depends on* spa_a*, i.e. without* spa_a*,* opa_a *can not be used in role-flow.*

Resource type role dependency policy ($\mathcal{C}_{ret\rightarrow r}$) is a reverse policy of role resource type dependency policy, in which it enables the role access on specific operation to depend on the support of a resource type. When $ret_i \times op_i = spa_a$, and $r_j \times op_j = opa_a$, only after the resource type ret_i has been used to support the specific operation op_i, the role r_j can then access the operation op_j. In the motivating scenario, the access of `Guarantor` on the operation *Monthly Bill* depends on the support of `Funder` on the operation *Monthly Bill*. It means that the operation *Monthly Bill* can be accessed by the `Guarantor`, only after the bill has been issued by the `Funder`, rather than `Lease Agent`. A resource type role dependency policy can be written as `Funder`$\times$***Monthly Bill***$\rightarrow$`Guarantor`$\times$***Monthly Bill***.

All dependency policies defined until now are used to restrict role access and resource type support from single element's point of view, e.g., a role access depends on

another role access. However, they are lack of capability to describe the dependency restriction between the groups of role accesses and resource type supports, where a role and a resource type are grouped and their access and support depends on the access and support of another grouped role and resource type.

Constraint 7 ***Group of Role and Resource Type Dependency Policy*** **($\mathcal{C}_{r\times ret\rightarrow r\times ret}$):** *Let* $\boldsymbol{OP}$ *be a set of Operations,* $\boldsymbol{R}$ *be a set of roles, and* $\boldsymbol{ReT}$ *be a set of resource types.* $\boldsymbol{R}\times\boldsymbol{OP}\subseteq OPA$ *is a set of relations of assigned roles on operations, and* $\boldsymbol{ReT}\times\boldsymbol{OP}\subseteq SPA$ *is a set of relations of assigned resource types on operations. A group of role and resource type dependency policy* $\mathcal{C}_{r\times ret\rightarrow r\times ret}$ *can be written as* $(opa_a, spa_a) \rightarrow (opa_b, spa_b)$*, where* opa_a *and* $opa_b \in OPA$*,* spa_a *and* $spa_b \in SPA$*, and the group of* opa_a *and* spa_a *depends on the group of* opa_b *and* spa_b*, i.e. without both* opa_a *and* spa_a*,* opa_b *and* spa_b *can not both be used in role-flow and resource type-flow, respectively.*

Group of role and resource type dependency policy ($\mathcal{C}_{r\times ret\rightarrow r\times ret}$) (See Fig. 3.1) is used to restrict a role access and a resource type support to depend on the access and support of another group of role and resource type. A role and a resource type can be grouped when they are both working on the same operation. When $ret_i\times op_i=spa_a$, $ret_j\times op_j=spa_b$, $r_i\times op_i=opa_a$, and $r_j\times op_j=opa_b$, only the role r_i and the resource type ret_i have both been used to access and support the specific operation op_i, the role r_j and the resource type ret_j can then be both used to access and support the operation op_j. In the motivating scenario at Chapter 1, the access of `Lawyer` and the support of `Lease Agent` on the operation of *Return* depends on the access of `Customer` and the support of `Lease Agent` on the operation of *Lease Application*. It means that, if `Lawyer` and `Lease Agent` are together to deal with the *Return* of the product, i.e, the none real supplier of the product and the none real user of the product, then the lease application must be submitted directly by the `Customer` and the product should be provided by the `Lease Agent`. The application submitted by the applicant in person can lower the risk of dispute on the product of return based on an assumption that, the more the `Customer` involves, the less the dispute can occur. A group of role and resource type dependency policy can be written as <`Customer`×***Lease Application***, `Lease`

Agent×***Lease Application***>→<Lawyer×***Return***, Lease Agent×***Return***>.

5.3 Specification of SOAC-Net

With the support of resource, SOAC is still not enough to manage composite web service authorization in process environment. The operations of the composite web service need to be accessed and supported by the service consumers and the component services according to an execution sequence, i.e., the business logic of composite web service. We need to extend our question by adding "How to do" on top of the previous question "what to do". The service consumer and the resource should not only understand what can access and support, but also need to find out the sequence of access and support based on the business logic of operations.

Therefore, we develop a process model named as SOAC-Net to represent the authorization flow, where the two types of authorization policies defined in last section can be enforced. SOAC-Net is divided into three parts, role-flow, resource type-flow, and constraint-flow. In SOAC-Net, a role-flow and a resource type-flow are derived from the execution sequence of the operations and associated authorization mappings, i.e. the mappings between role and operation, and the mappings between resource type and operation. These two types of flows are tightly synchronized with the execution sequence (control flow) of an operation. But they are not the same as the control flow since they have capability to manage the access sequence of role and the support sequence of resource type within one operation, where the control flow can only be used to coordinate the execution sequence between operations. Constraint-Flow is used to represent the various authorization dependency policies based on the role-flow and resource type-flow.

A role-flow is defined as a sequence of role accesses on specific operations based on the entire control flow and is constructed based on two steps (as shown in Fig. 5.1). Please note, the rolw-flow can not only deal with sequential execution sequence, but also can handle the other types of control flow patterns.

Step 1 The accesses of the roles on the operations is sequenced based on the operation

execution sequence, where the roles that can access the same operation are put together. The role access on operation can be obtained from the given authorization mappings, i.e. the mapping between the role and the operation (See Line 5 in Algorithm 1).

Step 2 If the relationship of the roles put together for accessing the same operation is *exclusive choice*, then *role or-split* and *role or-join* are used to set up the accesses of these roles on the operation into different branches. One of the branches will be selected during the role-flow execution. On the other hand, if the multiple roles need to concurrently access the same operation, *role and-split* and *role and-join* are used to divide these role accesses on the operation into different branches that execute concurrently. These four operators can be used as nested structure. By using this construction method, a role-flow is developed that can not only synchronize the execution sequence of the operations, but can also manage the role accesses within one operation (See Line 7 in Algorithm 1).

Let us take an example. In Fig. 5.1, the execution sequence of operations and the associated authorization mappings between roles and operations are given before step 1 to describe the motivating scenario. Then based on the provided information, the roles that can access the same operations are put together as the execution sequence of the operations that they can access (See the roles in clouds in Fig. 5.1 after step 1.). We can observe that the sequence of role access is matched with the sequence of operation execution after step 1 in Fig. 5.1. After step 2, the access pattern of the role that can access the same operation is determined as *exclusive choice* or *Concurrency*. Therefore, a role-flow is finally constructed as the diagram in Fig. 5.1 after step 2.

In role-flow, the operation will be used as a token to move from one node to another one based on the execution sequence of composite web service. Each node in role flow represents a specific role access on a specific operation. It means that, when a specific operation is required to be executed by the business logic of a composite web service, the token as the operation will be flowed to specific nodes that can access the operation. When the operation finishes its executing, the token as the specific operation

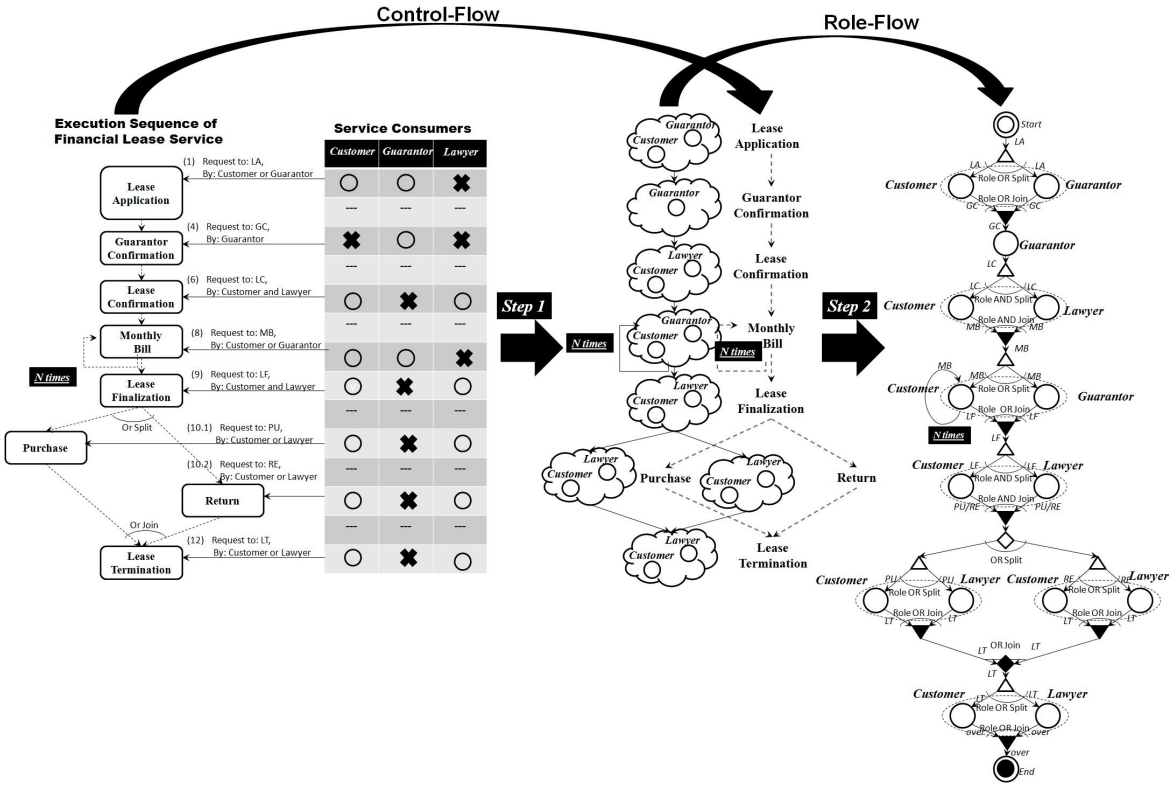

FIGURE 5.1: Construction Process of Role-Flow

will continue to flow to the next nodes that require to access the next operation in the business process according to the business logic of a composite web service. The synchronization between role-flow and control flow of the operations can then be ensured. Through this way, the authorization management system can control when the role can be used to access the specific operation based on the business logic. Here below, we present an algorithm to construct role-flow.

A resource type-flow is defined as a set of resource type supports on specific operations, and its construction method is similar to the method of constructing a role-flow, It also needs two steps (as shown in Fig. 5.2).

Step 1 The supports of the resource type on the operations are sequenced based on the operation execution sequence, where the resource types that can support the same operation are put together. The resource type supports on operation can

be obtained from the given authorization mappings, i.e. the mapping between the resource type and the operation (See Line 5 in Algorithm 2).

Step 2 If the relationship of the resource types that are put together for supporting the same operation is *exclusive choice*, then *resource type or-split* and *resource type or-join* are used to set up the supports of these resource types on the operation into different branches. One of the branches will be selected during the resource type-flow execution. On the other hand, if the multiple resource types need to concurrently access the same operation, *resource type and-split* and *resource type and-join* are used to divide these resource type supports on the operation into different branches that execute concurrently. These four operators can be used as nested structure. By using this construction method, a resource type-flow is developed that can not only synchronize the execution sequence of the operations, but can also manage the resource types supports within one operation (See Line 7 in Algorithm 2).

Let us also take an example of the motivating scenario. In Fig. 5.2, the execution sequence of operations and the associated authorization mappings between resource type and operations are given before step 1 to describe the motivating scenario. Then based on the provided information, the resource types that can support the same operations are put together as the execution sequence of the operations (See the resource types in clouds in Fig. 5.2 after step 1.). We can observe that the sequence of resource types supports is matched with the sequence of operation execution after step 1 in Fig. 5.2. After step 2, the support pattern of resource type that can support the same operation is determined as *exclusive choice* or *Concurrency*. Therefore, a resource type-flow is finally constructed as the diagram in Fig. 5.2 after step 2. The execution sequence of the operations used to construct resource type-flow is different with the execution sequence of the operations used to construct role-flow, since some of the operations in the motivating scenario do not require the support from component web services.

In resource type-flow, an operation is also used to pass from one resource type support to the other depending on when the specific operation should be supported

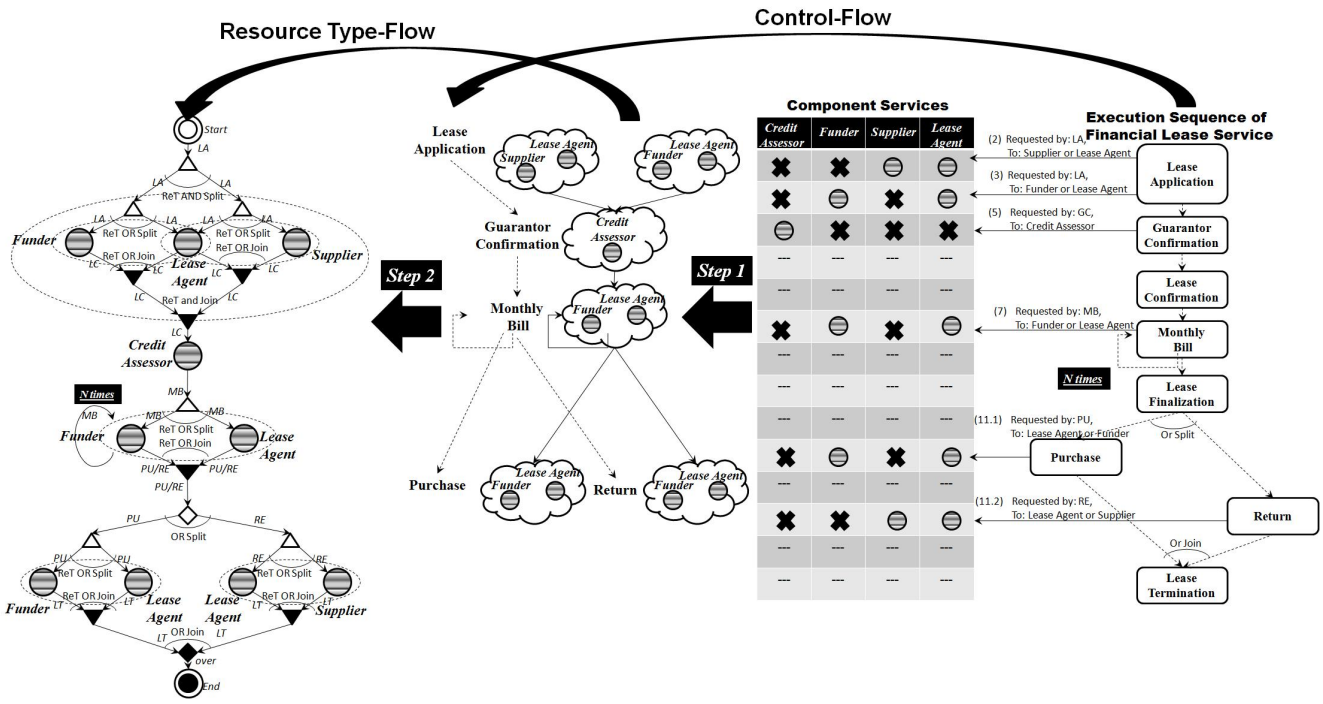

FIGURE 5.2: Construction Process of Resource Type-Flow

by a specific resource type. This can ensure the synchronization between resource type-flow and control flow of the operations. Here below, we present an algorithm to construct role-flow.

Constraint-flow is constructed based on the authorization dependency policies. An constraint node is used to link role accesses or resource type supports depending on the specific dependency policy that the constraint node represents. The incoming link of the constraint node comes from the depended role access or resource type support; while the outgoing link of the constraint node points to the depending ones. When the depended role access or resource types support finishes dealing with the specific operation, the constraint node will be activated by the incoming link. Through the outgoing link of the constant node, the depending role access or resource type support can be used on specific operations when they are needed. Otherwise, without the activation of the linked constraint nodes, they are not available.

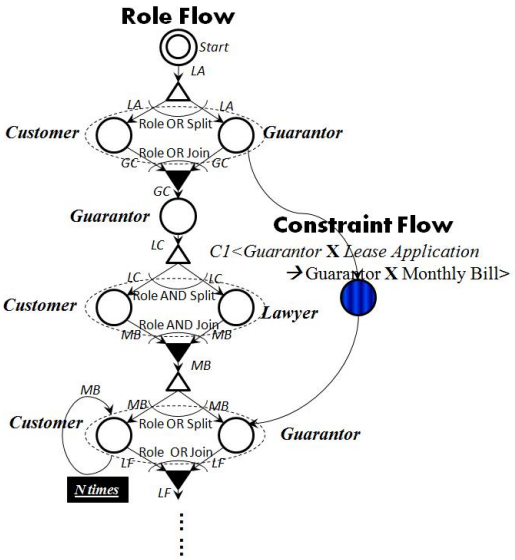

Figure 5.3: Constraints-Net:C1

In the motivating scenario, a role dependency policy $(\mathcal{C}1_{r\to r})$ is defined as `Guarantor` $\times$***Lease Application***$\to$`Guarantor`$\times$***Monthly Bill***, where the `Guarantor` can access the operation of *Monthly Bill* only if the *Lease Application* is also accessed by the *Guarantor*. This policy can be illustrated in constraints-flow as Fig. 5.3. In Fig. 5.3, a node with blue horizon stripes is used as constraint node to represent the policy. The incoming link of this node comes from the node in role-flow that represents the `Guarantor` access on the operation of *Lease Application*. The outgoing link of this constraint node points to the node in role-flow that represents the `Guarantor` access on the operation of *Monthly Bill*. Therefore, when the node in role-flow representing the `Guarantor` access on the *Lease application* finishes its activation, i.e. the `Guarantor` has been used to access the *Lease Application*, the related constrain node in constraints-flow is activated. It means that the node in role-flow representing the `Guarantor` access on the operation of *Monthly Bill* and pointed by the constraint node can be able to be activated. Note, it is also acceptable that the operation of *Monthly Bill* is accessed by another role-`Customer`. allowing the `Guarantor` to access the *Monthly Bill* does not mean that the operation of *Monthly Bill* must be accessed by the `Guarantor`. Under

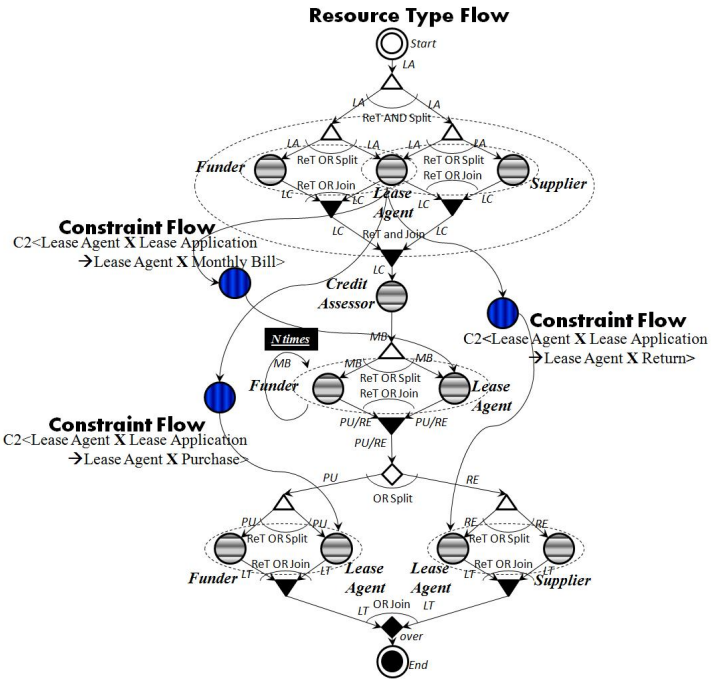

FIGURE 5.4: Constraints-Net:C2

this policy, the `Customer` who can also access the *Monthly Bill* is welcome as well.

In the motivating scenario, three resource type dependency policies $(\mathcal{C}2_{ret \to ret})$ are defined as,

1. `Lease Agent`$\times$***Lease Application***$\rightarrow$`Lease Agent`$\times$***Monthly Bill***;
2. `Lease Agent`$\times$***Lease Application***$\rightarrow$`Lease Agent`$\times$***Return***;
3. `Lease Agent`$\times$***Lease Application***$\rightarrow$`Lease Agent`$\times$***Purchase***.

These policies are illustrated in Fig. 5.4. In Fig. 5.4, three constraint nodes representing the above three authorization dependency policies respectively are used to link the specific nodes in resource type-flow. For example, the incoming link of the node used to represent the first policy comes from the node representing the support of `Lease Agent` on the operation of *Lease Application*. Its outgoing link points to the node in resource type-flow that represents the support of `Lease Agent` on *Monthly Bill*. The outgoing links of the nodes representing the second and the third policies point

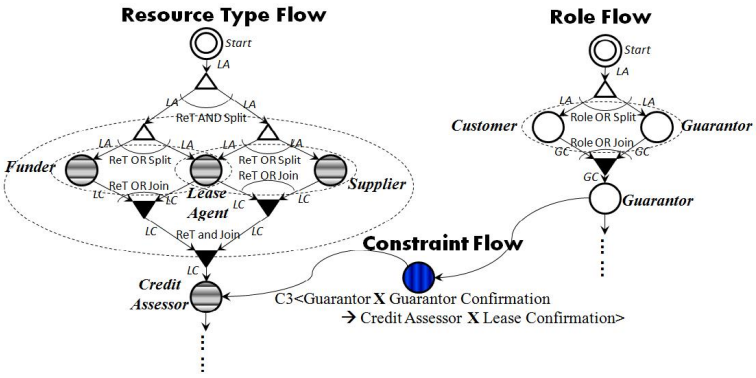

Figure 5.5: Constraints-Net:C3

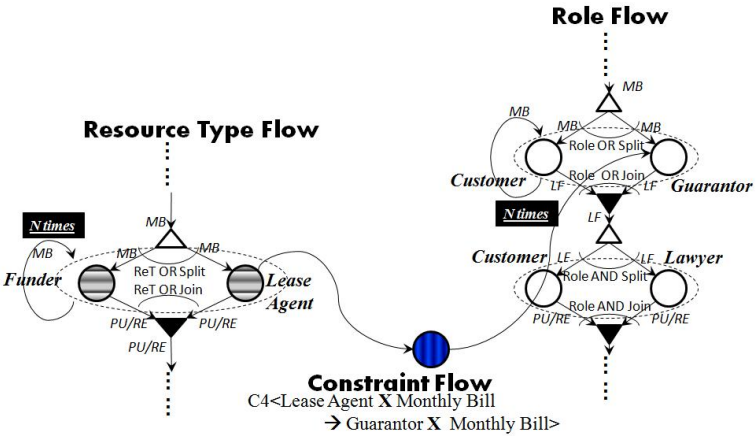

Figure 5.6: Constraints-Net:C4

to the nodes in resource type-flow that represent the support of `lease agent` on the operations of *Purchase* and *Return*, respectively. Their incoming links are the same as that of node representing the first policy. By enforcing these policies in constraint-net, the supports of `Lease Agent` on *Monthly Bill*, *Purchase*, and *Return*, can be implemented based on a precondition that the support of `Lease Agent` on the operation *Lease Application* has already been performed.

In Fig. 5.5, we use the constraint node to represent the role resource type authorization dependency policy *($\mathcal{C}3_{r\rightarrow ret}$)*. In the motivating scenario, a role resource type dependency policy is defined as `Guarantor`$\times$***Guarantor Confirmation***$\rightarrow$`Credit`

`Assessor`$\times$***Lease Confirmation***. It restricts that only after the *Guarantor Confirmation* has been accessed by the `Guarantor`, i.e. the information of the `Guarantor` has been collected, the support of `Credit Assessor` on the operation *Lease Confirmation* then can be performed. In Fig. 5.5, the incoming link of the constraint node comes from the node in role-flow that represents the `Guarantor` access on the operation of *Guarantor Confirmation*. Its outgoing link points to the node in resource type-flow that represents the support of `Credit Assessor` on the operation of *Lease Confirmation*. Therefore, the `Credit Assessor` can not be used to support the *Lease Confirmation* unless the related constraint node is activated by the execution of the node in role-flow representing the `Guarantor` access on the operation of *Guarantor Confirmation*.

In Fig. 5.6, a resource type role authorization dependency policy *($\mathcal{C}4_{ret\rightarrow r}$)* in the motivating scenario is illustrated, where if the `Guarantor` can access the operation of *Monthly Bill* to repay the rental depends on if the bill is issued by `Lease Agent`. We can observe in Fig. 5.6 that a constraint node is used to link the nodes in role-flow and resource type-flow, respectively. The incoming link of the constraint node comes from the node that represents the support of `Lease Agent` on operation of *Monthly Bill*. The outgoing link of the constraint node points to the `Guarantor` access on *Monthly Bill*. Hence, without the activation of the `Lease Agent`'s support on *Monthly Bill*, the `Guarantor` access on *Monthly Bill* can not come to true by enforcing this policy.

Fig. 5.7 presents a new type of authorization dependency policy in the motivating scenario, i.e., Group of Role and Resource Type Dependency Policy*($\mathcal{C}5_{r\times ret\rightarrow r\times ret}$)*, where the `Lease Agent` and `Lawyer` can deal with (support and access) the operation of *Return* under the condition that the operation of *Lease Application* has already been handled (supported and accessed) by `Lease Agent` and `Customer`. In Fig. 5.7, the incoming link of the constraint node that represents the dependency policy C5 comes from both nodes in role-flow and resource type-flow, which represent the `Lease Agent` support and the `Customer` access on the operation of *Lease Application*, respectively. The outgoing link of that constraint node points both the nodes representing the support of `Lease Agent` and the access of `Lawyer` on the operation of `Return`. By

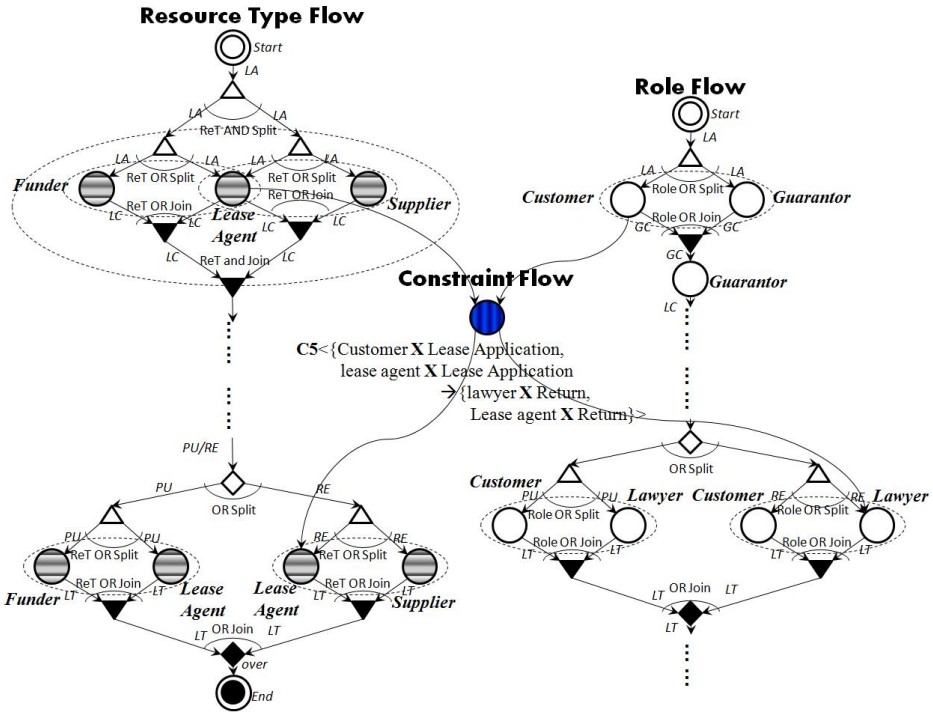

Figure 5.7: Constraints-Net:C5

enforcing this authorization policy, the operation *Return* can be supported by `Lease Agent` and can be accessed by `Lawyer` under the condition that they are not both used simultaneously to handle the operation of `Return`. If they have to, the precondition that the operation *Lease Application* which is dealt by `Customer` and `Lease Agent` must be satisfied firstly.

In summary, the construction processes of role-flow and resource type-flow should comply with the authorization synchronization policy, which makes role accesses and resource type supports follow the business logic of a composite web service, i.e., *when the role and resource type can access and support the operation respectively* is determined. The constraint-flow is classified into five categories and used to restrict the role access and resource type supports based on the authorization dependency policy, i.e. *which role and resource type can be used to access and support the operation* is decided. In Fig. 5.8, we present a complete view on SOAC-Net.

Therefore, based on the example above, we can understand that in order to properly

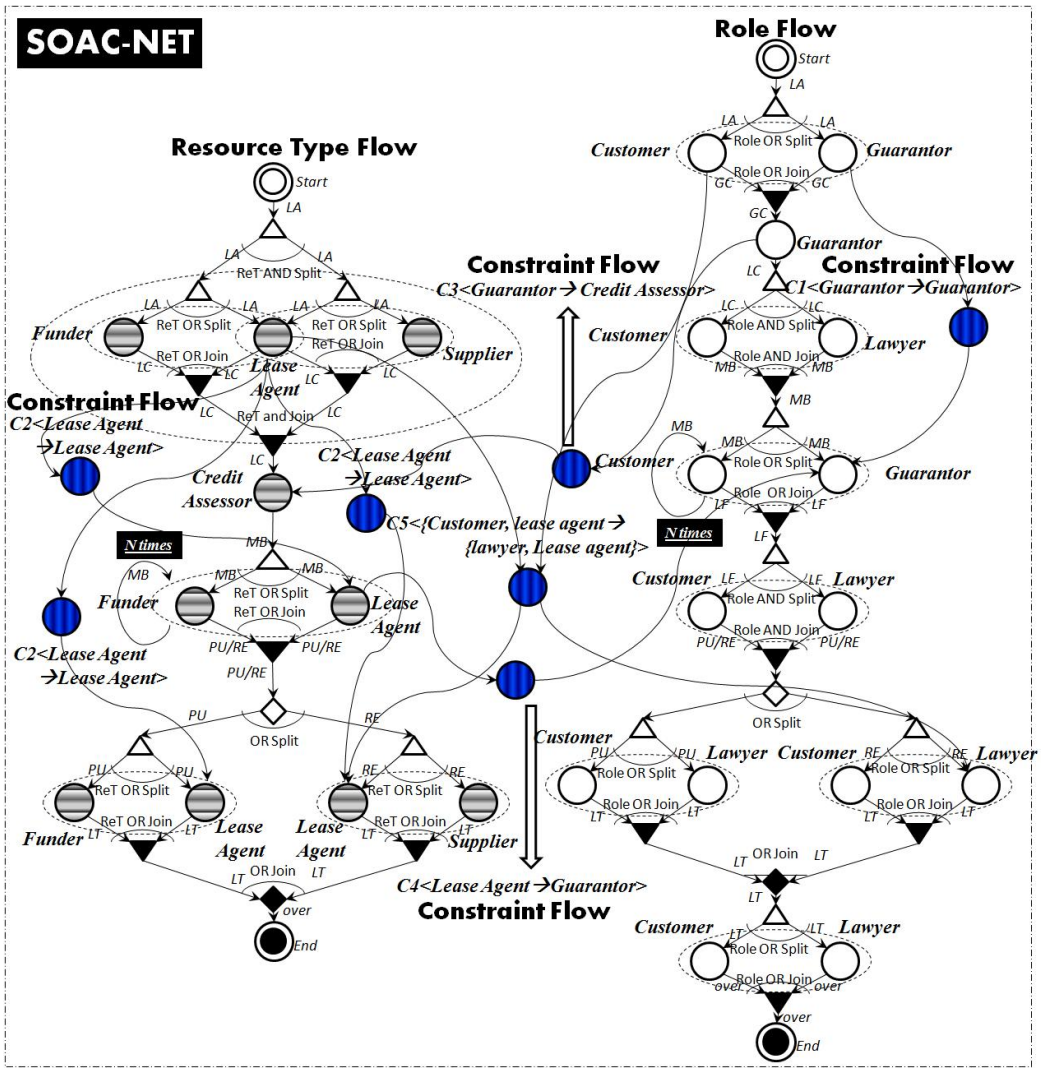

FIGURE 5.8: SOAC-Net

manage composite web service authorization, we need to develop a model which can explicitly present the authorization flow, capture the authorization policies and enforce them during the execution of the composite web service. The advantages of Petri-Net's graphically and mathematically founded modeling formalism with various algorithms for design and analysis make it a good candidate for modeling authorization flow in composite web services. Here, we design a Petri-Net based process model named SOAC-Net, to enforce the various types of authorization policies in composite web service authorization. SOAC-Net is separated into three parts, Role-Net, Resource Type-Net, and Constraints-Net. Each part is used to model the relevant authorization flow.

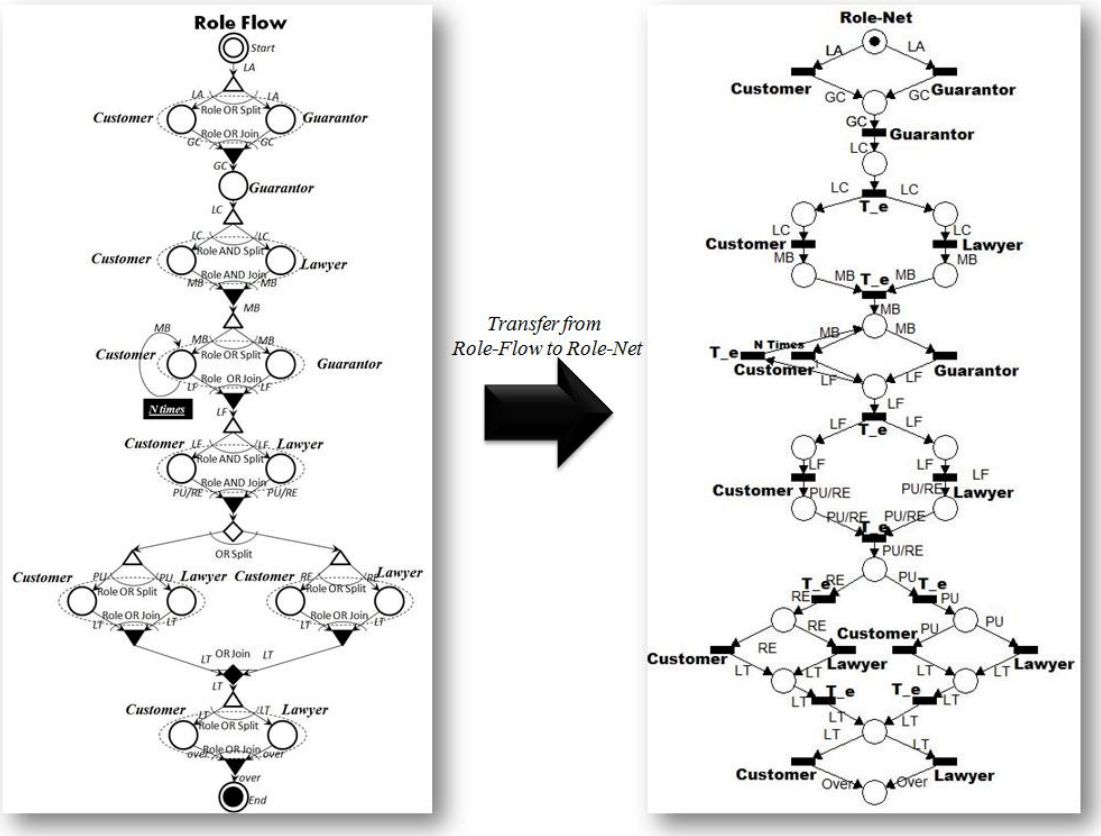

Figure 5.9: From Role-Flow to Role-Net

For example, role-flow is represented by role-net, resource type-flow is represented by resource type-net, and constraint-net in SOAC-Net models the constraint-flow. By integrating the three parts, SOAC-Net as an infrastructure can be used to enforce both the authorization synchronization policies and authorization dependency policies. In the next section, we will present the structure of SOAC-Net in details.

5.3.1 Structure of SOAC-Net

Role-Net

Role-Net is used to enforce the **Role Synchronization Policy**. In role-net (See Fig. 5.9), we use each transition (the black rectangle) to represent a role access on specific operation. We use token that flows between the transitions in role-net to

represent the operations. When the transition consumes one token, it means that the operation (token) has been accessed by the role (the transition). The new generated tokens by this transition will represent the next operation that need to be accessed according to the business logic of the composite web service. Fig. 5.9 illustrates how to transfer the role-flow into role-net. Since each node in role-flow is absolutely matched with the transitions in role-net, we can conclude that the semantics of role-flow is inherently passed to role-net. Since the role synchronization policy can be enforced in role-flow, it can also be enforced in role-net to restrict on when the role can be used to access the specific operation. It is formally defined as follows,

Definition 12 *Role-Net is a tuple* $\mathcal{N} = (P^r, T^r, F^r, i^r, o^r)$

- P^r *is a set of places, graphically represented as circles,*
- T^r *is a set of transitions, graphically represented as black bars. Transition is used to represent role access on specific operation in role-net.* $T^e \subset T^r$ *is a set of empty transitions as or-split, or-join, and-split, or and-join,*
- $F^r = (P^r \times T^r) \cup (T^r \times P^r)$,
- i^r *and* o^r *are input place and output place, respectively, to initially deposit a token and finally collect token in role-net.*

Resource Type-Net

Resource Type-Net is used to enforce the **Resource Type Synchronization Policy**. Based on the same theory of Role-Net, the resource type should follow a specific sequence to provide support to the operations of the composite web service. In Resource Type-Net (See Fig. 5.10), we use each transition to represent a resource type support on a specific operation. The token that flows between the transitions is used to represent each operation that the resource type (transition) supports. Through this way, when the resource type can be used to support the operation during the execution sequence of the operations can be clearly captured. Therefore, we should not only know what resource type can be granted to the specific resource, but also can perceive

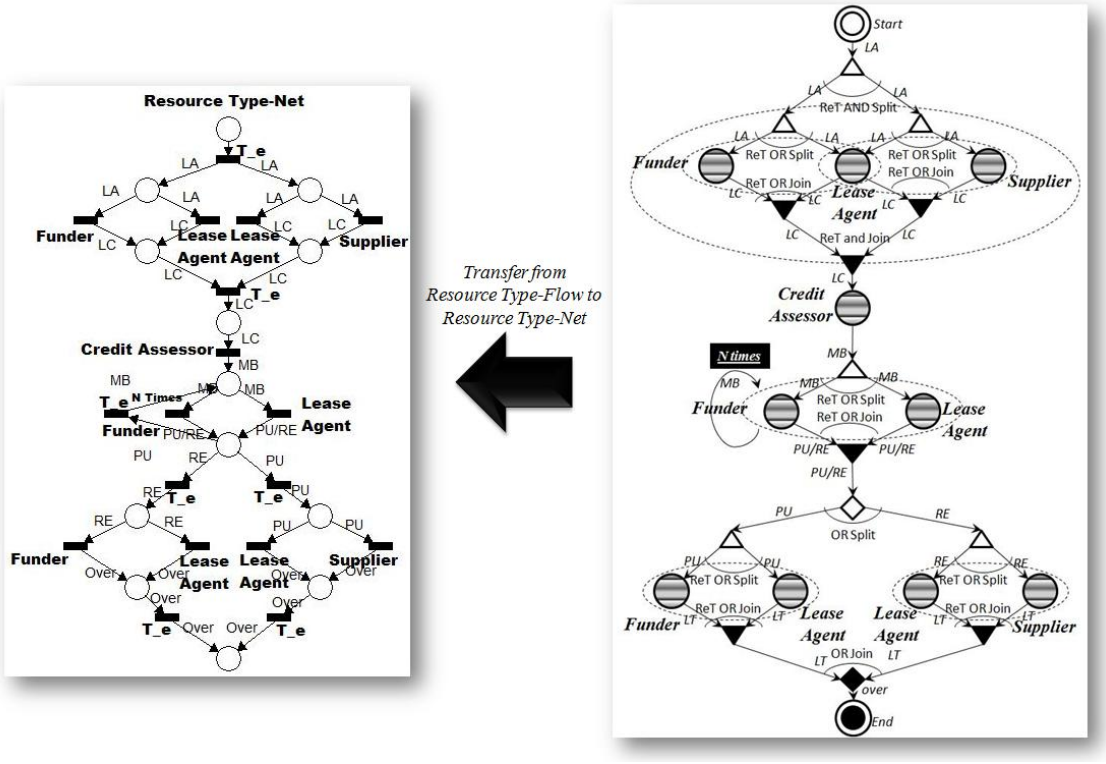

Figure 5.10: From Resource Type-Flow to Resource Type-Net

how they can be used, i.e., based on what kind of sequence. Fig. 5.10 illustrates how the Resource Type-Net can represent the resource type-flow. The formal definition of resource type-net is defined as follows,

Definition 13 *Resource Type-Net is a tuple* $\mathcal{O} = (P^{ret}, T^{ret}, F^{ret}, i^{ret}, o^{ret})$,

- P^{ret} *is a set of places, graphically represented as circles,*
- T^{ret} *is a set of transitions, graphically represented as black bars. Transition is used to represent resource type support in resource type-net.* $T^{e} \subset T^{ret}$ *is a set of empty transitions as or-split, or-join, and-split, or and-join,*
- $F^{ret} = (P^{ret} \times T^{ret}) \cup (T^{ret} \times P^{ret})$,
- i^{ret} *and* o^{ret} *are input place and output place respectively, to initially deposita token and finally collect token in resource type-net.*

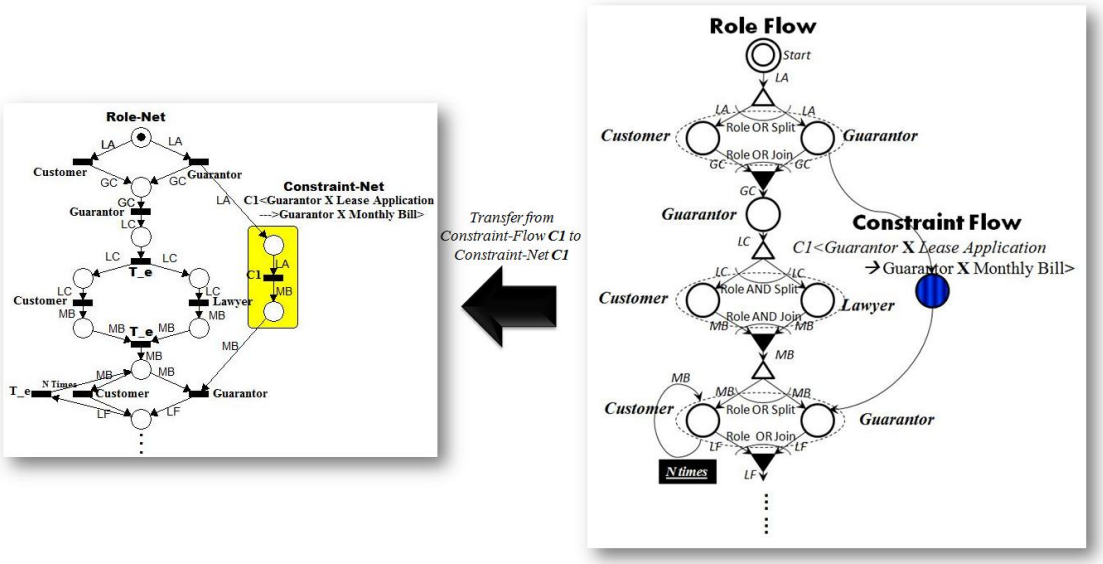

FIGURE 5.11: From Constraint-Flow C1 to Constraint-Net C1

Constraints-Net

Constraints-Net is used to enforce the five **Authorization Dependency Policies**. For policies $\mathcal{C}1_{r\to r}$, $\mathcal{C}2_{ret\to ret}$, $\mathcal{C}3_{r\to ret}$, and $\mathcal{C}4_{ret\to r}$, single transition is used to represent each policy (See Fig. 5.11, Fig. 5.12, Fig. 5.13, and Fig. 5.14 as the examples from the motivating scenario). Each transition in constraint-net is linked from one transition in role-net or resource type-net and pointed to another transition in role-net or resource type-net. Hence, without token movement through the transition in constraints-net, the relevant role-net or resource type-net which is pointed by the transition in constraints-net can not accumulate enough tokens to fire, according to the basic execution policy of Petri-Net. We can use this method to realize the dependency between roles, between resource types, even between role and resource type.

For example, in the **Resource Type Role Dependency Policy** *($\mathcal{C}4_{ret\to r}$)* in Fig. 5.14, the incoming link of the transition is from resource type-net and the outgoing link of the transition points to a transition in role-net. Therefore, without performing the support of relevant resource type, no token can be passed to the role-net through the transition in constraints-net. The pointed transition in role-net can not be fired, i.e.

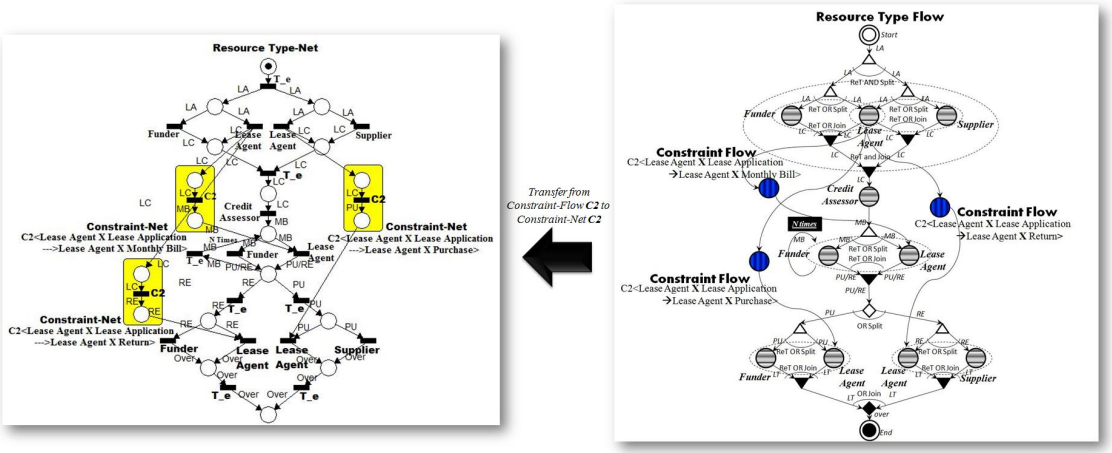

Figure 5.12: From Constraint-Flow C2 to Constraint-Net C2

this role can not be used to access the specific operation. The dependency relationship of the role access on that resource type support is realized by this resource type role dependency policy.

The **Group of Role and Resource Type Dependency Policy** ($\mathcal{C}5_{r\times ret\rightarrow r\times ret}$) is different from the other four dependency policies (See Fig. 5.16). In constraint-flow C5 in Fig. 5.7, the incoming links of the constraint node come from the nodes in role-flow and resource type-flow respectively, and the outgoing links of the constraint node also point to the nodes in role-flow and resource type-flow. The semantics of the constraint node in constraint-flow C5 is, that the role access and the resource type support pointed by the constraint node can be freely executed without the restriction of the authorization policy, when only one of them is performed. If both of the role access and the resource type support pointed by the constraint node require to run (They do not have to run simultaneously. They can be required to run one by one), then both the role access and the resource type support linked to the constraint node through incoming link must have already been performed firstly.

Hence, we can not only use one transition to represent the constraint node in

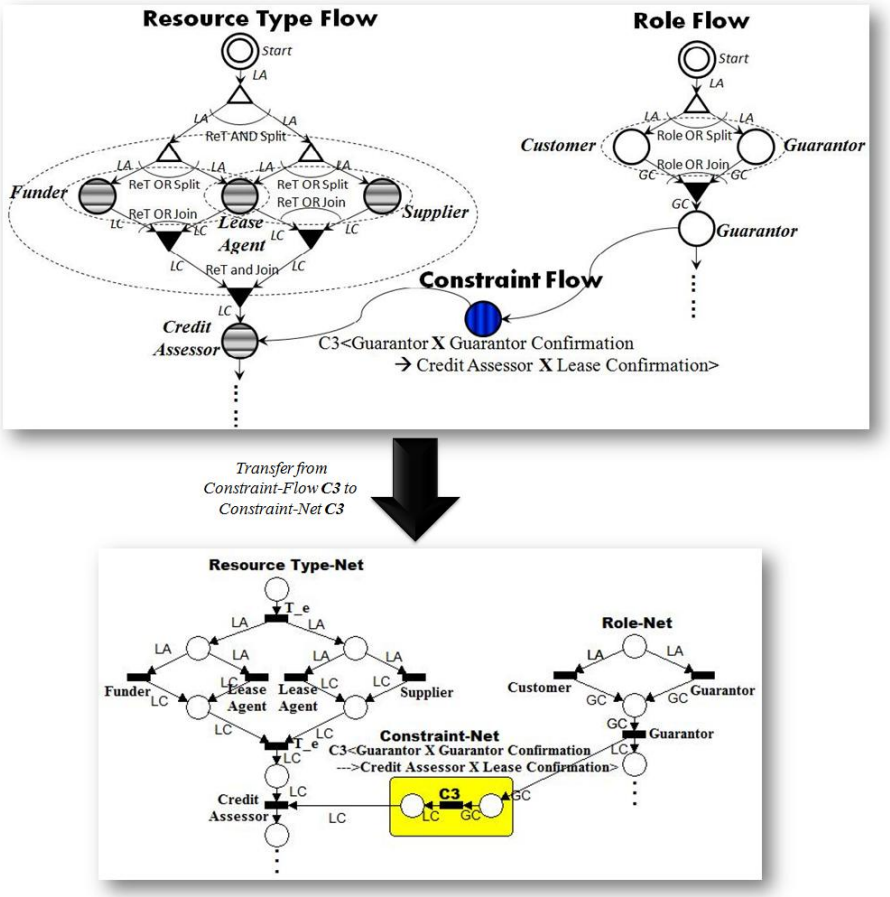

FIGURE 5.13: From Constraint-Flow C3 to Constraint-Net C3

constraint-flow C5 in Fig. 5.7. For example, in Fig. 5.15, the execution of the group of role access C (Depending Role Access (Ding-RA)) and resource type support D (Depending Resource Type Support (Ding-RTS)) depends on the execution of the group of the role access A (Depended Role Access (Ded-RA)) and resource type support B (Depended Resource Type Support (Ded-RTS)). All four transitions representing Ding-RA, Ding-RTS, Ded-RA, and Ded-RTS are linked to the single transition *C5* in constraint-net that tries to represent the Group of Role and Resource Type Dependency Policy $(\mathcal{C}5_{r\times ret\rightarrow r\times ret})$. However, in this case, we can find that even the role access C or the resource type support D needs to run individually, both role access A and resource type support B are required to be executed firstly. This is not consistent with the semantics of the Group of Role and Resource Type Dependency Policy, which is only

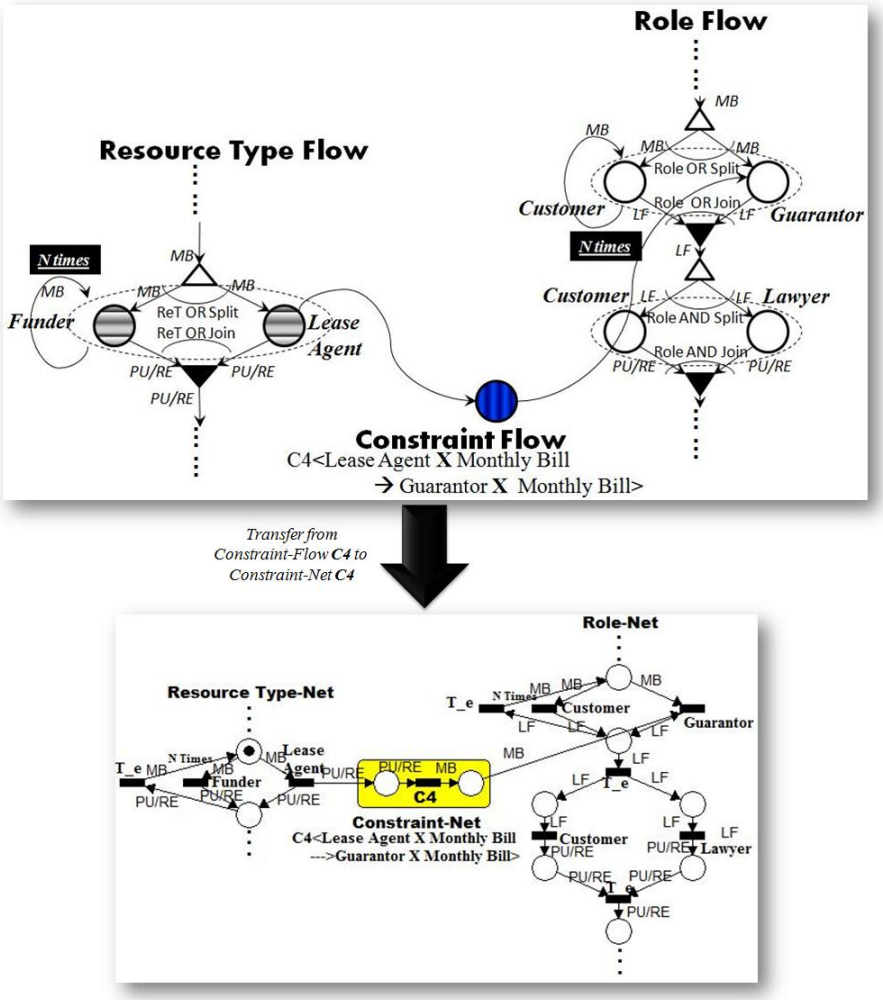

Figure 5.14: From Constraint-Flow C4 to Constraint-Net C4

performed when both of the role access C and the resource type support D require to run.

Therefore, in this type of authorization dependency policy, the transition is separated into three types (See Fig. 5.16).

- T_e is a transition used as *and-split* to pass the token that comes from the Ding-RA or Ding-RTS that has been executed, into different paths in constraint-net C5.

- T_x is a transition, whose execution is used to notify Ding-RA or Ding-RTS that has not been executed, that its partner in the same group has been executed. It

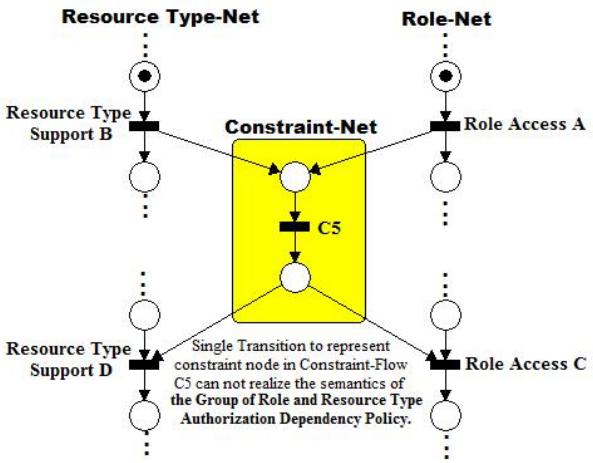

FIGURE 5.15: Single Transition Can Not Be Used in C5

means that, after T_x is executed, the token deposited at post-place of T_x makes it become one of pre-places of transition that represents Ding-RA or Ding-RTS which does not need to run. Hence, after the execution of T_x, the other Ding-RA or Ding-RTS can not be freely executed, if its pre-place in constraint-net C5 can not accumulate enough tokens. The restriction starts to constrain Ding-RA or Ding-RTS that has not been executed yet.

- The transition T_y is a transition that can be enabled by accumulating two tokens from the executions of Ded-RA or Ded-RTS respectively. Hence, the execution of T_y represents that both Ded-RA and Ded-RTS have been successfully executed. The token deposited at post-place of T_y (also known as the post-place of T_x) then makes the place accumulate enough tokens to enable the transition in Ding-RA or Ding-RTS that wants to perform and is restricted by policy C5. In summary, the restriction on the Ding-RA or Ding-RTS is released after post-place of T_x accumulates enough tokes (one from T_x and one from T_y).

The execution mechanism of constraint-net C5 will be introduced in details in the next section. In summary, the constraint-net C5 needs to check two indicators. The first indicator is used to examine if either Ded-RA or Ded-RTS has been executed. T_y is designed for this purpose. This indicator can decide if the restriction of the policy

can be released for Ding-RA or Ding-RTS. The second indicator is used to announce that one of the Ding-RA and Ding-RTS has been executed. Then the other Ding-RA or Ding-RTS that has not been executed needs to be restricted by the policy unless both Ded-RA or Ded-RTS has been already performed. T_x is therefore designed as the second indicator. In constraint-net C5, T_e is the transition used to coordinate the execution of the two indicators. The formal definition of constraint-net is defined as follows,

Definition 14 *Constraints-Net is a tuple* $\mathcal{C} = (P^c, T^c, F^c, W, Count)$*, Where:*

- P^c *is a set of places, graphically represented as circles,*
- T^c *is a set of transitiond, graphically represented as black bars. Transition is used to represent a constraint in Constraints-net C1 to C4 (See Fig. 5.3 to Fig. 5.6). For constraint-net C5, transition is separated as follows,*
  - $T^e \subset T^c$ *is a set of empty transitions as and-split to simultaneously split the token movement pathes to* $T_x \subset T^c$ *and* $T_y \subset T^c$*.*
  - $T_x \subset T^c$ *represents an indictor to show, when Group B depends Group A, if a role or a resource type in group B has already been used to deal with specific operation, i.e., if the constraint should be enabled.*
  - $T_y \subset T^c$ *represents an indictor to show if both role and resource type in group A have been executed, i.e., if the enabled constraint can be released.*
- $F^c = (P^c \times T^c) \cup (T^c \times P^c) \cup (T^{ret} \times P^c) \cup (P^c \times T^{ret}) \cup (T^r \times P^c) \cup (P^c \times T^r)$*, where,* $T_x\bullet \times T^r$ *and* $T_x\bullet \times T^{ret}$ $(T_x\bullet = \{p \in P^c \mid (T_x \times p) \in F^c\})$ *can be* ***weak relation*** *or* ***normal relation*** *depending on the amount of tokens in* $T_x\bullet$*. (1) If no token is deposited in* $T_x\bullet$*, then the relation between* $T_x\bullet$ *and the transition in role-net or resource type-net becomes a weak relation that will not affect the execution of the transition, i.e.* $T_x\bullet$ *is not counted as one of pre-places of the transition. (Note, the transition in role-net and resource type-net can be fired if and only if all the pre-places of the transition have accumulated enough tokens.) In this case, the transition in*

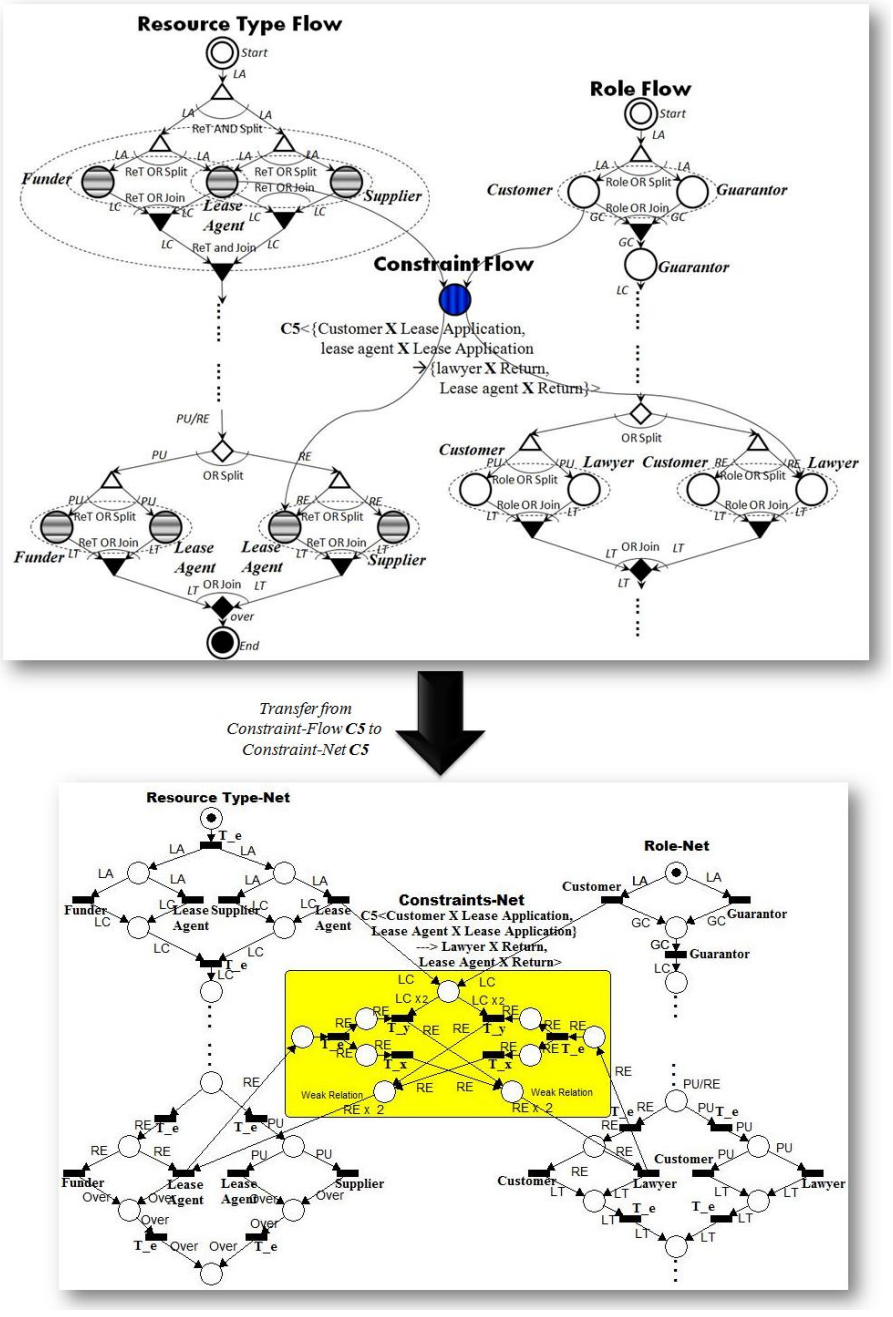

FIGURE 5.16: From Constraint-Flow C5 to Constraint-Net C5

role-net or resource type-net can be fired freely without considering if the token has been deposited in the pre-place which links with the transition by weak relation. (2) When at least one token is deposited in $T_x\bullet$, the relation between $T_x\bullet$ and the transition in role-net or resource type-net becomes normal relation. If a pre-place of a transition is link to the transition by a normal transition, then the transition can be fired under the condition that all pre-places with normal relation must have accumulated enough tokens.

- *$W{:}F^c \rightarrow IN$ is a weight function on the relation between place and transition. It reflects how many tokens are needed to pass the relation (IN represents Integer). For example, $W{:}(\bullet T_y, T_y) \rightarrow 2$ ($\bullet T_y = \{p \in P^c \mid (p, T_y) \in F^c\}$), $W{:}(T_x\bullet, T_r) \rightarrow 2$ and $W{:}(T_x\bullet, T_{ret}) \rightarrow 2$. All other weight of relation are equal to 1.*

- *Count:$P^c \rightarrow$ IN is a function to calculate the amount of tokens at each place. The result of this function on a specific place can be used to decide if the relation between the transition and this place is a weak relation or a normal relation.*

In Fig. 5.17, a complete view of SOAC-Net based on the motivating example is presented, where role-net, resource type-net, and each category of constraint-net are illustrated.

5.3.2 Execution of SOAC-Net

The SOAC-Net execution complies with the general execution policy of Petri-Net, where the transition is enabled when all pre-places are filled by enough tokens, consume all tokens in pre-places and generate new tokens for all post-places. A weight function $\texttt{W}{:}F^c \rightarrow \texttt{IN}$ is used in constraints-net to evaluate how many tokens accumulated in pre-place is enough to enable the relevant transition. At beginning, two tokens are deposited at each initial place of role-net and resource type-net. Once the tokens reach the final place of each net, we assume that the authorization flows are successfully executed. However, the execution mechanisms designed for **Group of Role and**

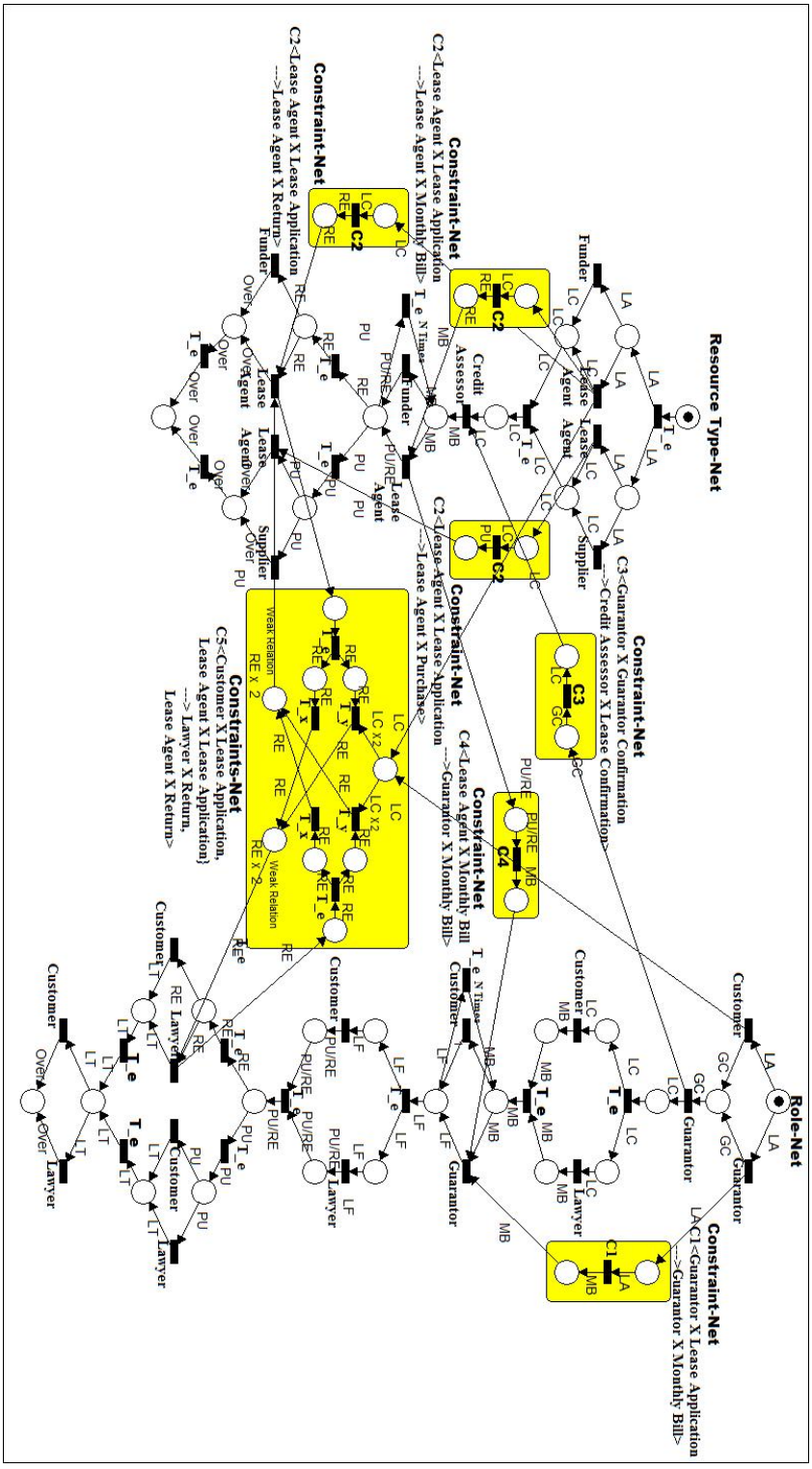

FIGURE 5.17: SOAC-Net

Resource Type Dependency Policy *($\mathcal{C}5_{r\times ret\to r\times ret}$)* is complicated, which will be introduced in details in this section.

For example, in Fig. 5.18, the group of `Lease Agent` support and `Lawyer` access on the operation of *Return* depends on the group of `Lease Agent` support and `Customer` access on the operation of *Lease Application*. A constraint-net C5 is used to realize the dependency policy. In Fig. 5.18, initially, only one token arrives at the pre-place of the transition (`Lease Agent` support on *Return*) in resource type-net. All the other accesses and supports related with that dependency policy are still not executed. Hence, there is no token deposited in $T_x\bullet$ in constraint-net C5. The relation between $T_x\bullet$ and the transition (`Lease Agent` support on *Return*) is a weak relation, that will not affect the fire of this transition (**See Step 1 in Fig. 5.18**). After its firing, one token will be passed to T_e in constraint-net C5 and the other one will be moved into the post-place of this transition in resource type-net (**See Step 2 in Fig. 5.18**). The token in the pre-place of T_e in constraint-net C5 will be split into $\bullet T_y$ and $\bullet T_x$ after T_e firing (**See Step 3 in Fig. 5.18**). At this time, T_x can be enabled and fired to deposit one token from $\bullet T_x$ to $T_x\bullet$. When `Count(`$T_x\bullet$`)=1`, the relation between $T_x\bullet$ and the transition (`Lawyer` access on *Return*) in role-net becomes a normal relation, i.e., $T_x\bullet$ becomes one of the pre-places of the transition to affect its firing. Since `W:`$T_x\bullet\times T_r$`=2` and `W:`$T_x\bullet\times T_{ret}$`=2`, one token in $T_x\bullet$ now is not enough to enable the normal relation with weight 2. Therefore, even if the pre-place of the transition (`Lawyer` access on *Return*) in role-net has accumulated enough tokens, the transition still can not be fired, since its pre-place ($T_x\bullet$) in constraints-net C5 has not accumulated enough tokens to pass the relation with weight 2. The **Group of Role and Resource Type Dependency Policy** *($\mathcal{C}5_{r\times ret\to r\times ret}$)* is enforced on the `Lawyer` access on *Return* (**See Step 4 in Fig. 5.18**).

From the above example, we can observe, that without the execution of depended role access and resource type support, one of the depending role access and resource type support can not be performed if the other one has already been executed. That is the function of the constraint-net C5. In this example, we can further observe that,

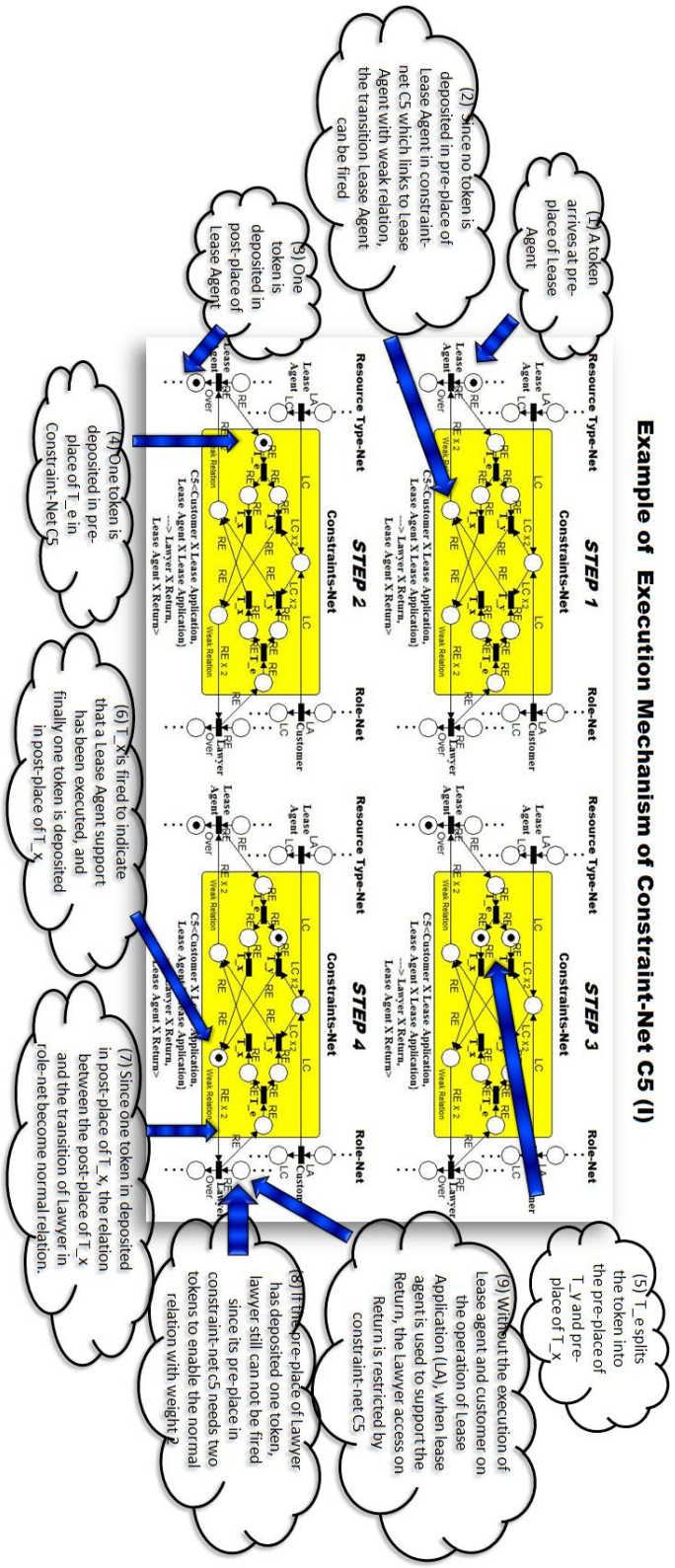

FIGURE 5.18: Example of Execution Mechanism of Constraint-Net C5 (I)

it does not matter which element in depending group to fire firstly. The execution mechanism of constraint-net C5 can automatically restrict the execution of the other element in the depending group, when one element in the group has been fired. Although several steps are needed in constraint-net C5, we assume that the time interval between the fire of one element and the restriction on the execution of the other element is so small that it can be ignored. So we can ignore the situation where two elements in depending group are required to be executed at the same time.

The restriction of Constraint-Net C5 can be released if both elements in a depended group have finished execution. For example, in Fig. 5.19, two tokens arrive at the pre-places of the transition (`Customer` access on *Lease Application*) in role-net and the transition (`Lease Agent` support on *Lease Application*) in resource type-net, respectively (**See Step 5 in Fig. 5.19**). After their firing, two tokens are deposited in $\bullet T_y$ in Constraint-Net C5 and the other tokens are moved in post-places of these transitions in role-net and resource type-net (**See Step 6 in Fig. 5.19**). Hence T_y can be fired as both pre-places of T_y have accumulated enough tokens. Note, the execution of one of the elements in a depended group is not enough to enable T_y since it can only deposit one token in $\bullet T_y$, and the weight of the relation between T_y and this $\bullet T_y$ is 2 `W:`$\bullet T_y \times T_y$`=2`. Now, both pre-places of T_y have enough tokens, one with 2 tokens from the execution of depended elements and the other with 1 token from T_e. T_y then can be fired to consume all tokens and one token can be deposited in $T_y\bullet$, also known as $T_x\bullet$ (**See Step 7 in Fig. 5.19**). Now, `Count(`$T_x\bullet$`)=2`, means that enough tokens have been accumulated to pass the relation between $T_x\bullet$ and the transition (`Lawyer` access on *Return*). If the pre-place of this transition in role-net has already accumulated enough tokens, then it can fire now, i.e. the restriction of Constraint-Net C5 has been released (**See Step 8 in Fig. 5.19**).

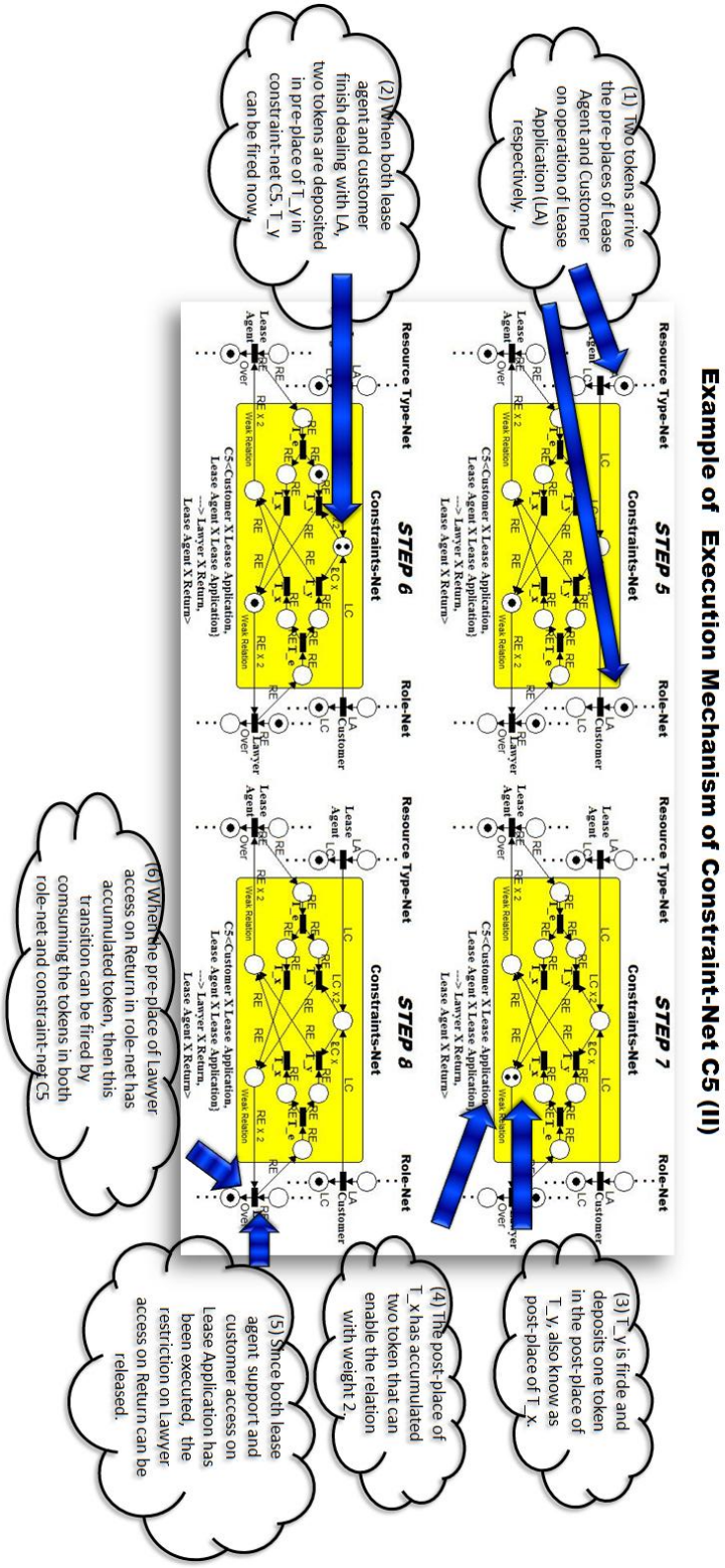

FIGURE 5.19: Example of Execution Mechanism of Constraint-Net C5 (II)

5.4 Conclusion

In this chapter, we propose a Petri-Net based process model, SOAC-Net, based on our conceptual model of service oriented authorization control. Currently, SOAC-Net is the only process model that takes the component services (resources) into account in managing composite web service authorization in process environment. In SOAC-Net, two types of authorization synchronization policies and five types of authorization dependency policies are enforced.

However, with more and more authorization policies are defined based on SOAC-Net, how to determine whether they are defined properly becomes a challenge. The improper authorization policy definition can cause the suspension of the execution of a composite web service. Hence, in the next chapter, a mechanism will be introduced based on SOAC-Net to verify the reliability of SOAC-Net in terms of authorization policy definition.

Algorithm 1 Algorithm for Constructing Role-Flow

Input: $\mathbb{R}$, $\mathbb{OP}$, OPA$\subseteq\mathbb{OP}\times\mathbb{R}$, $\mathbb{F}\subseteq\mathbb{OP}\rightarrow\mathbb{OP}$

Output: $\mathbb{E}\subseteq$OPA$\rightarrow$OPA

{Based on the operation's execution sequence $\mathbb{F}$, and the mapping OPA between Role $\mathbb{R}$ and Operation $\mathbb{OP}$, a sequence of role accesses $\mathbb{E}$ is constructed}

Steps:

1: **while NOT the end of $\mathbb{F}$ do**

2: /*Check if the operation op_i is mapped with roles by using the inverse function of assigned_op(r$\in\mathbb{R}$)*/

3: **if assigned_op^{-1}($op_i \in\mathbb{OP}$)$\neq\emptyset$ then**

4: /*Select all roles that are mapped with op_i into R$'$, a subset of Role $\mathbb{R}$. */

5: $\exists$ **R$'\subseteq\mathbb{R}$,** $\forall$ $r_i\in$**R$'$,** $r_i \times op_i$**=OPA$'\subseteq$OPA,** $\forall$ $r_j\in\mathbb{R}$**-R$'$,** $r_j \times op_i \not\subseteq$**OPA**;

6: /*Relationship() is a recursive function on finding all relationships of Exclusive-Choice or Concurrency between different OPAs or none for one OPA within the set of OPA$'$.*/

7: **W=Relationship(OPA$'$)**;

8: $\mathbb{E}$**=Sequence($\mathbb{E}$,R$'$,W)**; /* Based on the set of role R$'$ and the set of relationships W, the roles mapped with op_i are added in the role-flow $\mathbb{E}$ with the relationships within OPA$'$. */

9: **end if**

10: **Next(**$op_i\in\mathbb{OP}$**)**; /*Select the next operation $\mathbb{F}$*/

11: **end while**

Algorithm 2 Algorithm for Constructing Resource Type-Flow

Input: $\mathbb{ReT}$, $\mathbb{OP}$, SPA$\subseteq\mathbb{OP}\times\mathbb{ReT}$, $\mathbb{F}\subseteq\mathbb{OP}\rightarrow\mathbb{OP}$

Output: $\mathbb{D}\subseteq$SPA$\rightarrow$SPA

{Based on the operation's execution sequence $\mathbb{F}$, and the mapping SPA between Resource Type $\mathbb{ReT}$ and Operation $\mathbb{OP}$, a sequence of resource type supports $\mathbb{F}$ is constructed}

Steps:

1: **while NOT the end of $\mathbb{F}$ do**

2: /*Check if the operation op_i is mapped with resource types by using the inverse function of assigned_ret(ret$\in\mathbb{ReT}$)*/

3: **if assigned_ret^{-1}($op_i \in\mathbb{OP}$)$\neq\emptyset$ then**

4: /*Select all resource types that are mapped with op_i into ReT$'$, a subset of resource type $\mathbb{ReT}$. */

5: $\exists$ **ReT$'\subseteq\mathbb{ReT}$,** $\forall$ $ret_i\in$**ReT$'$,** $ret_i \times op_i$**=SPA$'\subseteq$SPA,** $\forall$ $ret_j\in\mathbb{ReT}$**-ReT$'$,** $ret_j \times op_i \not\subseteq$**SPA**;

6: /*Relationship() is a recursive function on finding all relationships of Exclusive-Choice or Concurrency between different SPAs or none for one SPA within the set of SPA$'$.*/

7: **W=Relationship(SPA$'$)**;

8: $\mathbb{D}$**=Sequence($\mathbb{D}$,ReT$'$,W)**; /* Based on the set of resource type ReT$'$ and the set of relationships W, the resource types mapped with op_i are added in the resource type-flow $\mathbb{D}$ with the relationships within SPA$'$. */

9: **end if**

10: **Next(**$op_i\in\mathbb{OP}$**)**; /*Select the next operation $\mathbb{F}$*/

11: **end while**

6

Authorization Policy Verification

6.1 Introduction

Authorization policies can be enforced based on the process model SOAC-Net to manage the composite web service authorization. However, how to decide whether all policies defined in SOAC-Net are correct becomes an issue that is not tackled. Therefore, we need to provide a verification mechanism based on SOAC-Net to ensure the defined authorization policies can consistently work with the execution of the composite web service. In a word, there at least exists one execution path in SOAC-Net to comply with the business logic of the composite web service execution. The existence of the path in SOAC-Net means that there exist proper service consumers and resources as specific roles and resource types that can access and support the operations of the composite web service, when they are needed according to the business logic of the

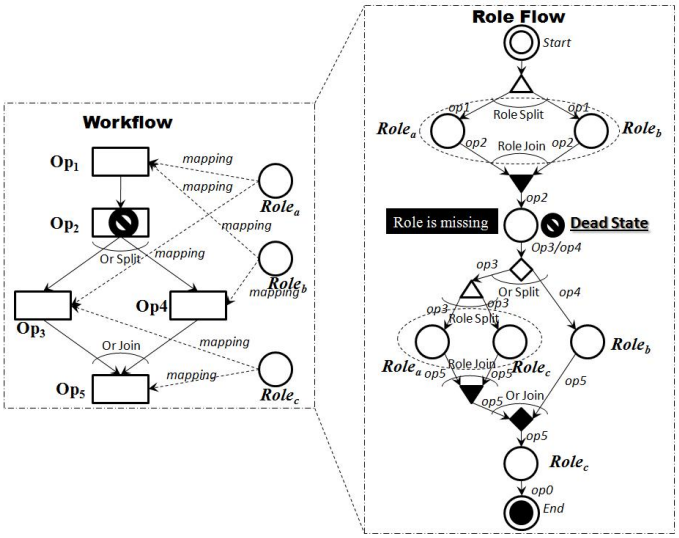

FIGURE 6.1: Improper Authorization Policy Definition I

composite web service. The whole business process of the composite web service will be suspended or even terminated due to the unavailable support or access. We call such authorization flow that can consistently work with the workflow of the composite web service as reliable authorization flow.

However, with more and more authorization policies being defined based on the process model SOAC-Net, the possibility of improper authorization policy definition becomes higher. When authorization policy is not defined properly, the necessary role or resource type may be missing, which makes service consumer or resource not be able to access and support the operations. Such missing can eventually suspend the whole business process, and cause the process terminated. When we use authorization synchronization policies to construct the role-flow and resource type-flow based on the business logic of a composite web service, we can find that, if the role-flow or resource type-flow are affected by the improper authorization policies, they will step into a dead state, i.e., a state in which the process is suspended. For example, in Fig. 6.1, operations 1 to 5 are used to construct a normal workflow of the operations of the composite web service. $Role_a$ to $Role_c$ are mapped to the operations. However, there is no role mapped to operation 2 that is key operation in the execution path of

the common workflow. When we develop a role-flow based on the business logic of a composite web service, we can find that the role-flow is put into a dead state where no service consumer can access operation 2 as a specific role during the business process.

Petri Net [44] provides a set of verification mechanisms[46], and its graphically and mathematically founded modeling formalism with various algorithms for design and analysis[45] makes it a good candidate for modeling authorization flow (Therefore, a Petri-Net based process model has been developed in last chapter). Various properties of Petri Net, including *reachability*, *dead marking*, *Soundness*, etc., are well presented in the literature. These properties have already been used for verifying the reliability of service composition and workflow[83–86, 88]. In this chapter, we propose a reliability property based on SOAC-Net to cater for the novel semantics of authorization flow, named *authorization-embedded dead marking freeness*. This new property based on SOAC-Net can be used to verify the improper authorization policy definition.

The rest of this chapter is organized as follows. In section 2, the verification mechanism is presented in details including formal method and example. Conclusion is presented in the last section.

6.2 Verification Mechanism

In this section, we will introduce a verification mechanism to verify the *Improper Authorization Policy Definition* based on SOAC-Net, i.e., to confirm in which step of SOAC-Net the authorization policies is defined improperly.

Currently, the mechanism that can be used to verify the improper authorization policy in SOAC-Net is only acyclic. In future, a more general mechanism should be developed to work on SOAC-Net which consists of different types of workflow patterns, including looping.

The *transitive matrix* L^*_{BP} [50] is an m×m matrix, where m denotes the amount of the place in a specific Petri-Net based process model. A transition t will be the element of the *transitive matrix* L^*_{BP} at row i and column j, if place i is the pre-place of the transition and place j is the post-place of the transition. Through the

transitive matrix L^*_{BP}, the structure of a specific Petri-Net based process model is transferred into a form of matrix. We extend the transitive matrix by associating it with the authorization information, called *authorization-embedded transitive matrix*, and use it to detect the *Improper Authorization Policy Definition*. The authorization information in the matrix can be the name of role , resource type, constraint name in related transition, or even T_e. For example, in role-flow, the transition R_2 is between the pre-place i and post-place j. Then R_2 as the authorization information will be put in the i^{th} row and j^{th} column of transitive matrix. If the authorization information is missing due to improper authorization policies definition, then an empty set symbol will be put in a relevant position in the transitive matrix.

Before presenting the verification method, we firstly describe the syntax of two types of matrix algebraic operator $\diamond$ and $\triangle$ which are used by the verification method on the property of *authorization dead marking freeness*.

Please note, the definition is regarding to how to build an authorization embedded transitive matrix. The elements in the transitive matrix are sets including the authorization information, which is quite different with traditional transitive matrix. The grammar of definition follows BNF-like notation:

$$M ::= M_1 \diamond M_2 | M_1 \triangle M_2$$

Where:

- $M_1 \diamond M_2$: Given an n×m matrix M_1, an m×n matrix M_2, and an n×n matrix M_3 where M_1=$[c_{ij}]$, M_2=$[d_{ji}]$, and M_3=$[e_{ii}]$ (i=1..n, j=1..m). Then

$$M_3 = M_1 \diamond M_2 \Rightarrow [e_{ii}] = \bigcup_{j=1}^{m} ([c_{ij}] \cap [d_{ji}])$$

- $M_1 \triangle M_2$: Given an n×m matrix M_1, an m×n matrix M_2, and an n×n matrix M_3 where M_1=$[c_{ij}]$, M_2=$[d_{ji}]$, and M_3=$[e_{ii}]$ (i=1..n, j=1..m). Then

$$M_3 = M_1 \triangle M_2 \Rightarrow [e_{ii}] = \biguplus_{j=1}^{m} ([c_{ij}] \times [d_{ji}])$$

Where $[c_{ij}] \times [d_{ji}]$ is a relation, i.e, the element in $[c_{ij}]$ is mapped to the element in $[d_{ji}]$. $\biguplus$ is an operator where it puts a sequence of available relations into a set. The available relation should not include any empty set operator . Since $[c_{ij}]$ and $[d_{ji}]$ can be a set to include multiple elements, $\biguplus_{j=1}^{m}[c_{ij}] \times [d_{ji}]$ will generate a set of relations where each element in $[c_{ij}]$ needs to be mapped to each element in $[d_{ji}]$

Let us see two examples to describe the two proposed algebraic operators.

$$M_1 = \begin{bmatrix} \emptyset & \{a,b\} \\ \{c,e\} & \emptyset \\ \{e\} & \{c\} \end{bmatrix}$$

$$M_2 = \begin{bmatrix} \{e\} & \emptyset & \{c\} \\ \{c,e\} & \{b\} & \{a\} \end{bmatrix}$$

$$M_x = M_1 \diamond M_2 = \begin{bmatrix} \emptyset & \{b\} & \{a\} \\ \{e\} & \emptyset & \{c\} \\ \{c,e\} & \emptyset & \emptyset \end{bmatrix}$$

$$M_y = M_1 \triangle M_2 = \begin{bmatrix} \{a\times c, a\times e, b\times c, b\times e\} & \{a\times b, b\times b\} & \{a\times a, b\times a\} \\ \{c\times e, e\times e\} & \emptyset & \{c\times c, e\times c\} \\ \{e\times e, c\times e, c\times c\} & \{c\times b\} & \{e\times c, c\times a\} \end{bmatrix}$$

The proposed algebraic operators guarantee that each result of matrix computing is still a matrix to which we can again apply algebraic operators. Below we formally define how these two operators can be used.

Definition 15 *An authorization-embedded transitive matrix:*

$$L_{BP}^{au} = (A^-)^{TP} \diamond Diag(t_1, ..., t_n) \diamond A^+$$

where A^-=$[a_{ij}^-]$ and A^+=$[a_{ij}^+]$ are $n\times m$ matrices (n transitions, m places). TP means transpose matrix, and Diag() represents diagonal matrix.

$$a_{ij}^{-} = \begin{cases} U \text{ (full set)} & (p_j, t_i) \in F^r \cap F^{ret} \cap F^c \\ \emptyset \text{ (empty set)} & (p_j, t_i) \notin F^r \cap F^{ret} \cap F^c \end{cases}$$

$$a_{ij}^{+} = \begin{cases} U \text{ (full set)} & (t_i, p_j) \in F^r \cap F^{ret} \cap F^c \\ \emptyset \text{ (empty set)} & (t_i, p_j) \notin F^r \cap F^{ret} \cap F^c \end{cases}$$

t_h(h=1,2,...,n) is the transition in a specific position to represent the authorization information, e.g. the name of role in role flow, resource type in resource type flow or the constraint in constraint-net. the authorization information can be formally described as:

$$t_h = \begin{cases} r & r \text{ represents a role in Role-Net} \\ ret & ret \text{ represents a resource type in Resource Type-Net} \\ c & c \text{ represents a constraint in constraints 3 to 6} \\ T_x \text{ and } T_y & T_x \text{ and } T_y \text{ represent the transition in constraint 7} \\ T_e & T_e \text{ represents an empty transition where it pass, split, or collect the tokens only.} \end{cases}$$

L_{BP}^{au} is used to represent the structure of the Petri-Net based process model from an authorization aspect. Marking M is a 1×m matrix to represent each step of the movement of the token in the process model. m is the amount of place in SOAC-Net. If there exists a token at the place j in a specific step of the token movement, the j^{th} element in the marking (at row 1 column j) is equal to the value of the token. The value of the token is a series of operations that need to be accessed and supported during the execution of a composite web service. All other elements in the marking is $\emptyset$. For example, marking M_1=[op_1, $\emptyset$, $\emptyset$] represents that a token op_1 is in place 1; while marking M_2=[$\emptyset$, op_2, $\emptyset$] means that a token is moved into place 2 from place 1 after one step token movement ($M_1 \rightarrow M_2$). The token movement from place 1 to place 2 represents a transition between these two places is fired. The value of token is changed during the token movement.

The relation between the transition and its pre-places is restricted by the weight function. Only enough tokens, i.e the amount of token is equal to or more than the weight of the relation, are accumulated, the transition then can be fired, and the firing will cause the token to move from the pre-place of the transition to the post-place of

the transition, i.e. from marking M_{K-1} to marking M_K. However, if the transition can not be enabled and fired due to the missing of the authorization information, the token movement will be ceased in this transition in SOAC-Net and is called *authorization dead marking*, a kind of dead marking caused by the authorization error. Here we introduce a verification method to see if a SOAC-Net is *authorization-embedded dead marking free* at the k^{th} step of token movement (k=1 to v, where v represents how many steps are needed for the token to move from the initial place to the final place in the process model). Before a token moves from Marking M_{K-1} to M_K, we use M_{k-1} to compute with L^{au}_{BP} to get a temporary 1×m matrix M^{*au}_k. Hence, we call the matrix M^{*au}_k as temporary marking. In M^{*au}_k, we need to check if M_{K-1} is an authorization dead marking. If yes, then M_K can not be realized from M_{K-1} and a dead marking is detected.

Formally,

$$M^{*au}_k = M_{K-1} \triangle L^{au}_{BP} \Rightarrow s[1j] = \biguplus_{i=1}^{m}(c[1i] \times a[ij])$$

where j=1..m, M^{*au}_k=s[1j], M_{K-1}=c[1i], and L^{au}_{BP}=a[ij].

Definition 16 *A SOAC-Net is* authorization dead marking free *if*

$$\frac{CountSet(s[1j])}{CountSet(a[ij])} \geq \min(Weight(p_i \times \{\bullet p_j\}))$$

Where CountSet()→ $\mathbb{N}$ is a function to count how many elements exist in a set. CountSet(a[ij])<>0 and s[1j] and a[ij] are defined in M^{*au}_k.

Since M^{*au}_k=s[1j] where j represents the j^{th} place (p_j) in M^{*au}_k, s[1j]=$\emptyset$ means that there is no way that a token can be deposited at the j^{th} place (p_j) in M_k after a specific transition is fired to make the marking from M_{k-1} to M_K. Therefore, if all elements in M^{*au}_k are empty, i.e. s[1j]=$\emptyset$, j=1..m, then the dead marking occurs as there is no place can be deposited a token after a transition firing, i.e., the marking is dead at M_{k-1} and M_k will never be fired. On the other hand, if s[1j] is a set which includes the relation of role×operation, resource type× operation, or any other relations, then it is possible that a token can be deposited at the j^{th} place (p_j) in Marking M_K from M_{k-1},

if enough tokens have been accumulated to enable the transition to fire. Hence, we will examine if enough tokes have been accumulated to cause the deposition of the tokes at the j^{th} place in marking M_K. The examine will be carried out based on the amount of relations in s[1j]. CountSet(s[1j]) is used to calculate how many relations exist in all non-empty places (p_j j=1..m). a[ij] is a set in L^{au}_{BP} that includes all transitions after the place p_i, before place p_j. CountSet(a[ij]) is used to calculate the amount of the transitions in a[ij]. Weight($p_i \times \{\bullet p_j\}$) is used to calculate the weight of the relations of the place p_i and the transitions between p_i and p_j (written as $\bullet p_j$). There can be multiple transitions between p_i and p_j. min() is a function to select the minimum weight on the relations between place p_i and the pre-transitions of p_j.

The transition can fire only if the amount of tokens deposited in p_i is equal to or more than the weight of the relation. We use CountSet(s[1j])/CountSet(a[ij]) to compare the minimum weight. Since a[ij] can include multiple transitions between the same pre-place and the same post-place, in L^{au}_{BP} an element at the i^{th} row and the j^{th} column becomes a set to include these transitions. When $M_{K-1} \triangle L^{au}_{BP}$, one token, also known as one operation in the i^{th} place at M_{K-1}, will construct a relation with each transition in the set a[ij]. In s[1j], at the j^{th} position, we can have the amount of relations as the amount of tokens at the i^{th} place (p_i) in M_{K-1} multiply the amount of transitions at the i^{th} row and the j^{th} column in a[ij] at L^{au}_{BP}. Obviously, we can not use this amount of the relations to compare with the weight of the relations between p_i and all transitions in $\{\bullet p_j\}$, since the amount of the relations in s[1j] can be increased with the amount of elements in a[ij]. For example, if a[ij] includes three elements representing three transitions between place p_i and place p_j, then one token at c[1i] (M_{k-1}) can cause three relations in s[1j]. We can not use the three relations to compare the weights between place p_i and each transition. Hence, we use CountSet(s[1j])/CountSet(a[ij]) to erase the influence caused by the amount of transitions in a[ij]. The result will precisely calculate how many tokens have been accumulated in p_i.

Since there can be multiple transitions between place p_i and place p_j, the relations between place p_i and these transitions can also be multiple. The amount of tokens in p_i only needs to be greater than the minimum weight between place p_i and these

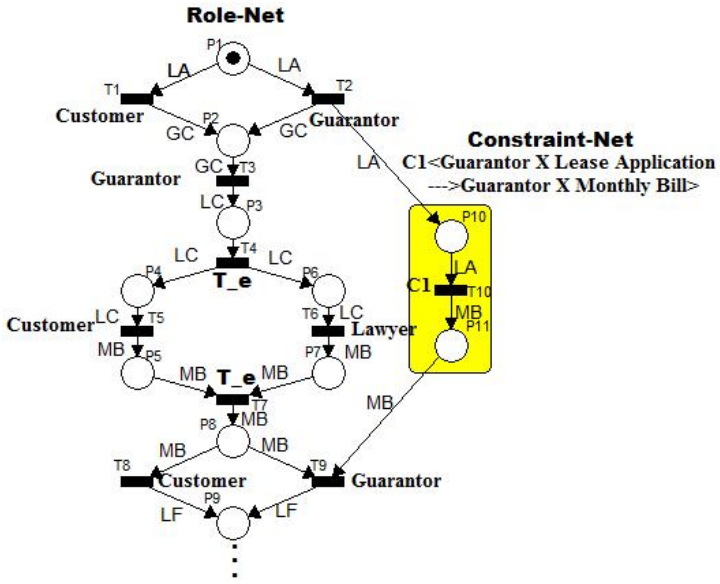

Figure 6.2: Example for Verification Mechanism

transitions, i.e. at least the transition with the minimum weight can be fired. Hence, we use min(Weight($p_i \times \{\bullet p_j\}$)) to compare with the amount of $\frac{CountSet(s[1j])}{CountSet(a[ij])}$.

In summary, if the amount of the relations at the j^{th} position in s[1j] divided by the amount of the transitions between place p_i and p_j is still greater than the amount of the minimum weight of the relations between place p_i and all transitions in $\{\bullet p_j\}$, then we can say that tokens can be moved from p_i to p_j so that marking M_{K-1} can step forward to M_K.

In Fig. 6.2, we take part of SOAC-Net as an example, where part of role-net and constraint-net C1 are included. Firstly, we need construct an *authorization-embedded transitive matrix* L^{au}_{BP} of this part of SOAC-Net. Since $L^{au}_{BP} = (A^-)^{TP} \diamond Diag(t_1, ..., t_n) \diamond A^+$, the $(A^-)^{TP}$, $Diag(t_1, ..., t_n)$, and A^+ should be built up based on the structure of our example diagram in Fig. 6.2. $(A^-)^{TP}$ is an m×n matrix where m is equal to the amount of places and n is equal to the amount of transitions. In Fig. 6.2, $(A^-)^{TP}$ is a 11×10 matrix where m=11 and n=10, i.e we have 11 places and 10 transitions in Fig. 6.2. U in $(A^-)^{TP}$ represents Full Set. In below $(A^-)^{TP}$, there is a full set U at the 3^{rd} row and 4^{th} column. It means there is a relation as $P_3 \times T_4$ in Fig. 6.2. $\emptyset$ at the i^{th}

row and j^{th} column in $(A^-)^{TP}$ means that there is no relation of $P_i \times T_j$ (See Fig. 6.2).

$$(A^-)^{TP} = \begin{bmatrix} U & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & U & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset \end{bmatrix}$$

$Diag(t_1, ..., t_n)$ is an $n \times n$ diagnose matrix where n is the amount of transitions. t_h (h=1..n) is the value of transition, e.g., the name of role, empty name (for T_e), or the name of the constraint that the transition represents. Hence, the $Diag(t_1, ..., t_n)$ below is a 10×10 matrix, where Cu represents `Customer`, Gu represents `Guarantor`, and La represents `Lawyer`.

Diag(t_1,...,t_n)

$$= \begin{bmatrix} Cu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & Gu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & Gu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & Cu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & La & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Cu & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Gu & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & C_1 \end{bmatrix}$$

A^+ is an n×m matrix where a Full Set U at i^{th} row and j^{th} column in A^+ represents

the existence of relation of $T_i \times P_j$. For example, at below A^+, there is a U full set at the 10^{th} Row and the 11^{th} Column. It means that in Fig. 6.2, there is a relation as $T_{10} \times P_{11}$.

$$A^+ = \begin{bmatrix} \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset \\ \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & U & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & U \end{bmatrix}$$

Hence, the *authorization-embedded transitive matrix* L_{BP}^{au} of this part of SOAC-Net is $L_{BP}^{au} = (A^-)^{TP} \diamond Diag(t_1, ..., t_n) \diamond A^+$, as below,

$$L_{BP}^{au} = \begin{bmatrix} \emptyset & \{Cu, Gu\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Gu & \emptyset \\ \emptyset & \emptyset & Gu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & T_e & \emptyset & T_e & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & Cu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & La & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{Cu, Gu\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & C_1 \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Gu & \emptyset & \emptyset \end{bmatrix}$$

Marking M_0 as the initial marking is [LA, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$] (See Fig. 6.2), where only token LA is deposited as Place P_1. Hence, we can calculate temporary

matrix M_1^{*au}=$M_0 \triangle L_{BP}^{au}$.

$$M_1^{*au} = \left[\begin{array}{cccccccccc} \emptyset & \{LA \times Cu, LA \times Gu\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{LA \times Gu\} \quad \emptyset \end{array} \right]$$

where there are two sets {LA×Cu, LA×Gu} and {LA×Gu} that are non-empty at the 2^{nd} and 10^{th} positions in M_1^{*au} respectively, i.e., M_1^{*au}=s[1j] j=1..11, s[1,2]={LA×Cu, LA×Gu}, s[1,10]={LA×Gu}. Hence, according to Definition 16, CounetSet(s[1,2])=2, CountSet(s[1,10])=1.

Let us examine **CountSet(s[1,2])=2** firstly. In L_{BP}^{au} (a[ij], i=1..m, j=1..m), when j=2, only a[1,2]={Gu,Cu}, and all other elements in the 2^{nd} column are ∅, e.g, a[2,2] to a[11,2] (CountSet(∅)=0), that can not be used in Definition 16. Hence, we only need check **CountSet(a[1,2])=2**. Now, we can conclude that **CountSet(s[1,2])/CountSet(a[1,2])=1** where i=1 and j=2. In Fig. 6.2, there are two transitions between P_i (i=1) and P_j(j=2), i.e. T_1 and T_2. Hence, **min(weight($P_1 \times \bullet P_2$))=1** since weight(P_1, T_1)=1 and Weight(P_1, T_2)=1. Now, in the temporary matrix M_1^{*au} in Fig. 6.2, for the 2^{nd} position, i.e. j=2,

$$CountSet(s[1,2])/CountSet(a[1,2]) = min(weight(P_1 \times \bullet P_2)) = 1$$

where i=1 and j=2.

According to Definition 16, the part of SOAC-Net is authorization dead marking free in M_0, i.e. M_0 can be transferred into M_1 based on the fire of a specific transition. In this example, the place P_2 will be deposited a token in M_1 after the firing of a specific transition, since the corresponding position in M_1^{*au} that satisfies the Definition 16 is the 2^{nd} position.

Now let us examine **CountSet(s[1,10])=1** where j=10. For all elements in a[i,10], i=1..11, (in L_{BP}^{au}), only a[1,10]={Gu} and all other elements in the 10^{th} column of L_{BP}^{au} are ∅ that can not be used in Definition 16 because of **CountSet(∅)=0**. Hence, we can only take CountSet(a[1,10])=1 into account. **CountSet(s[1,10])/ CountSet(a[1,10])=1** where i=1, j=10. min(weight($p_1 \times \bullet p_{10}$))=1 since there is only one transition between p_1 and p_{10}, i.e., T_2, and $p_1 \times T_2$=1. Now, in the temporary matrix M_1^{*au} in Fig. 6.2, for the 10^{th} position, i.e. j=10,

$$CountSet(s[1,10])/CountSet(a[1,10]) = min(weight(P_1 \times \bullet P_{10})) = 1$$

where j=10 and i=1.

According to definition 16, the part of SOAC-Net is authorization dead marking free in M_0, i.e. M_0 can be transferred into M_1 based on the fire of a specific transition. In this example, the place P_{10} will be deposited a token in M_1 after the firing of a specific transition, since the corresponding position in M_1^{*au} that satisfies the Definition 16 is the 10^{th} position.

In summary, after T_1's firing, M_1=[$\emptyset$, GC, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$], where a token is deposited in P_2 at marking M_1. On the other hand, if T_2 is fired, M_1=[$\emptyset$, GC, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, GC, $\emptyset$], where tokens can be deposited in both P_2 and P_{10} at marking M_1. We can conclude that in marking M_0, the part of SOAC-Net in Fig. 6.2 is *authorization dead marking free*.

However, in Fig. 6.2, if the role information is missing for accessing the operation of *Guarantor Confirmation*, then the part of SOAC-Net will become *authorization dead marking*, that can be verified by our proposed mechanism. In this case, the original L_{BP}^{au} will be changed as $L_{BP}^{au^*}$ as below,

$$L_{BP}^{au^*} = \begin{bmatrix}
\emptyset & \{Cu, Gu\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Gu & \emptyset \\
\emptyset & \emptyset & \emptyset^* & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & T_e & \emptyset & T_e & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & Cu & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & La & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & T_e & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{Cu, Gu\} & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & C_1 \\
\emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & Gu & \emptyset & \emptyset
\end{bmatrix}$$

We can observe that in above $L_{BP}^{au^*}$, the original role information {`Gu`} that represents `Guarantor` is missing, and changed as $\emptyset^*$ at the 2^{nd} row and the 3^{rd} column in $L_{BP}^{au^*}$. If we assume that marking is changed from M_0 to M_1 by the firing of transition T_1, then M_1= [$\emptyset$, GC, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$]. We can calculate temporary matrix M_2^{*au} as M_2^{*au}=$M_1 \triangle L_{BP}^{au^*}$.

M_2^{*au}=[$\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$]

Hence, for all elements in M_2^{*au}, CountSet(s[1j])=0 where j=1..11, and CountSet(s[1j])/ CountSet(a[ij])=0 where i=1..m, j=1..m. On the other hand, we know that the minimum weight of relations in SOAC-Net is 1, i.e, min(weight($P_i \times \bullet P_j$))=1. Therefore,

$$CountSet(s[1j])/CountSet(a[ij]) < min(weight(P_i \times \bullet P_j)) = 1$$

where i=1..11, j=1..11.

According to Definition 16, the above example of SOAC-Net becomes authorization dead marking at marking M_1, in case of the missing of role information on the operation of *Guarantor Confirmation*.

Currently, the verification mechanism is only suit for the business process with sequence workflow only, and the complexity of the verification mechanism is N^2.

6.3 Conclusion

With more and more authorization policies defined in authorization system, how to manage the improper authorization policy definition becomes a difficulty that impedes the composite web service authorization management. These improper authorization policy definition can even cause the missing of important authorization information on specific operations and lead to the suspension of the normal execution of a composite web service. In this chapter, we propose a property, named *authorization dead marking freeness* to verify improper authorization policy definition based on SOAC-Net. The associated verification mechanism is also introduced. This verification mechanism can be implemented at both design time and runtime to detect the missing of important authorization information caused by the improper authorization policy definition based

on SOAC-Net.

7

Tool Support

7.1 Introduction

In this chapter, an overview of tool support is presented, named SOAC Engine [91], which is developed based on conceptual model SOAC and process model SOAC-Net proposed in previous chapters. The purpose of this implementation is to demonstrate the functionality of an authorization engine in a service system. It has been developed with the following features:

- SOAC Engine is developed based on our conceptual model, SOAC, which is an extension of role based access control for the authorization of composite web services [92, 93]. In SOAC Engine, the authorization of a composite web service to a service consumer takes the constraints of the component services into consideration. The authorization engine will consider "who can do what under what

kind of support".

- Two concepts, Role and Resource Type, are introduced in SOAC Engine in order to reduce the administrative overhead in the authorization management. Based on the characteristics of component services and service consumers, we classify component services and service consumers and map them into resource types and roles.

- Authorization rules are embedded in SOAC Engine to prevent conflict of interest in terms of composite web service authorization at both design time and runtime.

- SOAC Engine provides an internal embedded Petri-Net process model simulation to model the SOAC-Net based on the authorization relationship of elements in SOAC.

- Authorization policy enforcement and verification on the SOAC-Net are deployed in the Petri-Net Process Model Simulation.

The technical significance of SOAC Engine can be presented as follows: (1) it is a first authorization engine that manages the web service authorization from only the composite service's point of view, rather than global view. (2) It can solve the conflict of interest in composite web service authorization caused by the involvement of the component services. (3) A Petri-Net simulation is embedded within the SOAC Engine to enforce and verify the authorization policies based on SOAC-Net. By itself, the problem of web service authorization from composite service's point of view that we address is challenging, and the method we provide here for avoiding conflict of interest in terms of authorization and managing authorization policy in a business process environment is unique and notably differs from any existing works in the area, which focus on either non-composite service environment [64] or manage composite web service authorization from a global point of view [61].

The rest of the chapter is organized as follows. Section 2 describes system architecture of SOAC-Engine. A demonstration of SOAC-Engine is presented in section 3. We conclude our implementation in Section 4.

7.2 System Architecture

The *SOAC Engine* (See Fig. 7.1) is developed with JAVA and Oracle, and resided in local web server which transfers the access request and authorization notification between Service Consumer and the engine through `Application Interface`. The local web server can also be used to collect the support from resources and manage their information through `Authorization Management Interface` of *SOAC Engine*. This interface is used by administrator and/or system developer as well to (1) set the elements of SOAC and their relationships, (2) identify the conflicted relationships among elements, (3) manage the authorization policies coordination for both composite web service and resource, and (4) define authorization dependency policy according to business security requirements.

The *SOAC Engine* includes four component packages: `Authorization Administrative Management`, `Credential Management`, `Authorization Decision Making`, and `Authorization Enforcement`, as follows,

- `Authorization Administrative Management`, is used to manage the elements of SOAC and their relationships, e.g., adding a role, or querying the assigned roles for specific service consumer. In `Authorization Administrative Management`, the elements of SOAC and their relationships in both service provision part and service realization part are defined and managed by using the 18 administrative operations, the black circles in Fig. 7.1. These components manage service consumers, roles, resources, resource types and the mapping relationships of service consumer/role and resource/resource type. The resource type is a new concept in our proposed authorization framework which has not been used in existing authorization systems. In `Authorization Administrative Management`, the business logic of operations of the composite web service is also stored in database. Static and dynamic conflict free authorization rules in terms of identifying the conflict of interest at both design time and runtime are also recoded in database within `Authorization Administrative Management`. Authorization dependency policy based on business security requirements defined by system administrator is

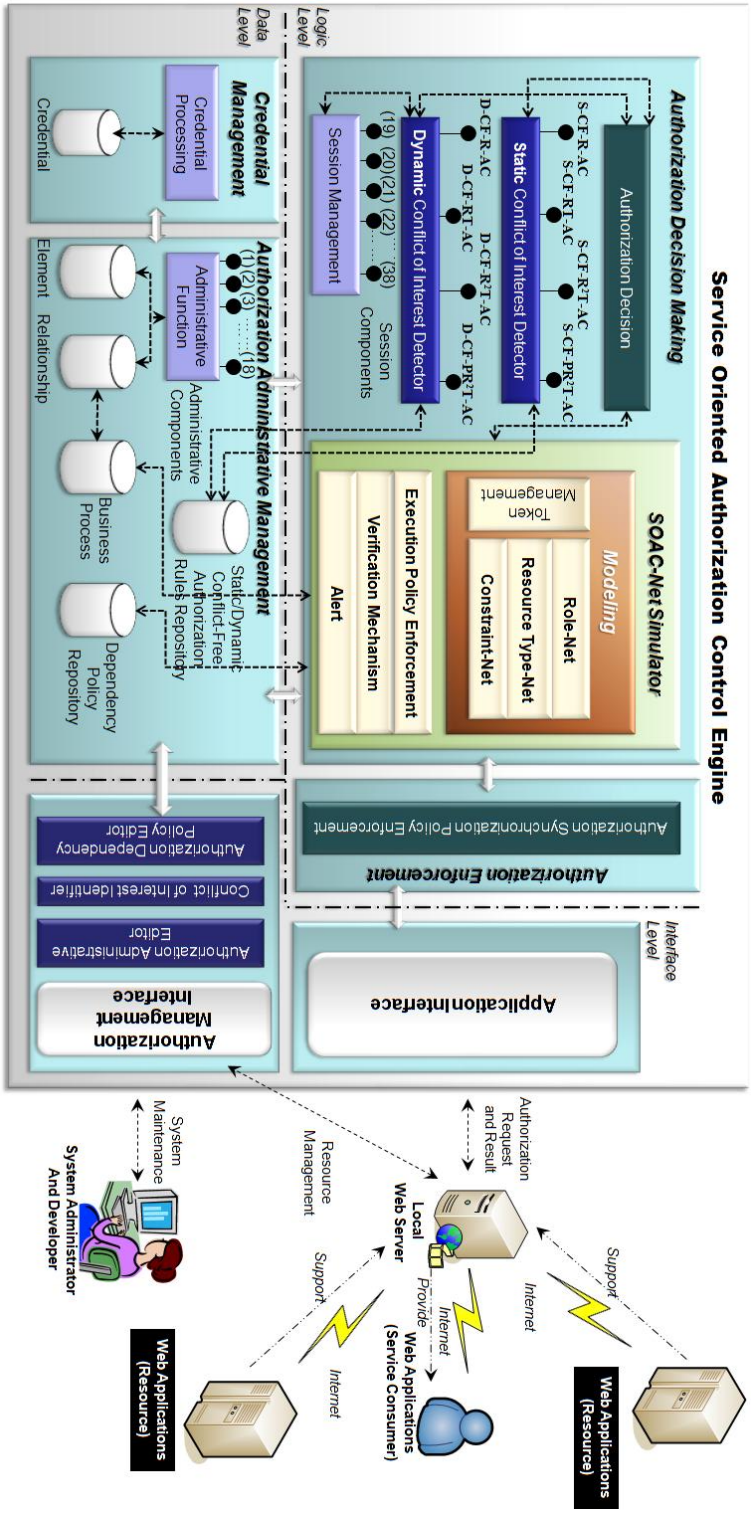

FIGURE 7.1: System Architecture of SOAC Engine

another type of authorization rules recorded in `Authorization Administrative Management`.

- `Credential Management` processes and manages credentials for service consumers and resources. The *SOAC Engine* will not bring new features for this package comparing to existing authorization engines.

- In `Authorization Decision Making`, authorization decision is made based on (1) the checking of Conflict of Interest (CoI) and (2) the verification of SOAC-Net.

  – The conflict of interest may be checked using the static authorization rules or the dynamic authorization rules according to different system contexts or design choices. For detecting Static CoI, four authorization rules, S-CF-R-AC, S-CF-RT-AC, S-CF-R^2T-AC, and S-CF-PR^2T-AC are checked by the *static conflict of interest detector* in the package of `authorization decision making`. For detecting Dynamic CoI, four authorization rules, D-CF-R-AC, D-CF-RT-AC, D-CF-R^2T-AC, and D-CF-PR^2T-AC are checked by the *dynamic conflict of interest detector* in the package of `authorization decision making` (Regarding to each type of conflict of interest, please refer to section 3 in chapter 4.) For checking Dynamic CoI, *Session Management* facilitates the management tasks related with sessions. In Fig. 7.1, black circles on *Session Management* represent the another 20 operations for a series of session management tasks. SOAC engine has the capabilities beyond existing authorization systems to identify and detect the conflict of interest focusing on one resource, a pair of service consumer and resource.

  – The verification of SOAC-Net is another factor that affects the authorization decision. Based on the business process and authorization dependency policy (See Fig. 7.1), a SOAC-Net can be modeled within a simulator, where a Petri-Net based process model can be executed. Role-Net, Resource Type-Net and Constraint-Net are constructed respectively. Based on the developed SOAC-Net, verification mechanism on the property of *authorization*

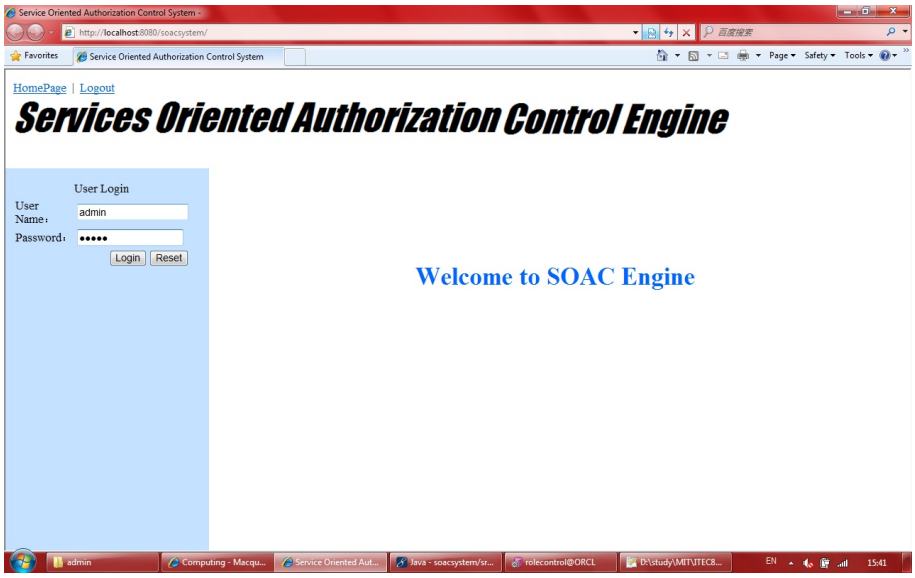

FIGURE 7.2: SOAC Engine

dead marking can be deployed. If improper authorization policy definition is identified, alert will be issued to notify the authorization decision.

`Authorization Decision Making` performs the policy compliance checking to make authorization decisions. The conflict of interest is checked and the verification on SOAC-Net is performed in this package.

- The `Authorization Enforcement` controls the enforcement of authorization decisions related with the composite web service and its component services. *Authorization Synchronization Policy* should be enforced when granting the authorizations.

7.3 System Demonstration

In this section, we will demonstrate the SOAC-Engine. In Fig. 7.2, an administrator interface is presented where the administrator can login into the engine by fill the user name and password.

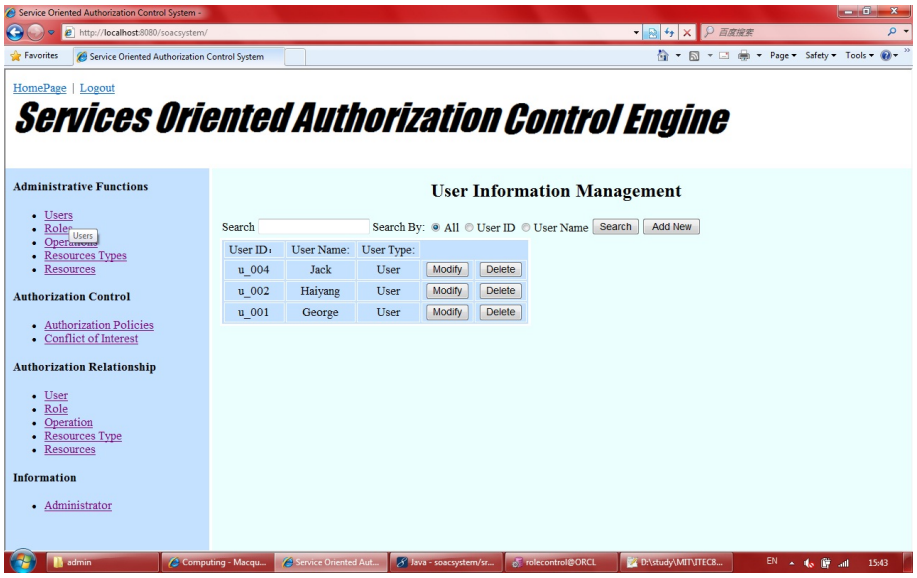

Figure 7.3: User Administration

The functions of SOAC Engine are separated into four parts, (1) Administrative Functions, (2) Authorization Control, (3) Authorization Relationship, and (4) Administrator (See the left-hand side at Fig. 7.3). The elements of SOAC and their relationships are designed by *Administrative Functions* and *Authorization Relationship*. Authorization policies and the design of conflict of interest are defined in *Authorization Control*.

In Fig. 7.3, we can add, modify, delete, and query the user, i.e the service consumer, in SOAC Engine. We can observe in Fig. 7.3 that a user name *Haiyang* is assigned a user ID u_002. In Fig. 7.4, all roles that have been defined in SOAC-Engine have been listed.

When clicking the *role* of *authorization relationship* at left-hand side of Fig. 7.5, a tree diagram is illustrated in Fig. 7.5, where all related authorization mappings of role r_001 are illustrated. That can be used by administrator to query what authorization mappings have been defined in SOAC-Engine. We can find that the user *Haiyang* with user ID u_002 has been assigned to role r_001 in the engine.

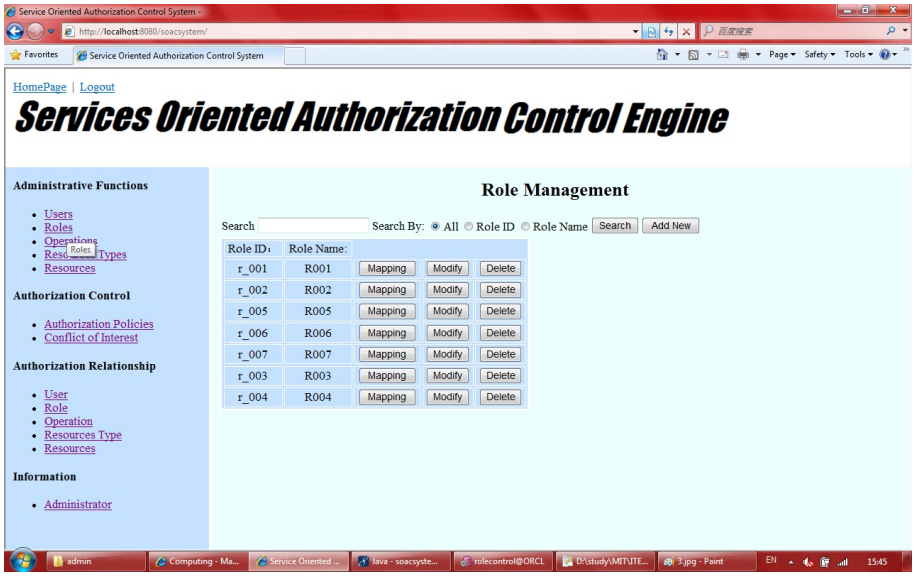

FIGURE 7.4: Role Administration

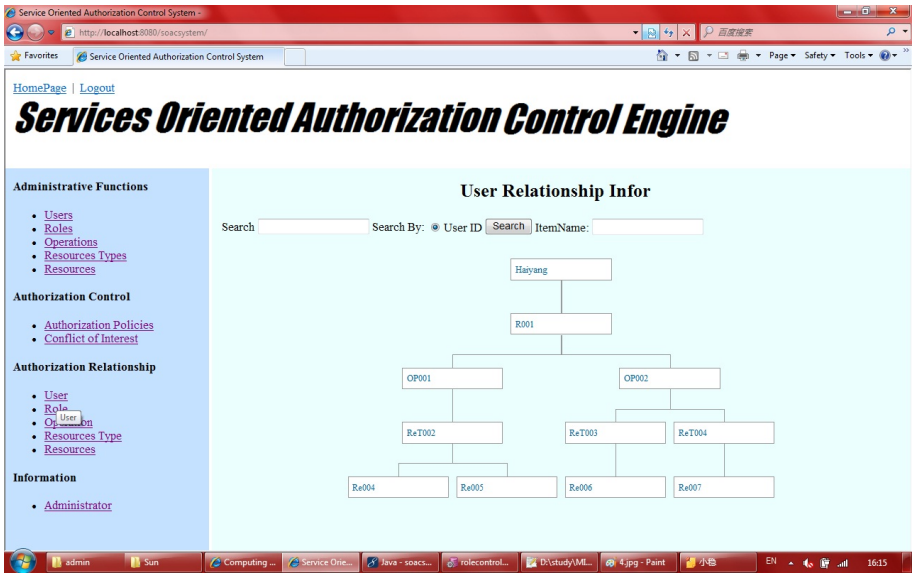

FIGURE 7.5: Authorization Mapping Tree

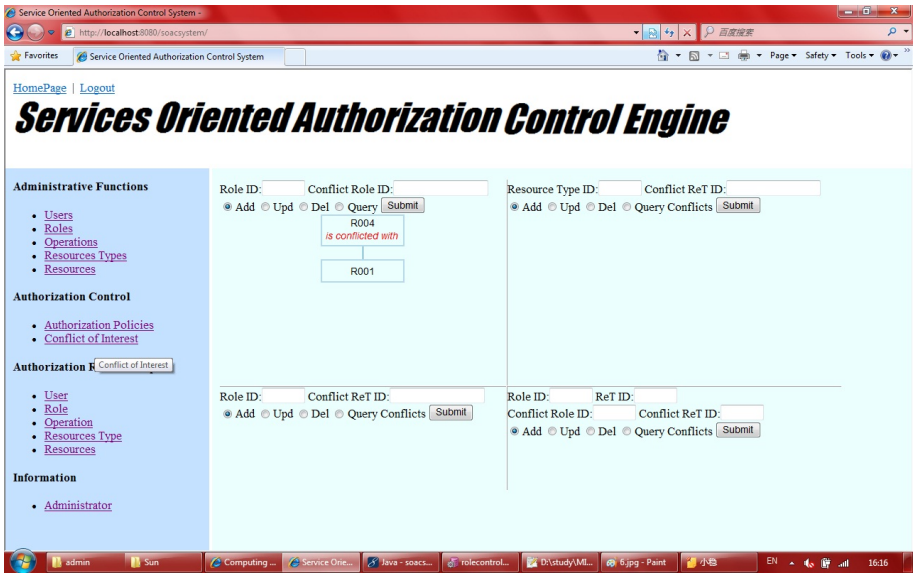

Figure 7.6: Conflict of Interest Identification

When you click the *conflict of interest* at *authorization control* at left-hand of Fig. 7.6, the various types of conflict of interest can be defined based on the business requirements. In Fig. 7.6, we can observe that role r_001 and role r_004 are conflicted with each other.

Hence, when we click `mapping` for r_004, then we can start to map related operation and user to this role (See Fig. 7.7). In Fig. 7.7, we can observe, that the user Haiyang with user ID u_002 will be mapped with role r_004, and role r_004 is mapped to the operation op_004 in the engine. However, since user u_002 has been assigned with role r_001 and role r_001 is conflicted with role r_004 (See Fig. 7.6), the user Haiyang with ID u_002 can not be assigned with conflicted role r_004. The information of unsuccessful mapping is shown in Fig. 7.8.

The group of conflict of interest can also be avoided in SOAC engine. In Fig. 7.9, we can observe that all elements related with user u_001(Gorge) are depicted in a tree diagram. For the group of u_001 and re_008, they have already been assigned with the group of role r_001 and resource type ret_002. In Fig. 7.10, all elements with role r_002 have been illustrated, where resource re_008 has been used to provide support as

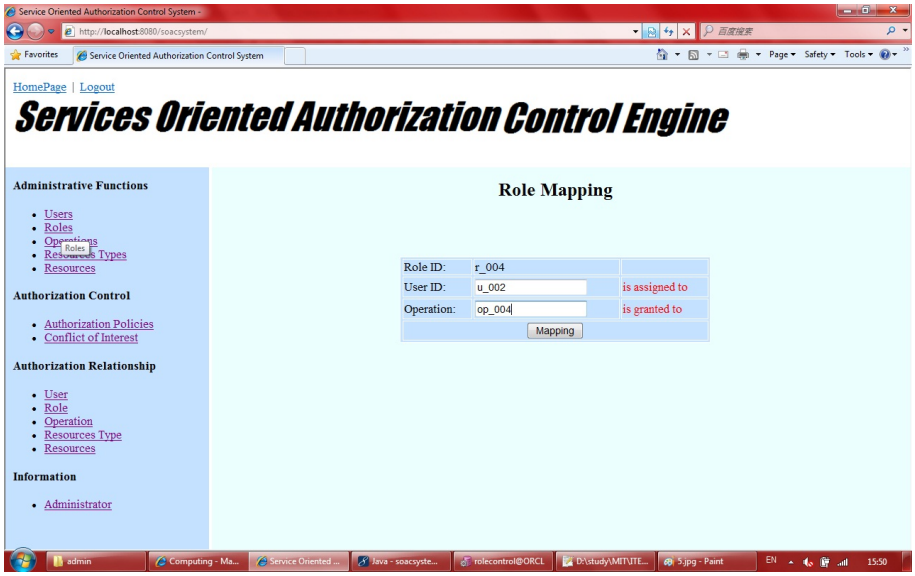

FIGURE 7.7: Role Mapping

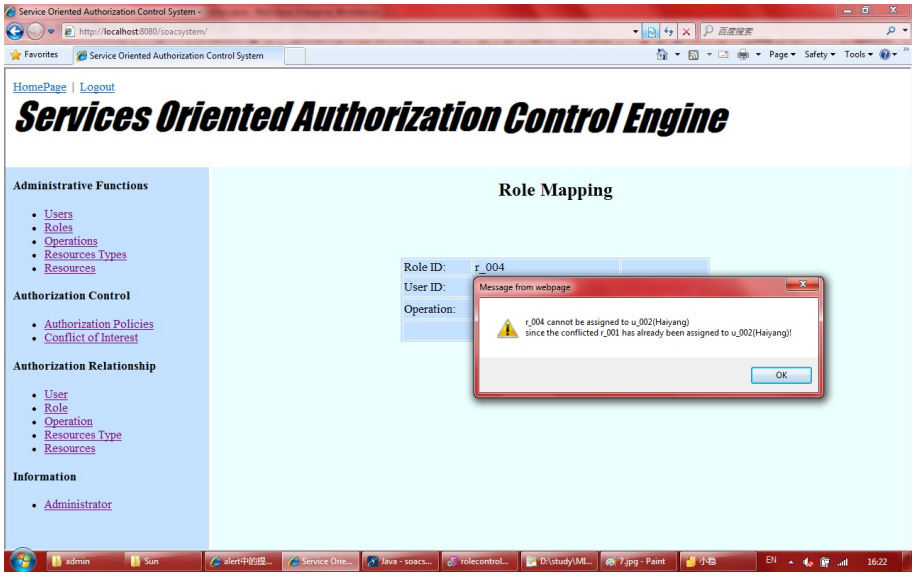

FIGURE 7.8: Bad Mapping

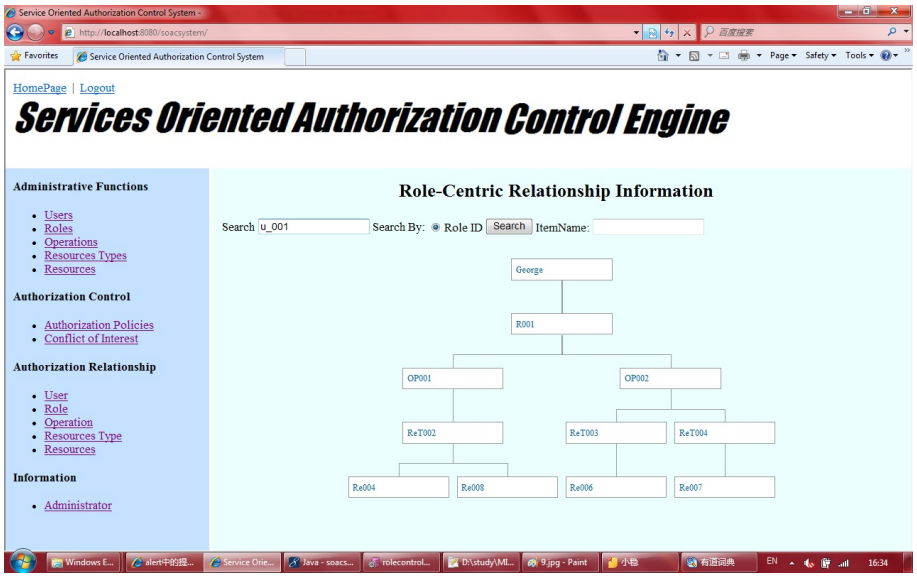

Figure 7.9: Authorization Mapping Tree 2

resource type ret_005.

In Fig. 7.11, the conflict of interest among the group of role and resource type is illustrated, e.g., $< R_2, ReT_5 >$ is conflicted with $< R_1, ReT_2 >$ and $< R_3, ReT_6 >$. When we want to map the user u_001 to role r_002 (See Fig. 7.12), the conflict of interest for group of user u_001 and resource re_008 occurs (See Fig. 7.13). The reason is that the group of user u_001 and resource re_008 has been mapped with group of role r_001 and resource type ret_002, and resource re_008 has been mapped with resource type ret_005. If user u_001 can be mapped to role r_002, then the conflicted groups of $< R_2, ReT_5 >$ and $< R_1, ReT_2 >$ are both assigned to the group of $< R_1, RE_8 >$, that can not be tolerated by SOAC engine (See Fig. 7.13).

In Fig. 7.14, a SOAC-Net simulation is illustrated when you click the Petri-Net in *authorization policy* at *Authorization Control*. In authorization policy, the dependency policy can be defined. Then the SOAC-Net simulator can model the SOAC-Net based on the defined authorization mappings and policies, and the business logic of composite

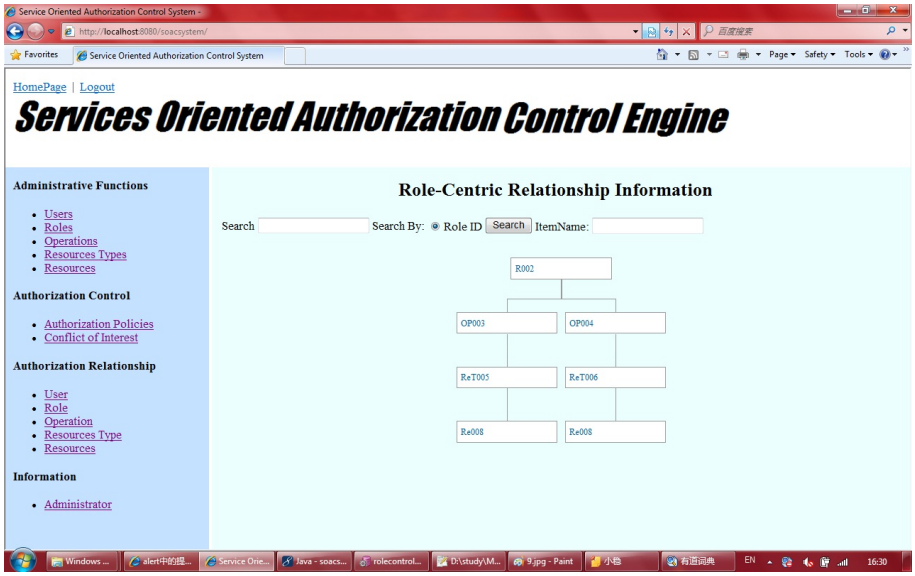

FIGURE 7.10: Authorization Mapping Tree 3

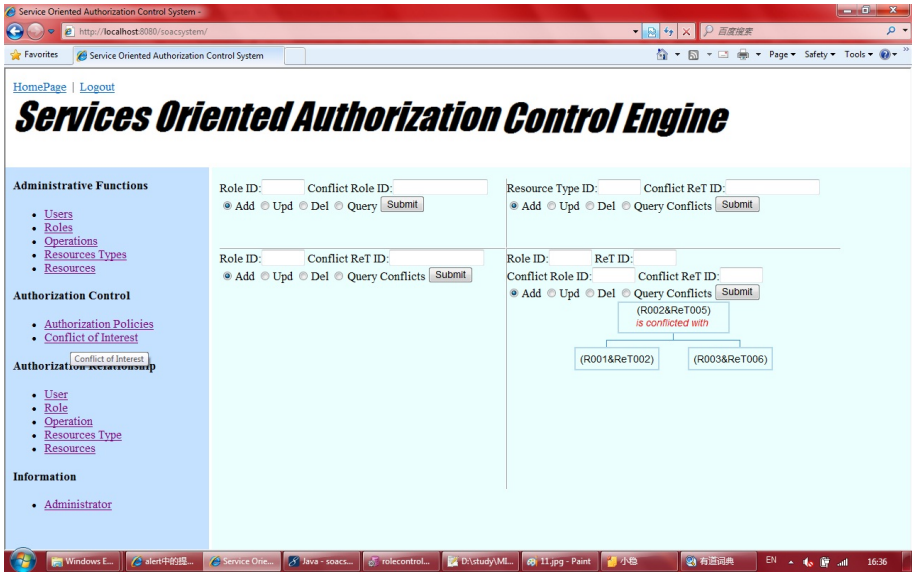

FIGURE 7.11: Grouped Conflict of Interest Identification

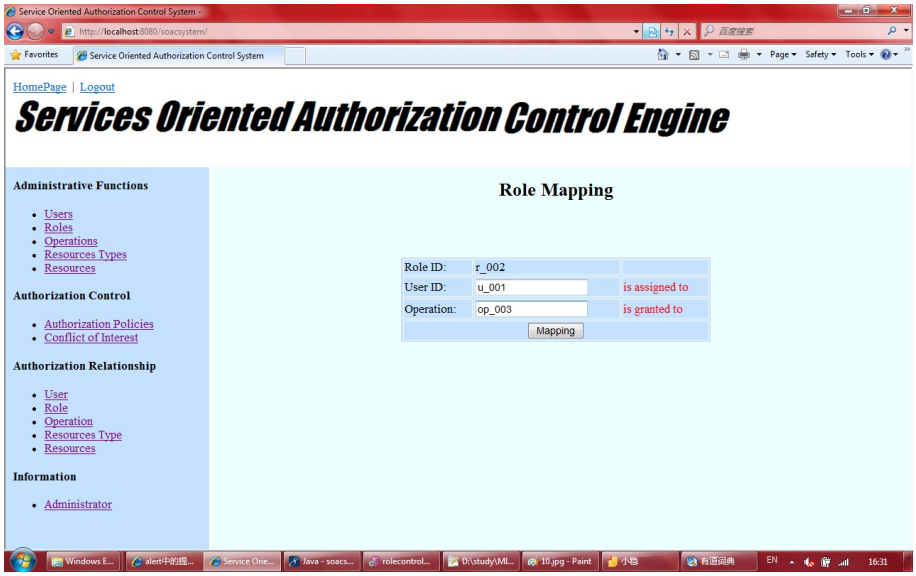

Figure 7.12: Role Mapping 2

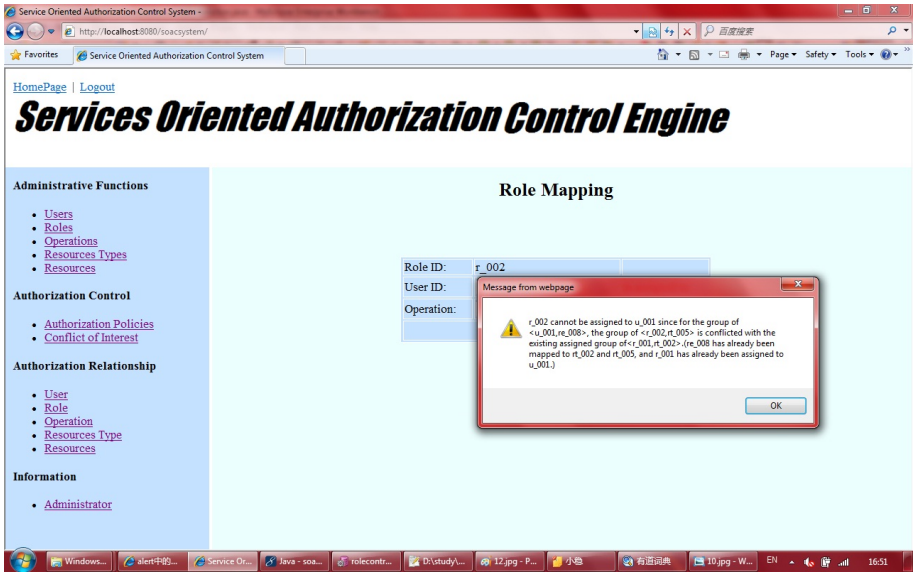

Figure 7.13: Bad Mapping 2

web service. In Fig. 7.14, we can observe that, the Petri-Net simulator is separated into five areas: *Menus*, *Main drawing*, *Information*, *Matrix computing*, and *Monitor*,

- *Menus* part consists of multiple icons to use the simulator, e.g. executing the SOAC-Net.
- *Main drawing* is used to display the modeled SOAC-Net.
- *Information* area in simulator is used to illustrate the properties of each object in the simulator, e.g., the properties of a place includes name, type, positions, index and so on.
- *Matrix computing* is used to display each step of marking. When the SOAC-Net runs, the marking will be changed according to each step of SOAC-Net. In Fig. 7.14, a marking M_0 is illustrated where only the position in marking for place p_1 and p_{11} are occupied by specific tokens. Note, E means $\emptyset$ in M_0 in Fig. 7.14. Since it is initial marking, the value of the token will be operation op_001.
- *Monitor* is used to display the verification result. If any authorization dead marking occurs at specific step of SOAC-Net, alert will appear at *Monitor* area to notify the administrator.

The Main Drawing area of SOAC-Net simulator is used to draw and display a SOAC-Net. After drawing, the authorization policies will be associated with each transition in SOAC-Net, and a labeled transitive matrix that is used to perform matrix calculation will be built by system.

7.4 Conclusion

In this chapter, we present an authorization engine, named SOAC engine based on our proposed conceptual model SOAC and process model SOAC-Net. Authorization policies, for conflict of interest, synchronization, and dependency, are all supported by the SOAC engine. In this chapter, we also demonstrate how the SOAC engine can

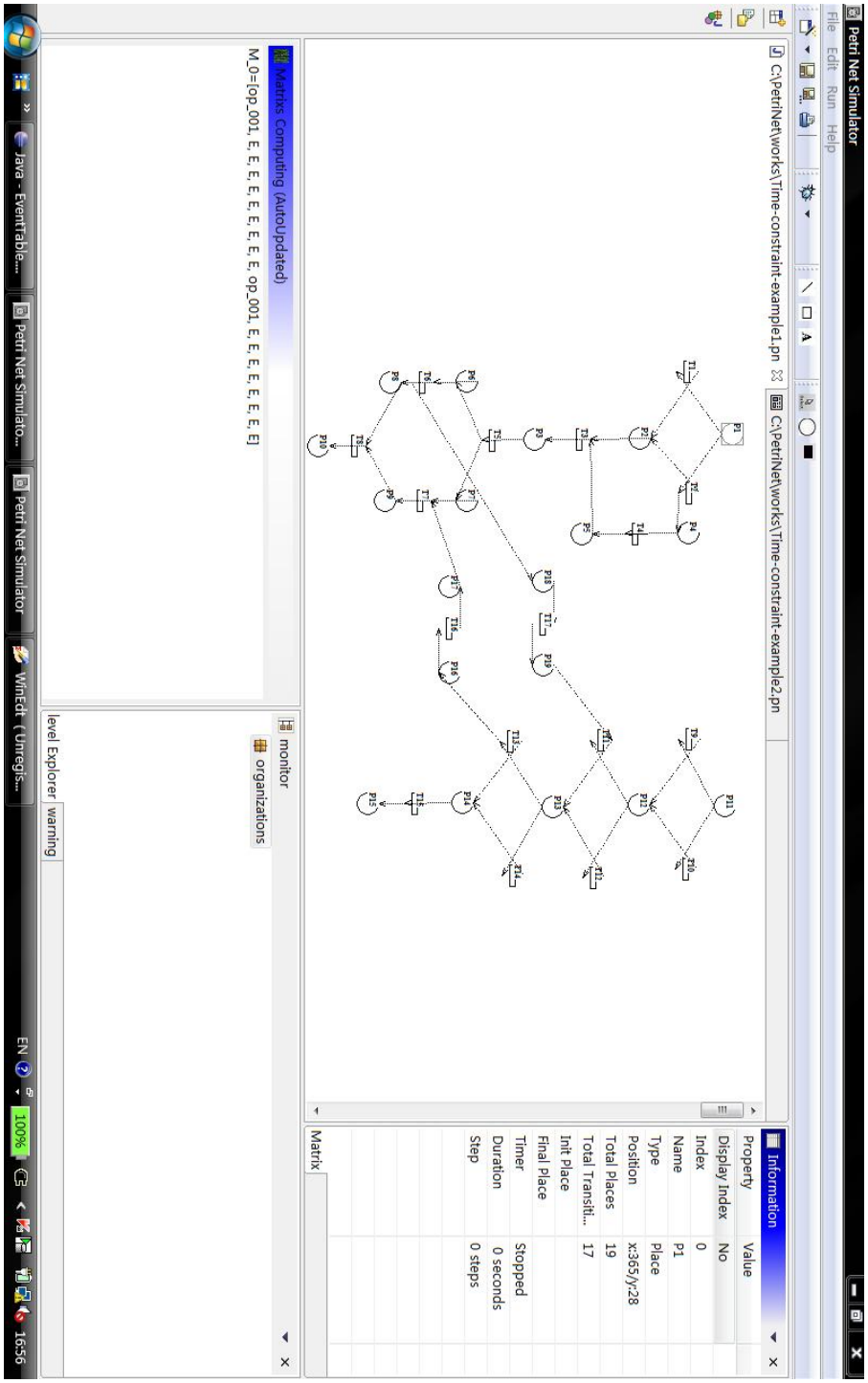

Figure 7.14: SOAC-Net Simulation

enforce authorization policies to avoid various types of conflict of interest. A Petri-Net simulator is presented as well in this chapter to describe how the SOAC-Net is modeled and how the verification based on SOAC-Net is performed.

8

Conclusion and Future Work

In this chapter, we summarize the contribution of this dissertation and discuss some future research directions.

8.1 Conclusion

In this dissertation, we propose a security management system in composite web service environment. Web service is autonomous and self-contained. Multiple web services can be composed together to provide comprehensive function to service consumer in a dynamic fashion and based on-demand. Hence, all web services are loosely-coupled. Based on such environment, security management, i.e. authorization management, to protect the information of the operations of composite web service becomes an issue that is not tackled in recent research domain. Five research problems in composite web

service authorization management are identified and solved by this dissertation.

- **Complicated Coordination of Authorization Policies**: The component services that can provide support to the operations of the composite web service have their own access policies. Granting the permissions to a service consumer on accessing the operations of the composite web service should not only confirm whether the service consumer is qualified, but also need to take access policy of the components service into account, i.e. the availability of the support from component service should be guaranteed. Obviously, without the support from specific component service, the operation that needs the support should not be accessed by the service consumer. A coordination on the authorization policies of composite web service and those of component service is required. The basic authorization question "`who can do what`?" should be changed to a more complicated one as "`who can do what under what kind of support`".

  In *chapter three*, a conceptual model named Service Oriented Authorization Control (SOAC) is proposed. In SOAC, an element named Resource is considered to provide support on specific operations of composite web service that need to be accessed by service consumer. Through this way, the support from component service is taken into account when deciding if the permission should be granted to service consumer.

- **Dynamicity of Component Services and Service Consumer**: A large amount of component web services can be used as resources to support the composite web service's operations; while the quantity of service consumers can also be large. Moreover, the resources and service consumers are normally prone-to-change. Since component services and service consumers are both out of control of composite web service, they can change their availability or authorization requirements at any time according to their own interests, without advanced notification to the composite web service (the relationship among web services are loosely-coupled.). If the changes occur frequently or happen in many web services, efficient authorization management is needed to reduce the administrative

overhead in authorization of composite web services.

In *chapter three*, the proposed SOAC conceptual model introduces two group concepts Role and Resource Type to encapsulate the large-quantity and prone-to-change service consumers and resources respectively. The operations of the composite web service are not mapped to service consumer and resource directly. Actually, they are mapped to these two group concepts. By this way, the frequent and large-amount changes of service consumer and resource can not affect the authorization management on the operation. Through managing the relationships between service consumer and role, and between resource and resource type, the administrative overhead is dramatically decreased as the relationships between role and operation, and the relationship between resource type and operation are kept stable.

- **Conflict of Interest**: The composite web service authorization management should avoid any conflict of interest. In composite web service environment, with the involvement of resource, the conflict of interest becomes complicated, in terms of authorization.

  In *chapter four*, we analyze each type of conflict of interest and provide relevant authorization rules to avoid conflict of interest. We summarize the conflict of interest in composite web service authorization from three factors as follows,

  - *Where does the conflict of the interest occur?*: The conflict of interest can occur at four places. (1) between service consumer and composite web service, i.e., at service provision part, (2) between resource and composite web service, i.e., at service realization part, (3) between resource, composite web service, and service consumer i.e., at integration part, and (4) between the groups of resource and service consumer, i.e, at service integration part.
  - *When does the conflict of interests occur?*: (1) The conflict of interest can occur at design time where no authorization should be given in terms of conflict of interest. (2) The conflict of interest can occur at run time. Sometimes, there is a tradeoff between the security and flexibility in business

requirement. In this case, the authorization that may cause conflict of interest can be granted. However, these authorizations can not be activated simultaneously so that the conflict of interest can be avoided at runtime.

- *What does the conflict of the interest include?*: (1) The conflict of interest may include several elements, e.g., several service consumers or resources. In this case, the conflict of interest is caused by the mismatch of the relationship between these elements. (2) On the other hand, since the relationships between the elements, e.g., between service consumer, are not easily to identify as the web services are autonomous and their relationships with other web services can not be identified. Hence, another type of conflict of interest which only includes one element, e.g., for one service consumer or for one resource, is proposed. This type of conflict of interest can be caused by the different authorizations for one element.

- **Compliance of Business Logic:** To achieve the composite service functions, a certain business logic is inherently set up within the composite service, which therefore requires these accesses of service consumers and supports of component services to comply with. A question that should be significantly marked is "*how the service consumer and the component service can pale others to be selected to work on the specific operations during the execution process of Composite web service*" on top of the question "what to do". The failure of coordinating on the large number of accesses of service consumers and supports of component services in the business process can cause various security issues.

  In *chapter five*, a process model named SOAC-Net is proposed based on the conceptual model of SOAC in chapter four. SOAC-Net is a Petri-Net based process model that can describe the access sequence of service consumer and the support sequence of resource according to the business logic of composite web service, i.e. *authorization synchronization policy*. The constraint-net that is used to model the *authorization dependency policy* is also included in SOAC-Net. Through enforcing these two types of authorization policies, SOAC-Net can

be used to manage the composite web service authorization management in a business process environment.

- **Authorization Policy Verification:** With more and more authorization policies are defined based on SOAC-Net to manage composite web service authorization, the possibility of defining an improper authorization policy becomes larger and larger. The improper authorization policy may cause the role access and resource support missing at the time when they are needed. Hence, the normal execution of composite web service can even be suspended due to the improper authorization policy definition.

  Therefore, in *chapter six*, we provide a verification mechanism to detect the improper authorization policy definition based on SOAC-Net. A novel property named *authorization dead marking freeness* is proposed. Through verifying this property, the detect of improper authorization policy definition is transferred to examine the marking of SOAC-Net, i.e, the each step of token movement in SOAC-Net. If a marking of SOAC-Net becomes a dead marking, the relevant authorization policy must be defined improperly.

Our proposed composite web service authorization management is realized by solving the above five identified research problems. Our solutions are embedded within an authorization engine, named SOAC engine, introduced in *chapter seven*. SOAC engine is developed based on the conceptual model SOAC and the process model SOAC-Net. All authorization policies in terms of conflict of interest, synchronization and dependency are implemented in SOAC engine. A Petri-Net model simulator is also encapsulated within SOAC engine to provide process modeling and policy verification function.

8.2 Future Work

In this dissertation, we have introduced a security system in terms of composite web service authorization management. We believe that this is an important research area,

which will attract a large number of attentions from the research community. This thesis raises a number of issues for future work in both practical and theoretical terms:

- **Hierarchy of Role and Resource Type**: In traditional RBAC, hierarchy of role is introduced in model of $RBAC_1$, which brings the system administrator a more flexible way to manage the access control. In hierarchy of role in RBAC, the permission of junior role can be inherited by the senior role; while the senior role also bears its exclusive permissions that can not be assigned to junior role. In this case, the system administrator only needs to design the inheritance of relationship between junior role and senior role, rather than repeatedly assigning the permission mappings of junior role to the senior one.

  In SOAC, the hierarchy of role is not included, nor contains the hierarchy of resource type. In the future, we will extend the conceptual model SOAC to include this significant feature of role-based access control model. We believe, that by introducing a new group concept Resource Type, the involvement of hierarchy of role and resource type can bring SOAC model more complicated.

- **More Types of Authorization Policies:** In this dissertation, we mainly focus on the authorization policies in terms of conflict of interest, synchronization, and dependency. However, the authorization policies in SOAC should not limited only for these three types.

  In the future, we will design more authorization policies in SOAC, e.g., *authorization binding policies*, and *authorization cardinality policies*, to provide comprehensive authorization management in composite web service environment.

- **Optimization Algorithm for Verification based on SOAC-Net:** SOAC-Net is a Petri-Net based process model, that can provide verification mechanism on improper authorization policy definition. However, the verification methods based on a Petri-Net based process model is time-consuming with the expansion of process model.

An optimization algorithm focusing on evaluating the complexity of existing verification techniques should be developed and implemented on SOAC-Net simulator in SOAC engine, where the performance of the SOAC-Net model can be simulated as well as optimized.

- **Resolution of Detected Improper Authorization Policies:** The verification mechanism proposed in this thesis can be able to detect the result of improper authorization policy definitions. However, the following steps to resolve the improper authorization policy definitions is still an issue that is not tackled. Redefining the authorization policy may cascading affect the existing authorization policies. Therefore, mechanism and algorithm should be developed to perform the policy oriented investigation regarding to correcting improper authorization policies.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: A research roadmap. International Journal of Cooperative Information Systems 17(2), 223 (2008). 1

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. Web Services - Concepts, Architectures and Applications (Springer, 2003). 2

[3] M. Singh and M. Huhns. Service-oriented Computing - Semantic, Processes, Agents (John Wiley and Sons Ltd, 2005). 2

[4] S. Dustdar and W. Schreiner. A survey on web services composition. International Journal on Web and Grid Services 1(1), 1 (2005). 2

[5] E. Newcomer and G. Lomow.:*Understanding SOA with Web Service, Pearson Education*, USA, 2005. 2, 49

[6] K. Gottschalk, S. Graham, H. Kreger, and J. Snell, "Introduction to Web Services Architecture," IBM Systems Journal, vol. 41, pp. 170-177, 2002. 2

[7] M. Sloman and E. Lupu, "Security and management policy specification," IEEE Network, vol. 16, pp. 10-19, 2002. 6

[8] C. Yang, "Designing secure e-commerce with role-based access control,"

International Journal of Web Engineering and Technology, vol. 3, pp. 73-95, 2007.

[9] J. S. Park and J. Hwang, "Role-based access control for collaborative enterprise in peer-to-peer computing environments," Proc of ACM symposium on Access control models and technologies, Como, Italy, pp. 93 - 99, 2003. 6

[10] Mike Havey.:*Essential Business Process Modeling*, O'Reilly, USA, 2005. 41, 49

[11] Wil M.P. can der Aalst, Arthur H.M. ter Hofstede, and Mathias Weske.: Business Process Management: A Survey, in Proceedings of International Conference on Business Process Management, Lecture Notes on Computer Science 2678, Springer, 2003, pp1-12. 49

[12] Michael P. Papazoglou and Willem-Janvan der Heuvel.: Web Services Management: A Survey, Internet Computing, IEEE Volume 9, Issue 6, Page(s):58-64, Nov.-Dec. 2005. 43

[13] Michael P. Papazoglou, and Jean-jacques Dubray.: A Survey of Web service technologies. Technical Report DIT-04-058, Informatica e Telecomunicazioni, University of Trento, 2004. 43

[14] Vil van der Aalst and Kees van Hee.: *Workflow Managmenet: Models, Methods and System*,MIT Press, 2002. 50

[15] Workflow Management Coalition. WFMC Home Page. http://www.wfmc.org. 50

[16] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju.: *Web Service: Concept, Architechture and Applications*, Spinger Press, 2004. 43

[17] Nilo Mitra and Yves Lafon.:Simple Object Access Protocol SOAP Version 1.2, http://www.w3.org/TR/soap12, 2007 44

[18] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana.: Web Services Description Language (WSDL) Version 2.0, http://www.w3.org/TR/2007/REC-wsdl20-20070626, June 2007. 44

[19] Luc Clement, Andrew Hately, Claus von Riegen, Tony Rogers.:Universal Description Discovery and Integration, http://uddi.org/pubs/uddi_v3, 2004. 44

[20] J. F. Chang, "Business Process Management System - Strategy and Implementation", Boca Raton, New York: Auerbach Publications, 2006. 49

[21] S. Roser and B. Bauer, "A Categorization of Collaborative Business Process Modeling Techniques," Proc of IEEE International Conference on E-Commerce Technology Workshops, Washington DC, USA, pp. 43-54, 2005. 49

[22] A. Lindsay, D. Downs, and K. Lunn, "Business Processes - Attempts to Find a Definition," Information and Software Technology, vol. 45, pp. 1015-1019, 2003. 51

[23] H. Smith and P. Fingar, Business Process Management: The Third Wave: Meghan-Kiffer Press, 2003.

[24] N. Berente, B. Vandenbosch, and B. Aubert, "Information flows and business process integration," Business Process Management Journal, Emerald Ltd, vol. 15, pp. 119-141, 2009. 51

[25] F. Casati and A. Discenza. Modeling and managing interactions among business processes. Journal of Systems Integration, 10(2):145-168, 2001. 50

[26] V. Popova and A. Sharpanskykh. Process-oriented organisation modelling and analysis. Enterprise Information Systems Journal, 2(2):157-176, 05 2008.

[27] A. Wombacher, B. Mahleko, and E. J. Neuhold. Ipsi-pf: A business process matchmaking engine. In 2004 IEEE International Conference on E-Commerce Technology (CEC 2004), 6-9 July 2004, San Diego, CA, USA, pages 137-145. IEEE Computer Society, 2004.

[28] A. Wombacher, B. Mahleko, and E. J. Neuhold. Ipsi-pf - a business process matchmaking engine based on annotated finite state automata. Inf. Syst. E-Business Management, 3(2):127-150, 2005. 50

[29] J. Wang, "A Web Services Secure Conversation Establishment Protocol Based on Forwarded Trust," Proc of International Conference on Web Services, Chicago, Illinois, USA, pp. 569-576, 2006. 5

[30] M. Shehab, K. Bhattacharya, and A. Ghafoor, "Web services discovery in secure collaboration environments," ACM Transactions on Internet Technology, vol. 8, 2007.

[31] R. Kanneganti and P. Chodavarapu, SOA Security: Manning Publications, 2008. 5

[32] W. Tolone, G.-J. Ahn, T. Pai, and S. P. Hong, "Access Control in Collaborative Systems," ACM Computing Surveys, vol. 37, pp. 29-41, 2005. 6

[33] M. T. Siponen and H. Oinas-Kukkonen, "A review of information security issues and respective research contributions," ACM SIGMIS Database, vol. 38, pp. 60-80, 2007. 6

[34] P. Chapin, C. Skalka, and X. S. Wang, "Authorisation in Trust Management: Features and Foundations," ACM Computing Surveys, vol. 40, 2008. 7

[35] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In Business Process Management, International Conference, BPM 2003, Eindhoven, The Netherlands, June 26-27, 2003, Proceedings, volume 2678 of Lecture Notes in Computer Science, pages 1–12. Springer, 2003. 49

[36] Alistair Barros, Marlon Dumas and Phillipa Oaks.: Standards for Web Service Choreography and Orchestration: Status and Perspectives, the Proceedings of the First International Workshop on Web Service Choreography and Orchestration for Business Process Management-BPM 2005. 46

[37] Chris Peltz.: Web Services Orchestration and Choreography, Computer, vol.36, no.10, pp.46-52, Oct., 2003. 47

[38] Remco Dijkman, and Marlon Dumas.: Service-oriented Design: A Multi-viewpoint Approach, Technical Report TR-CTIT-04-09 Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625, 2004. 2, 46

[39] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, and Yves Lafon.: Web Services Choreography Description Language Version 1.0 (WS-CDL), http://www.w3.org/TR/ws-cdl-10/, 2004. 49

[40] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek.:Web Service Choreography Interface (WSCI) 1.0, http://www.w3.org/TR/wsci ,2002. 48

[41] OASIS Web Services Business Process Execution Language (WS-BPEL) Technical Committee.: Web Services Business Process Execution Language Version 2.0 (BPEL4WS), http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html, 2007. 48, 57, 67

[42] W3C. Web Services Policy 1.2 - Framework (WS-Policy) 2006. 57

[43] W3C. Web Services Policy 1.2 - Attachment (WS-PolicyAttachment) 2006. 57

[44] C.A Petri, Kommunikation mit Automaten. PhD Thesis, Institute frinstrumentelle Mathematik, Bonn, 1962. 35, 63, 163

[45] C. Girault and R. Valk.:*Petri Nets for System Engeneering: A Guide to Modeling, Verification and Application*, Springer, German, 2003. 35, 63, 163

[46] T. Murata.:Petri Nets:Properties,Analysis and Application, Proceedings of the IEEE,77(4), IEEE, 1989, USA, pp.541-580. 35, 64, 65, 163

[47] Petia Wohed, Wil.M.P. van der Aalst, Marlon Dumas, and Arthur.H.M. ter Hofstede.: Analysis of Web Services Composition Languages:The Case of BPEL4WS, In Proceedings of the 22nd International Conference on Conceptual Modeling (ER), Chicago IL, USA, October 2003. Springer Verlag. 48

[48] P. Huber, K. Jensen, and R.M. Shapiro.: Hierarchies in Colored Petri nets, *Advances in Petri Nets 1990*, Springer, 1991, pp. 313-341. 65

[49] D.G. Stork, and R.J. van Glabbeek.:Token-controlled Place Refinement in Hierarchical Petri Nets with Application Active Document Workflow, the Int. Conf. on Application and Theory in Petri Nets, 2002, pp.394-413. 65

[50] Y. Song and J. LEE.:Deadlock Analysis of Petri Nets Using the Transitive Matrix, the SICE Annual Conf., 2002, pp.689-694. 64, 65, 66, 163

[51] M. Papazoglou, D. Georgakopoulos.: Service-Oriented Computing. Communications of the ACM **46**(10) (2003) 25–28. 2, 71

[52] RS. Sandhu, E. Coyne, H. Feinstein, C. Youman.: Role-based Access Control Models. IEEE Computer **29**(2) (1996) 38–47. 6, 51, 73

[53] D. Ferraiolo, J. Cugini, R. Kuhn.: Role Based Access Control: Features and Motivations. In: Proceedings of ACSAC. (1995). 51

[54] D. Ferraiolo, R. Sandhu et al.: Proposed NIST Standard for Role-Based Access Control. ACM Trans. on Information and System Security (TISSEC), **4**(3), 224-274, (2001). 58

[55] G. Ahn and R. Sandhu.: Role-Based Authorization Constraints Specification. ACM Transactions on Information and System Security (TISSEC), **3**(4), 207-226, (2000). 58

[56] Ninghui L., Mahesh V. T., and Ziad. B: On Mutually-Exclusive Roles and Separation of Duty. ACM Transactions on Information and System Security (TISSEC), **10**(2), (2007). 55

[57] Ninghui L., amd Qojia W.: Beyong Separation of Duty: An Algebra for Specifying High-Level Security Policies. Journal of ACM, **55**(3), (2000). 55

[58] Basit S., James B.D. J., Elisa B., and Arif G.: Secure Interoperation in a Multidomain Environment Employing RBAC Policies. IEEE Transactions on Knowledge and Data Engineering, **17**(11), (2005). 56

[59] Eric F., Tracy P., Lawrence P., Edward K., and Vijay K.: dRBAC: Distributed Role-based Access Control for Dynamic Coalition Environments. in Proceedings of the 22nd International Conference on Distributed Computing Systems, pp.411-420, 2002. 56

[60] R. Wonohoesodo, and Z. Tari.: A Role Based Access Control for Web Services. in Proceedings of SCC, (2004), 49–56. 23, 56, 73

[61] J. Fischer, and R. Majumdar.: A Theorey of Role Composition. in Proceedings of ICWS, (2008), 49–56. 23, 56, 73, 178

[62] P. Liu and Z. Chen. An Access Control Model for Web Services in Business Process In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, 2004. 57

[63] E. Bertino, J. Crampton, and F. Paci Access Control and Authorization Constraints for WS-BPEL In *Proceedings of ICWS*, 2006 57, 58

[64] F. Paci, E. Bertino, and J. Crampton.: An Access Control Framework for WS-BPEL. International Journal of Web Service Research 5(3) (2008) 20-43 57, 58, 178

[65] OASIS. "eXtensible Access Control Markup Language TC v2.0 (XACML)," http://docs.oasis-open.org/xacml/2.0/XACML-2.0-OS-NORMATIVE.zip Nov, 2007; 57

[66] Hristo K., and Fabio M.: Interactive Access Control for Web Services. In Proceedings of the 19th IFIP International Information Security Conference (SEC) 2004. 58

[67] Hristo K., and Fabio M.: An Access Control Framework for Business Processes for Web Services. In Proceedings of the 2003 ACM workshop on XML security, 2003. 58

[68] X. Wang, Y. Zhang, H. Shi, J. Yang.: BPEL4RBAC: An Authorisation Specification for WS-BPEL. In: Proceedings of WISE 2008. (2008) 381–395 58

[69] X. Wang, Y. Zhang and H. Shi.: Access Control for Human Tasks in Service Oriented Architecture, Proc of 2008 IEEE International Conference on e-Business Engineering (ICEB08), Xian China, pp 455–460, 2008. 58

[70] M. Mecella, M. Ouzzani, F. Paci, and E. Bertino.: Access Control Enforcement for Conversation-based Web Service. in Proceedings of the International World Wide Web Conference, (2006), 257–266. 23, 58, 73

[71] F. Paci, M. Ouzzani, and M. Mecella.: Verification of Access Control Requirements In Web Servies Choreography. in Proceedings of SCC, (2008), 5–12. 58

[72] C. Giblin and S. Hada.: Towards Separation of Duties for Services. the 6th Int. Workshop on SOA & Web Services Best Practices Committee, OOPSLA, October 19, 2008, Nashville. 58

[73] K. Knorr. Dynamic Access Control through Petri Net Workflows. In *Proceedings of ACSAC*, 2000. 59

[74] Vijayalakshmi Atluri and Wei-Kuang Huang.: An authorization model for workflows . in Proc. of the 4th European Symposium on Research in Computer Security (ESORICS'96), (1996). 23, 59, 60, 122

[75] Zhang Yi, Zhang Yong and Wang Weinong.: Modeling and Analyzing of Workflow Authorization Management. Journal of Network and Systems Management Volume 12, Number 4, 507–535, (2004). 23, 60, 122

[76] Tan, K., Crampton, J, and Gunter, C.A.: The consistency of task-based authorization constraints in workflow. 17th IEEE Workshop of Computer Security Foundations, 155–169, (2004). 23, 60, 123

[77] Reinhardt A. Botha, and Jan H. P. Eloff.: Separation of duties for access control enforcement in workflow environments. IBM Systems Journal Volume 40, Number 3, (2001). 60, 61

[78] Fan Hong and Guang-lin Xing: A Family of RBAC-Based Workflow Authorization Models. Wuhan University Journal of Natual Sciences Volume 10, Number 1, (2005). 61

[79] Guang-lin Xing, Fan Hong and Hui Cai: A Workflow Authorization Model Based on Credentials. Wuhan University Journal of Natual Sciences Volume 11, Number 1, (2006). 61

[80] Bertino, E., Ferrari, E., and Atluri, V.: The Specification and Enforcement of Authorization Contraints in Workflow Management Systems. ACM Transactions on Information and System Security (TISSEC), Volume 2, Number 1, pp.65–104, (1999). 61

[81] Le Yang and Yongsun Choi.: A Flexible Access Control Model for Dynamic Workflow Using Extended WAM and RBAC. The 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD), pp.488–497, (2007). 61

[82] Chritian Wolter and Andreas Schaad.: Modeling of Task-Based Authorization Constraints in BPMN. in Proceedings of The International Conference on Business Process Management, pp.64–79, (2007). 61

[83] Y. Chi, M. Tsai and C. Lee.: A Petri-net Based Validator in Reliability of a Composite Service. the IEEE Int. Conf. on e-Technology, e-Commerce and e-Service, 2005. pp.450-453. 67, 163

[84] Y. Yang, Q. Tan, J. Yu and F. Liu.: Transformation BPEL to CP-Nets for Verifying Web Service Composition, the Int. Conf. on Next Generation Web Services Practices, 2005. 67

[85] Y. Yang, Q. Tan, Y. Xiao, J. Yu and F. Liu.: Exploiting Hirachichical CP-Nets to Increase the Reliability of Web Services Workflow, the Symposium on Application and the Internet, 2006. 67

[86] X. Yi and K.J. Kochut.: A CP-nets-based Design and Verification Framework for Web Services Composition, the IEEE Int. Conf. on Web Services, 2004. 68, 163

[87] Van der Aalst, Wil.M.P.:The Application of Petri Nets to Workflow Management, The Journal of Circuits, Systems and Computers, 8(1), pp. 21-66, 1998. 68, 123

[88] W.M.P. van der Aalst.: Verification of Workflow Nets, the 18th Int. Conf. on Application and Theory in Petri Nets, 1997, pp. 407-426. 68, 163

[89] R. Hamadi, and B. Benatallan, "A Petri Net-based Model for Web Service Composition", Proceedings of the 14th Australian Database Conference, 2004,pp. 191-200. 68

[90] J. Zhang, C.K. Chang, J.Y. Chund, and S.W. Kim, "WS-Net: a Petri-net Based Specification Model for Web Services", ICWS, 2004, pp. 420 - 427. 68

[91] H. Sun, W. Zhao, J. Yang, and G. Shi, "SOAC Engine: A System to Manage Composite Web Service Authorization", in Proceedings of the 12th International Conference on Web Information System Engineering,(**WISE 2011**), Sydney, Australia, October 2011, Springer. 177

[92] H. Sun, W. Zhao, and J. Yang, "Managing Conflict of Interest in Service Composition", in Proceedings of 18th International Conference on Cooperative Information Systems (**CoopIS 2010**), Crete, Greece, October 2010, Springer. 177

[93] H. Sun, W. Zhao, and J. Yang, "SOAC: A Conceptual Model to Manage Service Oriented Authorization", in Proceedings of the IEEE International Conference on Service Computing (**IEEE SCC 2010**), Miami, USA, July 2010, IEEE Computer Society. 177

[94] S. Hinz, K. Schmidt, and C. Stahl, "Transforming BPEL to Petri Nets", the Int. Conf. on Business Process Management, 2005. 67

[95] Haiyang Sun, Weiliang Zhao, Jian Yang, and Jianwen Su, "TiCoBTx-Net: A Model to Manage Temporal Consistency of Service Oriented Business Collaboration", **IEEE Transactions on Services Computing**. (Accepted) 65

[96] Haiyang Sun, Jian Yang, and Weiliang Zhao, "A Temporal Rule-based Verification System for Business Collaboration Reliability", **Journal of Theoretical and Applied Informatics**, 16(2):65-68, 2009 65

[97] Haiyang Sun, Jian Yang, Xin Wang, and Yanchun Zhang, "A Verification Mechanism for Secured Message Processing in Business Collaboration", in Proceedings of the Joint International Conferences on Asia-Pacific Web Conference (**APWeb 2009**) and Web-Age Information Management (**WAIM 2009**), Suzhou-China, April 2009, Springer. 66

[98] H. Sun, X. Wang, J. Yang, Y. Zhang.: Authorization Policy Based Business Collaboration Reliability Verification. In: Proceedings of ICSOC. (2008) 66

[99] M. Wu, J. Chen, and Y. Ding, "Study on Role-Based Access Control Model for Web Services and its Application," Proc of International Conference on Telecommunications and Informatics, Istanbul, Turkey, 2006. 5

[100] R. Simon and M. E. Zurko, "Separation of duty in role-based environments," Proc of 10th Computer Security Foundations Workshop (CSFW'97), Rockport, MA, USA, pp. 183 - 194, 1997. 55

[101] H. K. Kim, R. Y. Lee, and H. S. Yang, "Frameworks for Secured Business Process Management Systems," Proc of Fourth International Conference on Software Engineering Research, Management and Applications (SERA '06), pp. 57-65, 2006. 20

[102] J. Yang and M. P. Papazoglou. Service components for managing the life-cycle of service compositions. Journal of Information Systems 29, 97 (2004). 46

[103] S. Chandrasekaran, G. Silver, J. A. Miller, J. Cardoso, and A. P. Sheth. Web service technologies and their synergy with simulation. In Proceedings of the 2002 Winter Simulation Conference (WSC '02), pp. 606-615 (San Diego, California, USA, 2002). 45

[104] A. Wombacher and M. Rozie. Piloting an empirical study on measures for work-flow similarity. In 2006 IEEE International Conference on Services Computing (SCC 2006), 18-22 September 2006, Chicago, Illinois, USA, pages 94-102. IEEE Computer Society, 2006. 50