Brno University of Technology
Faculty of Electrical Engineering and Computer Science

Department of Computer Science and Engineering

Authorization Model for Strongly Distributed Information Systems

Doctoral Thesis

Brno 2000 Daniel Cvrček

Authorization Model for Strongly Distributed Information Systems

by

Daniel Cvrček
Ing., Brno University of Technology, 2000

Submitted to the *Department of Computer Science and Engineering* in partial fulfillment of the requirements for the degree of Doctor of Philosophy at

Brno University of Technology
Faculty of Electrical Engineering and Computer Science
December 8, 2000

Signature of the Author ....
Department of Computer Science and Engineering
December 8, 2000

Certified by ....
(Thesis Supervisor)
Tomáš Hruška, Associate Professor
Faculty of Electrical Engineering and Computer Science, Brno University of Technology

Accepted by ....
Tomáš Hruška, Associate Professor
Faculty of Electrical Engineering and Computer Science, Brno University of Technology

Authorization Model for Strongly Distributed Information Systems

by

Daniel Cvrček

Submitted to the *Department of Computer Science and Engineering*
on December 8, 2000, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The title of the thesis contains words *strongly distributed systems*[1]. This means information systems that are able to process tasks that may be distributed not only in space, but also in time. What we are going to discuss are problems related to processing tasks with long time durability those processing is performed in multiple information systems with their own administration. More and more often we can hear about *Workflow management systems* (WfMS) that are used to automate business activities. Those information systems are typical systems with the above mentioned properties and the thesis uses both terms as synonyms.

Today, WfMS represent an important, inter-disciplinary area which is commercially significant. The widespread use of workflow applications has lead to increased awareness of data security issues. This dissertation identifies levels of access control and authorization requirements. Specifies *discretionary* and *mandatory* access controls for the workflow systems and proposes suitable security model. The thesis proposes a formal framework, based on *process algebra - Calculus of Communicating Systems* (CCS), for modeling and analyzing of security properties of WfMS. The specific contributions of the thesis are as follows:

1. Active authorization model (AAM) is proposed. The model provides basic features necessary for synchronization of authorization flow with

[1]The notion *strongly distributed system* is used throughout this thesis for systems that are distributed in space (there are several autonomous nodes that communicate) and that are able to execute computational tasks that have long lifetime and that may be executed on several autonomous nodes

workflow, separation of duties[2], event-based authorization and abstraction of access rights necessary for heterogeneous environments.

2. The extension of CCS is defined that allows specification of security properties of processes. That specification is divided into two distinct domains - *basic* and *mandatory*. The extension of CCS is constructed in such a way that bisimulations may still be used for process equivalence decisions.

3. Mechanism for process consistency testing is featured to decide feasibility of the process execution.

Thesis Supervisor:	Tomáš Hruška
Title:	Associate Professor
	Faculty of Electrical Engineering
	and Computer Science,
	Brno University of Technology

[2]Further information may be found in [62].

Acknowledgements

I would like to thank some of people who helped me to finish this work and reach the top of the official education, I could ever dream of. The most important person for this dissertation to appear (except me) is my supervisor Tomáš Hruška. He does not specialize in security but his guidance and help allowed me to publish some of my results on various conferences and finish the thesis in relatively short time.

I also have to thank Petr Kolenčík, Wil van der Aalst and Twan Basten who spared part of their time to read the thesis, give me comments and encourage me in my work.

At last, but not at least, I thank Douglas Adams for writing his trilogy in four parts *The Hitchhikers Guide to the Galaxy*. The book has changed my view on the world around and has made me to find out that there are probably more important thinks in our Galaxy.

Contents

1 Introduction **1**
1.1 Problem Statement 2
1.1.1 Security in Database Systems 2
1.1.2 Security Overview of WfMS 4
1.2 Research Issues 5
1.3 State of the Art 6
1.4 Overview 9

2 Background **11**
2.1 Security Policies 11
2.2 Access Control Models 12
2.2.1 Discretionary Access Control 12
2.2.2 Mandatory Access Control 13
2.2.3 Role-based Access Control 15
2.3 Related Research 16

3 Overview of CCS **19**
3.1 Basic Definitions 19
3.1.1 Action and Transition 20
3.1.2 Basic Language 20
3.1.3 Transitional Semantics 21
3.1.4 Derivatives and Derivation Trees 22
3.1.5 Sorts 23
3.2 Equational Laws 24
3.2.1 The Dynamic Laws 24
3.2.2 The Expansion Law 26

3.2.3 The Static Laws 27
3.3 Strong Bisimulation and Strong Equivalence 29
3.3.1 Strong Bisimulation 30
3.3.2 Basic Properties of Strong Bisimulation 30
3.4 Bisimulation and Observation Equivalence 31
3.4.1 The Definition of Bisimulation 31
3.4.2 Properties of Bisimilarity 33
3.5 The Theory of Observation Congruence 34
3.5.1 Experiments and Substitutivity 34
3.5.2 Equality and its Basic Properties 35

4 Architecture of the Authorization System 37
4.1 Basic Structure 38
4.2 Communication of the Layers 40
4.3 The Conversion Layer 42

5 Active Authorization Model 45
5.1 Distributed System 45
5.2 Basic Concepts of the Model 47
5.3 The Model Definition 49

6 Incorporating Security into CCS 57
6.1 Refinement of AAM Principles 57
6.1.1 Access Control and Flow Control 58
6.1.2 Categorization of System Elements 59
6.1.3 Abstraction and Specialization of Authorizations 62
6.2 Definition Of Secure System 66
6.2.1 Access Control of Secure System 67
6.2.2 Flow Control of Secure System 68

7 Secure CCS 69
7.1 Action and State 70
7.2 The Basic Language 73
7.3 The Expansion Law 75
7.4 Strong Bisimulation 77

7.5 Bisimulation and Observation Equivalence 81
7.5.1 Basic Properties of Bisimilarity 85
7.5.2 Further Properties of Bisimilarity 87
7.6 The Theory of Observation Congruence 88
7.6.1 Axioms for Finite Agents 91

8 Inheritance of Behavior **95**
8.1 Extension of Secure CCS 96
8.1.1 Axioms 97
8.1.2 Equality 97
8.2 Definition of Inheritance 99
8.3 Properties of Inheritance 101
8.4 Further Remarks 104

9 Conclusions and Directions **107**
9.1 Conclusions 107
9.2 Future research 109
9.2.1 Further Development of Authorization System 109
9.2.2 Dynamic Task Dependency 110
9.2.3 Web-based Implementation 110
9.2.4 Cryptography 111

Chapter 1

Introduction

Workflow management is a fast developing technology that has gained increasing attention recently. Workflow management systems are very suitable for the business environment because they provide powerful solution for effective administration of business processes, for re-engineering and allow rapid changes of business processes and environment.

A *workflow management system* (WfMS) is a system that supports specification, execution, coordination and management of workflow processes through the execution of software [38]. Workflow management systems are already used in a wide variety of commercial sectors, e.g. manufacturing, healthcare, banking, insurance companies, service order processing, collaborative software environment, telecommunications, office automation, and so on. Workflow management is a new area and therefore an emerging new area of research. Standardization effort undertaken by the Workflow Management Coalition (WfMC) is also in the very beginning [38, 3].

A workflow is usually defined as a sequence of tasks executed by automated systems or people to carry out a process. These processes are built by linking together diverse activities and specifying the flow of data and control among them. There is no specification where activities are to be executed nor which particular subject is to perform them. The WfMS may be strongly distributed and there is no way to perform administration of such a system from one place. It means that the whole system is composed from a number of *autonomous* workflow management systems that are administered by different persons with different policies. They may also be based on different software and hardware platforms. What we are interested in is security of such systems.

Talking about security of information systems, three information *properties* are usually meant: confidentiality, integrity and availability. Confidentiality aims at preventing unauthorized disclosure of information, integrity ensures correctness of the data [53, 55] and availability regards accessibility of data by authorized subjects (humans, processes, etc.).

The truth is that security of information systems has not gained larger attention until about ten years ago. The main aim of developers and designers was to ensure functionality of information systems and only increasing security risks moved eyes of researchers into the direction of security [13, 12, 60, 54]. Workflow management systems are the same story. Commercial products are improving functionality, but security is in the napkins.

1.1 Problem Statement

Despite increasing research efforts, commercial products by software vendors and standardization efforts by Workflow Management Coalition, security aspects in WfMS have not been given enough attention.

In the following, we specify security requirements for database and information systems.[1] We elaborate the problems encountered in enforcing security in strongly distributed information systems.

1.1.1 Security in Database Systems

Let us take a look at list of security areas that should be covered by database systems [23] (it is not the only one classification, see e.g. [51]). I think that it is a very well done list that may be used for all information systems today.

1. Protection from improper access - It means that access to the data is granted only to authorized users. The control is performed on smaller objects (record, attribute, value).

2. Protection from inference - Precaution of inference of confidential information from the open ones. It is important for the statistical databases.

3. Integrity - Integrity is partially guaranteed by the system measures of database management system (e.g. atomicity of transactions) and by the backup and recovery procedures and partially by the security procedures.

4. Operational integrity of data - It is logical consistence of data during concurrent transactions (concurrency manager), serializability and isolation of transactions (locking mechanisms).

[1]What is information and what database system nowadays? It seems to me that boundaries become vague and in some situations, the notions may be interchanged.

5. Semantical integrity of data - It ensures that attribute values are in allowed ranges. The control is enforced by integrity constrains.
6. Accountability and auditing - This requirement is composed from possibility to log all data accesses and from analysis of the logged events.
7. User authentication - Unique identification of database users. User identification is a basis of all authorization mechanisms.
8. Management and protection of sensitive data - The access is granted only to a narrow group of users.
9. Multilevel protection - We mean partitioning of data according to their sensitivity and access control differentiated according to that partitioning.
10. Confinement - It is necessary to avoid unauthorized information flow among processes (authorized channels, memory channels, covert channels).

Assurance of the previous security requirements is achieved by three types of security controls. They are:

1. Flow control - regulates distribution (flow) of information among available objects. E.g. reading of an information from an object X and its direct writing into an object Y. Policies for the flow control need a list or constrains of allowed flows. Flow control is very often solved by classification of system elements and definition of allowed flows between different classification classes.
2. Inference control - with aim to protect data from indirect detection [34]. It means that a set of data X that is available for a user, can be used to determine data set Y ($Y = f(X)$) that is directly unavailable. There are three basic ways how to realize the inference: (1) data correlation when data sets are semantically connected, (2) recognition of missing data that allow detection of existence of hidden data and (3) statistical inference of information.
3. Access control - is used for enforcing rules of security policies for all direct accesses to objects in the system. We are usually working with notions subjects, objects and operations. It is composed from two parts: set of access policies and rules (allowed accesses) and set of control procedures - security mechanisms (perform control of access requests according to policies).

Because we are not interested in statistical databases too much, only two controls are important for us - flow and access control. Although the previous classification has not been built up for workflow systems it is very suitable for our purposes.

1.1.2 Security Overview of WfMS

Strongly distributed systems are very complex for security management. The first thing we have to do is to split the distributed system into *homogeneous* (from the security point of view) parts that are centrally administered. We shall call them *autonomous information system* or *s-node*. Distributed system is then composed form a number of s-nodes that have to securely cooperate. To ensure such a secure cooperation, we have to solve several separate problems:

1. Access control in autonomous system - This problem comprises access control to resources local in the s-node and is solved by access control model implemented in the local platform (operating system, database management system, ...) the s-node uses. Each autonomous system may have another access control model. And even in the case the access models are the same, the security policies used for s-nodes may differ and may therefore be incompatible.

2. Global administration of system - Distributed system has to solve, somehow, problems with heterogeneity of its s-nodes and enforce uniform administration of security properties. This demands some sort of backbone, some basis that may be used as referential by all s-nodes.

3. Flow control - We are talking about systems that allow space distributivity of computational tasks. Those tasks use data with different sensitivity, stored on many s-nodes. There has to be a common template, common rules for data flow control, some *reference monitor* [16].

The work [39] uses notions discretionary and mandatory security for WfMS. I do not think that this is the correct partitioning. Discretionary access control means that access control is enforced by the owner of the resource, but it is not what the security in WfMS is about. On the other side, we may discern several layers in security of a distributed system.

1. Access control to resources on s-node - It is the lowest layer. We may use discretionary access control or mandatory access control or anything else. Not users, but tasks (that are invoked by users or other tasks) access resources according to workflows' definitions. The principal problem is to replace a task with the proper user on the s-node.

2. Access control to resources in workflow - It is higher layer. We have to define authorizations in workflow in some common way. One may say, there is no difference between this and the previous item, but the opposite is true. The difference is in the way, access rights are specified (s-node vs. distributed system). And the conversion between those two ways is the biggest problem for security in heterogeneous systems.

3. Access control to workflows - It comprises management of activation of computational tasks by users and other tasks. This is again more abstract view on access control.

4. Flow control of workflows - This is in the same layer as access control in workflows. We may say that access control and flow control are two sides of the same coin.

You can see that *discretionary* may be used only in the lowest level of access control. The mandatory access control may be on the other side used for access control to workflows and to local resources, but does not have to. We may use more relaxed or totally different policies. (The WfMS security is very often connected with role-based access control [18, 37, 63, 57, 22, 30, 59]).

1.2 Research Issues

Defining security for distributed information systems that allow long-timed computational tasks that are executed on several s-nodes needs a formal model that will settle rules for secure cooperation of various access control models. Design of such a formal model covers number of research issues:

1. Definition of methodology that sharply discerns security problems that shall be solved on particular layers of security model of distributed system. The layers shall be specified correctly.

2. A formal security model for task-based authorizations does not exist. Such a theoretical model is necessary to provide a basis for formal proofs that enable to establish higher confidence in the correctness of the workflow.

3. A formal framework that covers all layers of security in the heterogeneous distributed system's environment does not exist. We may identify the following areas of problems according to layers of security abstraction.

    (a) Access control to resources in computational tasks and especially its cooperation with underlying access control models is not solved.

It is necessary to ensure secure transition of access privileges from global definitions in computational tasks into definitions usable in s-nodes' access control systems and vice versa. This transition must have properties that prevent obtaining access rights by unauthorized subjects (users, tasks), but also ensures successful execution of tasks.

(b) Control of information-flow and authorization-flow has to be maintained in such a way that allows enforcement of mandatory access control when necessary. This control is based on classification of resources and subjects. The constrained information flow must ensure:

   i. Confidentiality - the task execution does not give raise to direct or indirect illegal disclosure of sensitive information.

   ii. Integrity - the task execution does not allow any unauthorized changes of data.

   But the controls still have to ensure availability of resources that are necessary for execution of tasks and should protect computational tasks from delays and starvation.

   We have got three basic properties of secure data access: confidentiality, integrity and availability. It is natural that availability goes down any time we increase control of confidentiality or integrity of resources. Further, confidentiality and integrity are opposites (in some sense) and their demands go against each other. It means that any independent tightening of requirements on confidentiality or integrity worsens availability.

   This is the biggest problem of mandatory access control. It is so tight that its usage in commercial environment is ineffective or even impossible. It has been the reason for role-based access control to appear.

(c) Access rights for activation of workflows (computational tasks) have to be solved in such a manner that allows effective access control and tasks' distribution throughout the distributed system.

1.3 State of the Art

Workflow technology is widely presented by many software vendors for at least five years. But when I try to compare my vision of workflows with existing technology, I have to remember a bank that calls its cards with magnetic strip as *electronic wallet*.

There is a number of products in the market. It is uneasy to name all of them so just the most important: Exchange Server (Microsoft), Lotus Notes (Lotus), GroupWise (Novell), Vision Office (IBM), some applications on SAP system, and we may name some more not so important, such as Staffware Workflow (Cimage Enterprise Systems), VisiFlow (Datamax Technologies), Step2000 (Document Imaging Solutions), Keyflow (DSSI) or Team-Leader (Logic Systems) and many others. Those are off-the-shelf products that are in some sense usable at large. We should not forget number of systems that are implemented and tailored to particular enterprises and that have to solve problems that are unique.

Existing technologies and products support a wide range of business applications such as banking, insurance, military, office automation, document processing or manufacturing. There may be found five categories [39]:

1. *Document based* systems use some global database of documents that are managed by a number of users. The database is the central point around which routing of electronic documents among participated individuals is performed.

2. *E-mail based* systems use e-mail programs as a medium for distribution of tasks among group(s) of users. E.g. Lotus Notes and Exchange Server use e-mails very strongly.

3. *Groupware based* products offer features that improve communication and cooperation of groups of users that cooperate on common tasks.

4. *Transaction based* systems use properties of database management systems to support some business transactions.

5. *Internet based* applications use Internet as the environment that allows very easy communication and that offers uniform basis for easy implementation of various tasks. WWW environment is developing very fast and offers more and more sophisticated instruments.

Although the products are from tiny to very complex, we can find some general properties. They assume some sort of unified environment. It means that they create their own resources (have a database) or when accessing resources of the underlying platform they can be used just on this platform (Exchange Server and Windows NT). It means that they need a homogeneous environment.

I have not found a product that would perform flow control, all products incorporate some type of access control based on HRU model (matrix based) or on its variations - i.e. they use of access control lists.

Most of systems do not restrict sets of users that may be assigned a task. This is given by the fact that resources in systems are not assigned security labels specifying sensitivity or nature of the resources' contents. This property is characteristic especially for smaller systems.

There are products that use role-based access control to enforce some control of data flow. But there are practical problem to stabilize privileges for roles because users are faced with limited access to resources that bounds their work. The problem is in combination of RBAC and DAC models and common feeling of security by administrators, of course.

The existing state of security, as described above, is natural because workflow systems are *generated* by needs of potential customers and there is no time and resources to perform some prerequisite research showing directions applicable for ten or more years to the future. Systems are designed ad hoc, according to basic programmer's rule that sounds like: Start with the most simple version possible and after it runs, begin to add whatever is needed.

And what about standardization effort? There has been already mentioned Workflow Management Coalition (WfMC) that is working towards XML as a medium for workflow systems. Very important seems to be Simple Workflow Access Protocol (SWAP) introduced as a draft by IETF. This document introduces protocol, based on HTTP1.1, that allows invocation and control of long-term processes. Basic goals are good, SWAP shall not invent anything that is already standardized somewhere else and should be as simple as possible. Let us take a look at security. About forty lines is denoted to the subject. The problem is that it talks only about security of data transmission. Who may send what, where and to whom depends on applications. There are defined some properties of documents that could be used for some more sophisticated security controls, but how to build them is not specified at all.

Another initiative comes from Object Management Group (OMG) that has developed its own standard for workflows called jFLOW. The group OMG BODTF developed a standard inspired by work of WfMC. It specifies that workflow management systems implicitly allow ad-hoc specifications of *executors* and this may be restricted. There are no more or less abstract guidelines for security policies, there. It means that security of workflow systems is fully in applications' hands and therefore questionable.

I can see another rising problem. There is a strong movement to agent-based applications. In such application, it is not clear what is going to be performed in advance. The first thought that occurs to me is that big problems with starvation and dead-locks, resulting from application of some security model, become even more important. What I am talking about is the fact that there are missing answers for workflow systems with some static definitions and the technology moves towards even more complicated systems.

Result from all the words in this section is short, but discouraging. Security of workflow systems is solved mostly poorly. Main reasons lie in fast evolution of the area and in complexity of stated tasks.

1.4 Overview

This dissertation identifies specific requirements for incorporating security into distributed information systems and proposes basic security model that allows synchronization of workflow execution and authorization flow. There is proposed classification of security layers that allows easier treatment of security problems.

There is given a brief introduction into *process algebra* - CCS. CCS (Calculus of Communicating Processes) [45, 35] has been designed for modeling communicating processes. CCS is then enriched with the proposed security model that allows formal specification of secure processes and their modeling and verification.

The remaining text of the thesis is divided into eight chapters. In the next chapter (Chapter 2) a short survey of security in information and database systems is given. Chapter 3 introduces basic notions of Calculus of Communicating Processes (CCS). This calculus is used to formalize security of distributed systems.

Rest of the thesis is the author's contribution to the field of security in strongly distributed systems. The theory is based on CCS and ideas of inheritance of processes (Chapter 8). The CCS theory and its description is strongly used in Chapter 7, where the original calculus is modified to satisfy security requirements. Chapter 4 describes general architecture of authorization system for strongly distributed information systems. The architecture is the cornerstone of the subsequent construction. The basic structure of the authorization system is defined in Section 4.1 and basic communication and cooperation among layers is introduced in Section 4.2.

Chapter 5 introduces Active authorization model (AAM). This model is the basis for authorization system. The original idea has been to use this model as is, but it showed up that a formalism for tasks' definitions is needed. That is reason, why e.g. relations of authorization steps are defined. Basic concepts of distributed systems are given in Section 5.1. The following two Sections 5.2 and 5.3 define the model itself.

The next chapter (Incorporating Security into CCS) is the first chapter that addresses security in CCS. The chapter consists from three sections. Section 6.1 makes minor changes in AAM necessary for successful incorporation into CCS. It introduces definitions of access and flow control, specifies their manipulation and defines classification/categorization of system elements. Section 6.1.3 introduces forms of authorization information for states and defines operator that enables *joining* of states. The last section defines properties of secure distributed system and secure tasks (and workflows).

Chapter 7 goes through specification of CCS and makes appropriate changes to ensure security requirements stated in the previous chapters. The aim was to do as little changes of CCS as possible. The resulting *Secure CCS* ensures properties of secure distributed system while preserving simplicity and useful properties of the original specification. The most important are notions of bisimilarity that had to be changed, but the original ideas stay preserved.

Chapter 8 incorporates ideas of Van den Aalst and Basten about inheritance of processes into the frame developed in the previous text. The thesis concludes with results of the work and possible directions for future research.

Chapter 2

Background

This chapter introduces basic classification of existing access control models. We start with the two most known types of models - discretionary and mandatory. Those are followed by role-based access control. We shall remember that the first two classes of models were introduced in 70-ies and they are still the most often used. Ideas of those models are used for (not only) military classifications of computer systems [2, 40]. But we should mention motivation that lead to the design of access control models before we move to the models themselves.

2.1 Security Policies

We have mentioned security requirements and various security controls in the previous chapter. We know what they mean, but we do not know which are important and which are not. Which should be implemented in the particular information system and which should be not. Answers for those questions shall be defined in *security policies*.

Security policies are abstract (high-level) guidelines concerning design and management of security. It means that security policies also describe authorization systems [36] and express basic choices for system's security. Security policies define *principles* and rules upon which data access is granted or denied and how the rules are administered. The rules that are used for such decisions are called authorization rules and determine the system behavior.

There are three basic types of administration in database systems and information systems generally. They are *hierarchical decentralized authorization* where a central authorizer is responsible for distributing administrative responsibilities, *ownership* with administration of objects by their creators and *cooperative authorization* when a predefined group of subjects must cooperate for administration of objects. Those types are however defined only for centralized systems and are not sufficient for distributed systems those nodes are, in high degree, autonomous - they assume homogeneous environment and one set of security policies.

Security of information systems may be based on one of two basic policy models proposed back in 70-ies: *mandatory access control* (MAC) or *discretionary access control* (DAC). The object-oriented technology has brought new policy model - *role-based access control* (RBAC) that is not so strict as MAC. It is more flexible and therefore more suitable for commercial environments.

In the moment we define the security rules, we have to choose mechanisms able to enforce our security requirements. It is not point to make a complete list of security mechanisms, we mention only two basic categories - external and internal security mechanisms. External security mechanisms are outside the system. They consist of administrative and physical controls and their task is to secure system from nature disasters and from unauthorized access of humans. Internal security mechanisms, on the other side, assume that humans working with the system are authorized to access the system and behave according to administrative rules. First task of internal mechanisms is to perform authentication of users and according to the results enforce authorization rules for data access.

2.2 Access Control Models

This section is about basic features of three types of access control (authorization) models. It is not the aim to feature exact specifications of models, but to introduce principles of most often used model families.

2.2.1 Discretionary Access Control

Models implementing discretionary access control govern access of users information on the basis of the user's identity. There are defined rules specifying, for each user and each object in the system, types of accesses a user is allowed on an object. The models are very flexible and therefore suitable for various types of systems and applications for which security is not the prime target (comparison of commercial and military systems [24]). Each user that creates an object administers access to the object by its own. That is the ground of flexibility, if you want to grant access right to your friend you are able to do it immediately without any 'outer' help. But also without any control!

Discretionary access control policies (and models that implement them) do not provide a real assurance on satisfaction of defined protection requirements. Although each access is controlled and allowed only if authorized, it is easily possible to bypass stated access restrictions. For example, a user who is permitted to read data can pass it to other users who are not authorized to read it without the cognizance of the data's owner. The main problem is that discretionary policies do not impose any restriction on the usage of information once obtained by a user. Dissemination of information is not controlled! This property allows attacks such as Trojan Horses.

To understand Trojan Horse attacks, let us consider the following example. Suppose that user x creates file f_1 and writes some information in it. Suppose, further, that user y creates file f_2. User y is owner of the file and he is therefore allowed to execute any operation on the file. In particular, y can grant other users authorizations on the file, so user x may receive (without knowledge) write privilege in file f_2. Consider now a program P (a Trojan Horse) which performs some utility and contains a hidden piece of code composed of a read operation on file f_1 and a write operation on file f_2. When user x executes P, the program may read file f_1 and may write it to file f_2. The illegal transmission has been performed.

Is there a solution to the problem? - Yes it is. The solution lies in the control of data flow. The first definition of a model that uses the notion has been proposed in 1976 [16] and theory is generally called *mandatory access control*.

2.2.2 Mandatory Access Control

Mandatory access control and flow control allow one to track flow of information and therefore provide some kind of protection from Trojan Horse attacks. The necessary condition for such controls is possibility to classify all objects and subjects (users) in the information system.

Such classification allows definition of rules that are used to control data flow by *reference monitor* that enforces mandatory policy (its axioms and rules). At the first sight it seems well, but it has been showed that mandatory access control policies have the drawback of being too rigid and therefore unapplicable to some environments from several reasons.

- It is not possible to assign clearances to users of commercial information systems.
- It is very often necessary to manually solve competent asks for data transfer.

- It is impossible to model non-monotonous hierarchies of access rights. (It is not possible to loose a privilege when moving up in the hierarchy.)

To fill the existing gap, variants of DAC models have been being proposed up to 1993. Propositions tried to strengthen discretionary access control models to ensure secure state. Some work has been aimed at eliminating or at least limiting the vulnerability of discretionary control to Trojan Horses [71, 21, 42, 61]. The approaches were based on need-to-know policy or by limiting objects (files) accessible by application programs on the basis of some knowledge about programs (active entities). Bertino [20] proposed the use of a strict need-to-know policy for overcoming the vulnerability of discretionary control policies in object-oriented systems. For access control in object-oriented systems see also [47, 50].

We may say that MAC extends DAC models with new properties - axioms [49]. When using mandatory policy, the access control usually combines mandatory access control and discretionary access control. This combination allows users to grant permissions to objects they own and on the other side, there is possible to state general rules for a system that limit possibility of unauthorized data flow. The assessment of elements in the system that is basis for such rules consists from two parts:

- *Classification* - It is a set composed of usually (military classification) four elements (Top Secret, Secret, Confidential and Unclassified). This set is fully ordered $TS > S > C > U$. Each element in the system is assigned one element of the set.

- *Set of categories* - It is a non-hierarchical set of elements. Elements of this set depend on the considered environment and refer to the given application area. Each element in the system is assigned a subset of the original set.

We obtain a set of security levels, whose elements form a *lattice* which is partially ordered ($\leq$). Each object receives its security level that does not change during its life-time. And each subject has got security clearance that represents maximum set of privileges.

The last fact that is worth noticing is primitivity of accesses that are granted. They are usually *read*, *write*, *append*, *execute*. Suppose an example of a bank when a clerk needs a privilege to change sum on an account. We have to give him a privilege to read the old sum and a privilege to write the new sum. But we are only able to give him rights *read* and *write*. You can see that the clerk is able to write any sum he wishes.

This changes with object-oriented systems that allow to grant privileges to execute methods of objects e.g. add sum to an account. We move from four privileges specified above to theoretically unlimited set of privileges to execute particular, well designed methods. We just need to check correctness of methods' definitions.

All access control models, as DAC as MAC, always represent just one security policy. Assuming, you administer information systems of a developing organization, then change of security policies makes necessary to implement new access control model.

The following subsection presents another access paradigm that appeared in the first half of 90-ies. It is the first family of models that do not contain, but express security policies.

2.2.3 Role-based Access Control

We have mentioned problems with using mandatory access control in some environments. This resulted in specification of a new type of access control *Role-based Access Control* (RBAC) [18, 63, 37, 57, 14, 43, 30, 58, 59, 19].

The basic idea is very simple. Access privileges are administered centrally, it means that e.g. an enterprise is able to fully control access to information resources according to its security policies. This administration is performed through *roles*. Privileges are granted to roles. Roles form a lattice that allows inheritance (we may define several types of inheritance) of access privileges. It makes administration even easier. Particular user obtains access privileges with assignment to a role. The approach is very intuitive. Roles may be created according to organizational structure of the corporation and granting and revoking of access to corporal resources for an employee can be done by one assignment or its cancellation.

It shows up that such an abstraction is very important for administration of large amount of users and resources. When using other access models then e.g. removing access privileges from a fired employee in the large, heterogeneous environment is generally complex task and it is very dangerous potential security hole.

Administration of access privileges is not enforced by common users and RBAC is therefore called non-discretionary access control model. RBAC supports some important policies for unclassified, but sensitive information. They allow determination of competence for execution of particular tasks, enforce principle of lowest privilege for administrators and specify conflicts

It has been shown that RBAC access control is able to express different variants of access models based on lattice hierarchy used for classification of system entities (in fact, lattice-based access control is a subset of RBAC [57, 56]).

RBAC access control is able to restrict data flow to a one-way flow. It is analogous to simple security property and star property from BLP model [16].

One may say that RBAC access control is neutral in respect to security policy; it may express particular policy, but it does not contain it. If we define a classification of access control, considering their generality as follows:

1. Trust objectives - basic aims that the system shall satisfy.
2. Requirements for extern interface - security requirements for the interface system-environment.
3. Internal requirements - requirements that shall be satisfied in the system components.
4. Operational rules - describe the way to ensure inner requirements.
5. Functional design - functional description of behavior of system components.

Then MAC and DAC access control represent item (3), while RBAC access controls are somewhere above that - it is much more abstract.

2.3 Related Research

Considerable studies in workflow management have been done by researchers from various perspectives in recent years [1]. Inter-task dependencies, atomicity requirements [9], properties of the entities while executing a task and their impact on execution [41] are studied. On the other side, there seems to be a strong deficiency in general formal model for workflows and for specification of tasks with long time-persistency. There are, of course, researchers that try to solve this problem (Van den Aalst should be certainly mentioned). One of the theoretical grounds for such specification seems to be Petri Nets [48, 52, 46].

Security of distributed systems and of execution of their tasks seems to be overlooked. It is natural, from some point of view, that we need a functional system and then we start carry of *miscellaneous* properties. I know about two sources that try to solve somehow, the problem of security. The first one comes from Thomas [66, 68, 67, 64, 65, 69], the second is concerned with Wei Kuang Huang and Vijay Atluri [39, 11, 10].

Both directions seem to be insufficient from some reasons. The works of Thomas ended in the informal definition of *Task-Based Authorization Control* (TBAC) with no further continuation. Some ideas are very interesting, but it has showed up that some concepts are unrealistic and some properties of TBAC are uselessly complicated. Wei Kuang Huang et al. went much further. They have proposed simple formal model for managing authorizations in distributed systems and applied it on the formal model of Petri-nets.

But we can see some problems with approach of Wei Kuang Huang. He applies mandatory and discretionary controls directly to his model and does not differentiate between local systems and global system that would ensure communication throughout the distributed system of isolated nodes. This model can not be therefore applied in heterogeneous information systems. Further, his conception of mandatory access control is rather different from standard view on the notion and problems with execution of tasks are solved by changes in the tasks' definitions. A set of axioms based on dependencies between tasks is built up to rearrange workflows.

We are going to introduce Active authorization model (AAM). Next step is to use theoretic model of communicating processes designed by Robin Milner [45] called *Calculus of Communicating Systems* (CCS). He shows conformity of his model with Petri-nets, but offers something totally new - bisimilarity - that is equivalence relation over processes. We want to enrich CCS with security properties of AAM (incorporate AAM into CCS). The result will be a formal model for securely communicating processes.

Chapter 3

Overview of CCS

Distributed systems that are to support long tasks that are executed by a number of agents lack theoretic foundation that would allow definition of their formal model. Existing workflow management systems are mostly created ad hoc, without any formal specification. This situation does not allow validation and verification of not only the system, but also of defined tasks. This situation is very dangerous and we have to restrict potential of the system or reconcile with possible delays and deadlocks during execution of workflows.

There is a strong movement towards use of Petri Nets [10, 4] for modeling workflow systems. But I was not satisfied after study of the literature (i.e. [70, 5]). Petri Nets seem not to cover real-world problems, there were proposed Higher-order Object Nets [44], but they lack formal model of basic P/T nets. At last I have found process algebras and *Calculus of Communicating Systems* (CCS)[45, 35]. This calculus has been proposed for study of communication and concurrency among processes. The basic idea has been to propose a usable definition of equivalence for processes.

This chapter should be only a short introduction into CCS because we are going to redefine it and it will take much more place later (see Chapter 7). That is why we also omit most of proofs.

According to [45], I am going to use for CCS the term *process calculus*.

3.1 Basic Definitions

This section describes basics of process calculus as defined in [45]. I try to shorten it as much as possible and propose only notions that are elementary for the calculus and notions that are to be extended.

First of all, we should mention agents and ports. CCS distinguishes states that perform actions that are invisible from a given point of view. Those states are called agents or more generally agent expressions. Each time an agent makes a visible action (starts a communication) it is performed through a port that determines properties of communication

3.1.1 Action and Transition

We assume an infinite set $\mathcal{A}$ of names, and use $a, b, c, \ldots \in \mathcal{A}$ (input ports). We denote $\bar{\mathcal{A}}$ the set of co-names $\bar{a}, \bar{b}, \bar{c}, \ldots \in \bar{\mathcal{A}}$ and $\mathcal{L} = \mathcal{A} \cup \bar{\mathcal{A}}$ is the set of *labels* that label ports (places that agents communicate through) and we shall use ℓ, ℓ' to range over $\mathcal{L}$.

To write simultaneous changes of states, there is used special perfect action τ. This action is only the one because it has no visible effects and we recognize no property of it. Henceforward we shall let $Act = \mathcal{L} \cup \{\tau\}$ be the set of all actions in the system and $\alpha, \beta, \ldots \in Act$.

The τ action is very important because it allows to ignore internal (perfect) actions of the analyzed system. We wish to regard two systems as equivalent if they exhibit the same pattern of *external* actions. This needs some kind of abstraction that is done with replacing unimportant actions by τ action. It turns out that a sequence

$$P_1 \xrightarrow{\tau} P_2 \xrightarrow{\tau} \ldots \xrightarrow{\tau} P_n$$

of internal actions is equivalent to a single internal action

$$P_1 \xrightarrow{\tau} P_n$$

3.1.2 Basic Language

Let us continue with definition of the calculus language. We shall use K, L to stand for subsets of $\mathcal{L}$ and $\bar{L}$ for the set of complements of labels in L. A *relabelling function* f is a function from $\mathcal{L}$ to $\mathcal{L}$ such that $f(\bar{\ell}) = \overline{f(\ell)}$ and also extend f to Act by decreeing that $f(\tau) = \tau$.

We introduce the set $\mathcal{X}$ ($X, Y, \ldots \in \mathcal{X}$) of *agent variables* and the set $\mathcal{K}$ ($A, B, \ldots \in \mathcal{K}$) of *agent constants*. Further, I and J will stand for indexing sets ($\{E_i : i \in I\}$). And at last we define $\mathcal{E}$, the set of *agent expressions* ($E, F, \ldots \in \mathcal{E}$). $\mathcal{E}$ is the smallest set that includes $\mathcal{X}$ and $\mathcal{K}$ and contains the following expressions, where $E, E_i \in \mathcal{E}$:

1. $\alpha.E$, a *Prefix* ($\alpha \in Act$)
2. $\sum_{i \in I} E_i$, a *Summation* (I an indexing set)

3. $E_1|E_2$, a *Composition*
4. $E\backslash L$, a *Restriction* ($L \subseteq \mathcal{L}$)
5. $E[f]$, a *Relabelling* (f a relabelling function)

We should say a few words about special cases of (2). We have used general notation, but usually we shall use only binary Summation $E_1 + E_2$. Another special case is when $I = 0$, the empty set. This gives us the inactive agent, capable of no action. We use the special name for it - $\mathbf{0}$. It is defined as

$$\mathbf{0} \stackrel{\text{def}}{=} \sum_{i \in 0} E_i$$

Further, we define decreasing binding power of the combinators in the following order: Restriction and Relabelling (tightest binding), Prefix, Composition, Summation.

$$R + a.P \mid b.Q\backslash L \qquad \text{means} \qquad R + ((a.P) \mid (b.(Q\backslash L)))$$

We write $Vars(E)$ for the set of variables occurring free in E and we say that an agent expression E is an *agent* if it contains no free variables. We denote the set of agents by $\mathcal{P}$ ($P, Q, \ldots \in \mathcal{P}$). A Constant is an agent whose meaning is given by a defining equation. In fact, we assume that for *every* Constant A there is a defining equation of the form

$$A \stackrel{\text{def}}{=} P$$

We often need to substitute expressions or variables. Let $\{X_i \mid i \in I\}$ or $\tilde{X}$ be an indexed family of variables, all distinct; then we write $\{E_i/X_i \mid i \in I\}$ or $\{E/X\}$ for the operation which simultaneously replaces all free occurrences of X_i by E_i for all $i \in I$. We shall always use $E_1 \equiv E_2$ to mean that two expressions are *syntactically identical*.

3.1.3 Transitional Semantics

The general notion of a *labeled transition system* is

$$(S, T, \{\xrightarrow{t} \mid t \in T\})$$

which consists of a set S of *states*, a set T of *transition labels*, and a *transition relation* $\xrightarrow{t} \subseteq S \times S$ for each $t \in T$. We shall take S to be $\mathcal{E}$, the agent expressions, and T to be Act, the actions. Our semantics for $\mathcal{E}$ consists of the definition of each transition relation $\xrightarrow{\alpha}$ over $\mathcal{E}$.

We now give the complete set of transition rules. The names **Act**, **Sum**, **Com**, **Res**, **Rel** and **Con** indicate that the rules are associated respectively with Prefix, Summation, Composition, Restriction, Relabelling and with Constants.

$$\textbf{Act}\ \frac{}{\alpha.E \xrightarrow{\alpha} E} \qquad \textbf{Sum}_j\ \frac{E_j \xrightarrow{\alpha} E'_j}{\sum_{i\in I} E_i \xrightarrow{\alpha} E'_j}(j \in I)$$

$$\textbf{Com}_1\ \frac{E \xrightarrow{\alpha} E'}{E|F \xrightarrow{\alpha} E'|F} \qquad \textbf{Com}_2\ \frac{F \xrightarrow{\alpha} F'}{E|F \xrightarrow{\alpha} E|F'}$$

$$\textbf{Com}_3 \frac{E \xrightarrow{\ell} E' \quad F \xrightarrow{\bar{\ell}} F'}{E|F \xrightarrow{\tau} E'|F'}$$

$$\textbf{Res}\ \frac{E \xrightarrow{\alpha} E'}{E\backslash L \xrightarrow{\alpha} E'\backslash L}(\alpha, \bar{\alpha} \notin L) \qquad \textbf{Rel}\ \frac{E \xrightarrow{\alpha} E'}{E[f] \xrightarrow{f(\alpha)} E'[f]}$$

$$\textbf{Con}\ \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'}(A \overset{\text{def}}{=} P)$$

The rule for Summation can be read as follows: if any one summand E_j of the sum $\sum_{i\in I}$ has an action, then the whole sum also has that action.

3.1.4 Derivatives and Derivation Trees

Whenever $E \xrightarrow{\alpha} E'$, we call the pair (α, E') an *immediate derivative of* E, we call α an *action of* E and we call E' an α*-derivative of* E.

Analogously, whenever $E \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} E'$ we call $(\alpha_1, \ldots, \alpha_n, E')$ a *derivative* of E, we call $\alpha_1, \ldots, \alpha_n$ an *action-sequence* of E, and we call E' an $\alpha_1, \ldots, \alpha_n$*-derivative* (or sometimes just a *derivative*) of E. In the case $n = 0$, we have that ε (empty action) is an action-sequence of E, and E is a derivative of itself.

It is convenient to collect the derivatives of an expression E into the *derivation tree* of E, where for each expression at a non-terminal node, all its immediate derivatives are represented by outgoing arcs. We call the tree *total* if expressions at terminal nodes have no immediate derivatives, otherwise *partial.*

3.1.5 Sorts

A *sort* is just a set of labels. We say that an agent P *has sort* L, if all the actions which P may perform at any time in the future have labels in L.

Definition 3.1. For any $L \subseteq \mathcal{L}$, if the actions of P and all its derivatives lie in $L \cup \{\tau\}$ then we say P *has sort* L, or L *is a sort of* P, and write $P : L$. ■

Now the following useful property can be very simply proved.

Proposition 3.1. *For every E and L, L is a sort of E if and only if, whenever $E \xrightarrow{\alpha} E'$, then*

1. $\alpha \in L \cup \{\tau\}$
2. *L is a sort of E'* ■

Clearly every agent has a minimum sort - the intersection of all its sorts - but this is not always easy to determine. However, there is a perfectly natural way to assign a sort $\mathcal{L}(E)$ to each agent expression E on the basis of its syntax. First, we assume that each variable X is assigned a sort $\mathcal{L}(X)$, and each Constant A is assigned a sort $\mathcal{L}(A)$.

Definition 3.2. Given the sorts $\mathcal{L}(A)$ and $\mathcal{L}(X)$ of Constants and variables, the *syntactic sort* (which is sometimes just called *the sort*) $\mathcal{L}(E)$ of each agent expression E is defined as follows:

$$\begin{aligned}
\mathcal{L}(\ell.E) &= \{\ell\} \cup \mathcal{L}(E)\\
\mathcal{L}(\tau.E) &= \mathcal{L}(E)\\
\mathcal{L}(\sum_i E_i) &= \bigcup_i \mathcal{L}(E_i)\\
\mathcal{L}(E|F) &= \mathcal{L}(E) \cup \mathcal{L}(F)\\
\mathcal{L}(E\backslash L) &= \mathcal{L}(E) - (L \cup \bar{L})\\
\mathcal{L}(E[f]) &= \{f(\ell) : \ell \in \mathcal{L}(E)\}
\end{aligned}$$

Further, for any defining equation $A \overset{\text{def}}{=} P$ the inclusion $\mathcal{L}(P) \subseteq \mathcal{L}(A)$ must hold. ■

3.2 Equational Laws

Equational laws of the calculus fall naturally into three groups. Each group represents one aspect of the behavioral equality of agents.

The first category consists of Composition, Restriction and Relabelling, which may be called *static combinators*. The rules of action for these combinators are all of the form

$$\frac{E_{i_1} \overset{\alpha_1}{\rightarrow} E'_{i_1} \quad \ldots \quad E_{i_m} \overset{\alpha_m}{\rightarrow} E_{i_m}}{comb(E_1, \ldots, E_n) \overset{\alpha}{\rightarrow} comb(E'_1, \ldots, E'_n)}$$

where $\{i_1, \ldots, i_m\}$ is a subset of $\{1, \ldots, n\}$, and E'_i is identical with E_i unless $i \in \{i_1, \ldots, i_m\}$. The significant point is that the combinator *comb* is present after the action as well as before, and moreover the only components which have changed are those whose actions have contributed to the action of the compound agent.

The second category consists of *dynamic combinators*: Prefix, Summation and Constants. In each rule for these combinators, an occurrence of the combinator is present before the action and absent afterwards.

We thus gain a three-way classification of equational laws, which we shall use in our presentation:

- The *static laws*, involving only the static combinators. These laws can be regarded as an algebra of flow graphs.
- The *dynamic laws*, involving only the dynamic combinators. These laws can be regarded as an algebra of transition graphs.
- The laws which relate one group to the other. As far as concerned, these laws may all be collected into a single one known as the *expansion law*.

3.2.1 The Dynamic Laws

The first set of laws concerns Summation alone:

Proposition 3.2. *monoid laws*

1. $P + Q = Q + P$
2. $P + (Q + R) = (P + Q) + R)$
3. $P + P = P$

4. $P + \mathbf{0} = P$ ■

It cannot be done many proofs, but if we write any of the laws as $E_1 = E_2$, then E_1 and E_2 in each case have exactly the same derivatives; that is

$$E_1 \xrightarrow{\alpha} E' \quad \text{iff} \quad E_2 \xrightarrow{\alpha} E'$$

This is strong enough to ensure that the laws are valid under *any* definition of equality based upon the structure of derivation trees.

The second set of laws concerns Prefix and focuses on the meaning of the silent action τ:

Proposition 3.3. *τ laws*

1. $\alpha.\tau.P = \alpha.P$

2. $P + \tau.P = \tau.P$

3. $\alpha.(P + \tau.Q) + \alpha.Q = \alpha.(P + \tau.Q)$ ■

It seems that some discussion on terms of derivation trees is in order. Let us talk about item (3), denoting the left- and right-hand sides by E_1 and E_2. Now E_1 and E_2 do not have exactly the same α-derivatives, since

$$E_1 \xrightarrow{\alpha} Q \quad \text{but not} \quad E_2 \xrightarrow{\alpha} Q$$

Because there is one *silent action.* This situation and previously stated tendency to ignore τ actions motivate the definition of a new transition relation.

Definition 3.3. $P \overset{\alpha}{\Rightarrow} P'$ if $P(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^* P'$. ■

In this definition, we use $(\xrightarrow{\tau})^*$, the transitive reflexive closure of $\xrightarrow{\tau}$, meaning 'zero or more τ actions'. Now, if we write (2) or (3) from the previous Proposition as $E_1 = E_2$, then for any α

$$E_1 \overset{\alpha}{\Rightarrow} E' \quad \text{iff} \quad E_2 \overset{\alpha}{\Rightarrow} E'$$

as may be checked by drawing derivation trees.

Corollary 3.4. $P + \tau.(P + Q) = \tau.(P + Q)$

Proof. $P + \tau.(P + Q) \overset{?}{=} \tau.(P + Q)$
$P + P + Q + \tau.(P + Q) \overset{?}{=} P + Q + \tau.(P + Q)$ (Proposition 3.3 (2))
$P + Q + \tau.(P + Q) = P + Q + \tau.(P + Q)$ (Proposition 3.2 (3))
it implies $P + \tau.(P + Q) = \tau.(P + Q)$ □

You may ask, why some attractive 'laws' has been rejected. First, consider the 'law' $\tau.P = P$. If we allowed it, we could deduce $a.P + \tau.b.Q = a.P + b.Q$. But we can feel the difference because via the τ action (in the left hand agent) we may move to a state, from which we are not able to perform action a. While the right hand agent does not offer this property.

Second, consider the distributive 'law' $\alpha.(P + Q) = \alpha.P + \alpha.Q$. If we allowed it, we could deduce $a.(b.P + c.Q) = a.b.P + a.c.Q$. But our transitional semantics allows $b.P$ as an a-alternative of the right-hand agent, and - assuming $b \neq c$ - this derivative is incapable of a c action, contrary to the left-hand agent. Thus a deadlock may occur for one agent, but not for the other.

At this place we overpass the set of recursion laws.

3.2.2 The Expansion Law

A concurrent system may be naturally expressed as a restricted Composition, i.e. in the form

$$(P_1|\ldots|P_n)\backslash L$$

Assume the following example. We have got two copies of C, with sort $\{in, \overline{out}\}$ and chain them together. For this purpose, we must relabel the ports in the middle, before composing and restricting; therefore the definition of the appropriate linking combinator $\frown$, applied to arbitrary agents P and Q, is:

$$P \frown Q \stackrel{\text{def}}{=} (P[\texttt{mid/out}]|Q[\texttt{mid/in}])\backslash\texttt{mid}$$

where mid is a fresh name (i.e. neither mid nor $\overline{\texttt{mid}}$ is in $\mathcal{L}(P)$ or $\mathcal{L}(Q)$).

In fact this form, a restricted Composition of relabelled components arises so often that we shall call it a *standard concurrent form* (scf); its general form is

$$(P_1[f_1] \mid \ldots \mid P_n[f_n])\backslash L$$

The expansion law is concerned with the immediate actions of an agent in standard concurrent form, as displayed schematically above.

Proposition 3.5. ***(The expansion law)*** *Let $P \equiv (P_1[f_1] \mid \ldots \mid P_n[f_n])\backslash L$, with $n \geq 1$. Then*

$$\begin{aligned} P = \sum \Big\{ & f_i(\alpha).(P_1[f_1]|\ldots|P'_i[f_i]|\ldots|P_n[f_n])\backslash L \mid \\ & \mid P_i \xrightarrow{\alpha} P'_i, f_i(\alpha) \notin L \cup \bar{L} \Big\} \\ + \sum \Big\{ & \tau.(P_1[f_1]|\ldots|P'_i[f_i]|\ldots|P'_j[f_j]|\ldots|P_n[f_n])\backslash L \mid \\ & \mid P_i \xrightarrow{\ell_1} P'_i, P_j \xrightarrow{\ell_2} P'_j, f_i(\ell_1) = \overline{f_j(\ell_2)}, i < j \Big\} \end{aligned}$$

■

A frequent use of expansion is when the relabelling functions f_i are all the identity function Id. We get a simplified form of the law.

Corollary 3.6. *Let $P \equiv (P_1|\ldots|P_n)\backslash L$, with $n \geq 1$. Then*

$$\begin{aligned} P = &\sum\left\{\alpha.(P_1|\ldots|P_i'|\ldots|P_n)\backslash L \mid P_i \xrightarrow{\alpha} P_i', \alpha \notin L \cup \bar{L}\right\} \\ &+ \sum\left\{\tau.(P_1|\ldots|P_i'|\ldots|P_j'|\ldots|P_n)\backslash L \mid \right. \\ &\qquad\qquad \left. \mid P_i \xrightarrow{\ell_1} P_i', P_j \xrightarrow{\ell_2} P_j', i < j\right\} \end{aligned}$$

■

The case $n = 1$ of the expansion law has special interest. The law specializes to some important laws relating Prefix with Restriction and Relabelling.

Corollary 3.7. *Specialities of the expansion law*

1. $(\alpha.Q)\backslash L = \begin{cases} \mathbf{0} & \text{if} \quad \alpha \in L \cup \bar{L} \\ \alpha.Q\backslash L & \text{otherwise} \end{cases}$
2. $(\alpha.Q)[f] = f(\alpha).Q[f]$
3. $(Q + R)\backslash L = Q\backslash L + R\backslash L$
4. $(Q + R)[f] = Q[f] + R[f]$ ■

3.2.3 The Static Laws

We claimed at the beginning of this chapter that the static laws, relating Composition, Restriction and Relabelling, can be regarded as an algebra of flow graphs. We begin this section by making more precise, what we mean by a flow graph.

We define a *flow graph* to be a set of nodes, where some pairs of ports are joined by *arcs* and some ports are assigned (outer) labels under the following conditions:

1. If two ports have outer labels ℓ and $\bar{\ell}$ then they are joined.
2. If two ports are joined and one has an outer label ℓ, then the other has outer label $\bar{\ell}$.

It is almost obvious how to define the operations of Composition, Restriction and Relabelling upon flow graphs. Let G and G' be arbitrary flow graphs. Then

- $G|G'$ is formed by joining every pair of ports - one in G and one in G' - which have complementary outer labels.
- $G\backslash L$ is formed by erasing outer labels ℓ and $\bar{\ell}$ from G, for each $\ell \in L$.
- $G[f]$ is formed by applying the relabelling function f to all outer labels in G.

The sort of $\mathcal{L}(G)$ of G is just its set of outer labels and the above rules ensure that

$$\begin{aligned}
&\mathcal{L}(G|G') = \mathcal{L}(G) \cup \mathcal{L}(G')\\
&\mathcal{L}(G\backslash L) = \mathcal{L}(G) - (L \cup \bar{L})\\
&\mathcal{L}(G[f]) = f(\mathcal{L}(G))
\end{aligned}$$

There are many different expressions for a single flow graph. The ideal situation would be

- to have a set of equational axioms for the static combinators which are sound and complete for flow graphs; that is, axioms from which can be deduced exactly those equations which are true in the flow graph interpretation.
- to know that every equation which is true in the flow graph interpretation is also true in the process interpretation.

We present the equations in three groups.

Proposition 3.8. ***(Composition laws)***

1. $P|\,Q = Q|P$
2. $P|(Q|R) = (P|Q)|R$
3. $P|\,\mathbf{0} = P$ ■

Note that $\mathbf{0}$ (in the flow graph interpretation) stands for the empty flow graph!

Proposition 3.9. ***(Restriction laws)***

1. $P\backslash L = P \qquad \textit{if} \quad \mathcal{L}(P) \cap (L \cup \bar{L}) = \emptyset$
2. $P\backslash K\backslash L = P\backslash(K \cup L)$
3. $P[f]\backslash L = P\backslash f^{-1}(L)[f]$
4. $(P|Q)\backslash L = P\backslash L \,|\, Q\backslash L \qquad \textit{if} \quad \mathcal{L}(P) \cap \overline{\mathcal{L}(Q)} \cap (L \cup \bar{L}) = \emptyset$ ■

The side-condition on (1) ensures that the labels L, and their complements, do not appear as labels in the flow graph P, so the Restriction $\backslash L$ is vacuous. (3) asserts that Restriction and Relabelling commute. (4) is a kind of distributive law; it says that Restriction distributes over Composition. (Communication between P and Q must not be restricted.)

Proposition 3.10. ***(Relabelling laws)***

1. $P[Id] = P$
2. $P[f] = P[f'] \qquad \textit{if} \quad f \restriction \mathcal{L}(P) = f' \restriction \mathcal{L}(P)$
3. $P[f][f'] = P[f' \circ f]$
4. $(P|Q)[f] = P[f] \,|\, Q[f]$ *if* $f \restriction (L \cup \bar{L})$ *is one-to-one,* $L = \mathcal{L}(P|Q)$. ■

In (1), Id is the identity function. The side-condition on (2) ensures that f and f' have like effect upon P (the notation $f \restriction D$ means the function f restricted to domain D). In (4) the side-condition is needed to ensure that when $[f]$ is applied to P and Q separately, it does not create more complementary port-pairs than existed previously.

3.3 Strong Bisimulation and Strong Equivalence

In this section, we set up a notion of equivalence between agents, based intuitively upon the idea that we only wish to distinguish between two agents P and Q if the distinction can be detected by an external agent interacting with each of them. For the entire section we shall treat τ (internal action) exactly like any other action. This will yield a rather strict equivalence relation which we shall call *strong equivalence*, in which we even distinguish between $a.\tau.\mathbf{0}$ and $a.\mathbf{0}$.

3.3.1 Strong Bisimulation

For the purpose of comparing processes, we consider an equivalence relation with the following property:

> *P and Q are equivalent iff, for every action α, every α-derivative of P is equivalent to some α-derivative of Q, and conversely.*

We can write this formally as follows, using $\sim$ for our equivalence relation.

$P \sim Q$ iff, for all $\alpha \in Act$, (*)

1. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$ and $P' \sim Q'$
2. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $P' \sim Q'$

What we are looking for, is the *largest* (or most generous) relation $\sim$ which satisfies the property (*).

Definition 3.4. (Strong bisimulation) A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a *strong bisimulation* if $(P, Q) \in \mathcal{S}$ implies for all $\alpha \in Act$,

1. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathcal{S}$
2. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \xrightarrow{\alpha} P'$ and $(P', Q') \in \mathcal{S}$ ■

3.3.2 Basic Properties of Strong Bisimulation

The first property which is enjoyed by the notion of strong bisimulation is that it is preserved by various operations on relations. In general, we define the *converse* $\mathcal{R}^{-1}$ of a binary relation and the *composition* $\mathcal{R}_1\mathcal{R}_2$ of two binary relations by

$$\begin{aligned} \mathcal{R}^{-1} &= \{(y, x) \mid (x, y) \in \mathcal{R}\} \\ \mathcal{R}_1\mathcal{R}_2 &= \{(x, z) \mid \text{for some } y, (x, y) \in \mathcal{R}_1 \text{ and } (y, z) \in \mathcal{R}_2\} \end{aligned}$$

Proposition 3.11. *Assume that each $\mathcal{S}_i (i = 1, 2, \ldots)$ is a strong bisimulation. Then the following relations are all strong bisimulations:*

(1) Id_p (3) $\mathcal{S}_1\mathcal{S}_2$

(2) $\mathcal{S}_i^{-1}$ (4) $\bigcup_{i \in I} \mathcal{S}_i$

■

Definition 3.5. P and Q are *strongly equivalent* or *strongly bisimilar*, written $P \sim Q$, if $(P, Q) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. This may be equivalently expressed as follows:

$$\sim = \bigcup \{\mathcal{S} : \mathcal{S} \text{ is a strong bisimulation}\}$$

■

Proposition 3.12.

1. *$\sim$ is the largest strong bisimulation*
2. *$\sim$ is an equivalence relation* ■

We are now in a position to prove most of the equational laws which were introduced in section 3.2. We shall in fact prove them valid when '=' is interpreted as strong equivalence; they will therefore be valid also for the weaker notion of equality introduced in the next section. However, Proposition 3.3 (τ laws) is not valid for strong equivalence.

3.4 Bisimulation and Observation Equivalence

We have introduced the notion of strong bisimulation in Section 3.3. It assumes that every α action of one agent must be matched by an α action of the other one - even for τ actions. Here we relax the requirement and τ action must be matched by *zero or more* τ actions.

3.4.1 The Definition of Bisimulation

A few preliminary definitions are needed.

Definition 3.6. If $t \in \mathit{Act}^*$, then $\hat{t} \in \mathcal{L}^*$ is the sequence gained by deleting all occurrences of τ from t. ■

Note, in particular, that $\widehat{\tau^n} = \varepsilon$ (the empty sequence).

Definition 3.7. If $t = \alpha_1 \ldots \alpha_n \in \mathit{Act}^*$, then we write $E \xrightarrow{t}$ if $E \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} E'$. We shall also write $E \xrightarrow{t}$ to mean that $E \xrightarrow{t} E'$ for some E'. ■

We now define a labelled transition system

$$(\mathcal{E}, \mathcal{L}^*, \{\overset{s}{\Rightarrow} \mid s \in \mathcal{L}^*\})$$

over agent expressions, in which the transition relations $\overset{s}{\Rightarrow}$ are defined as follows. For convenience we actually define $\overset{t}{\Rightarrow}$ for all $t \in \mathit{Act}^*$, i.e. for sequences which may contain τ.

Definition 3.8. If $t = \alpha_1 \ldots \alpha_n \in \mathit{Act}^*$, then $E \overset{t}{\Rightarrow} E'$ if

$$E(\xrightarrow{\tau})^* \overset{\alpha_1}{\rightarrow} (\xrightarrow{\tau})^* \cdots (\xrightarrow{\tau})^* \overset{\alpha_1}{\rightarrow} (\xrightarrow{\tau})^* E'$$

We shall also write $E \overset{t}{\Rightarrow}$ to mean that $E \overset{t}{\Rightarrow} E'$ for some E'. ■

Thus $E \overset{ab}{\Rightarrow} E'$ means that $E \overset{\tau^p}{\rightarrow}\overset{a}{\rightarrow}\overset{\tau^q}{\rightarrow}\overset{b}{\rightarrow}\overset{\tau^r}{\rightarrow} E'$ for some $p, q, r \geq 0$. Note also that $E \overset{\varepsilon}{\Rightarrow} E'$ iff $E \overset{\tau^n}{\rightarrow} E'$ for some $n \geq 0$.

We now introduce a notion analogous to derivative.

Definition 3.9. If $t \in \mathit{Act}^*$, then E' is a *t-descendant* of E iff $E \overset{\hat{t}}{\Rightarrow} E'$. ■

Note that if $t \in \mathcal{L}^*$ this just means $E \overset{t}{\Rightarrow} E'$, since $t = \hat{t}$ in this case. But notice that E' is a τ-descendant of E iff $E \overset{\tau^n}{\rightarrow} E'$ for some $n \geq 0$, and this includes the case $n = 0$ in which $E' \equiv E$.

Remark 3.1. Let us summarize the difference between the three relations $\overset{t}{\rightarrow}$, $\overset{t}{\Rightarrow}$ and $\overset{\hat{t}}{\Rightarrow}$. All specify action-sequence with observable content t. What changes is intervening of τ actions:
$\overset{t}{\rightarrow}$ specifies exactly the τ actions;
$\overset{t}{\Rightarrow}$ specifies at least the τ action;
$\overset{\hat{t}}{\Rightarrow}$ specifies nothing about τ actions.

So bearing in mind what we said about matching any τ action by *zero or more* τ actions, we want a notion of equivalence - which we shall call *observation equivalence* - with the following property:

> P and Q are observation-equivalent iff, for every action α, every α-derivative of P is observation-equivalent to some α-descendant of Q, and similarly with P and Q interchanged.

Definition 3.10. (Weak bisimulation) A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over agents is a *(weak) bisimulation* if $(P, Q) \in \mathcal{S}$ implies, for all $\alpha \in \mathit{Act}$,

1. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \overset{\hat{\alpha}}{\Rightarrow} Q'$ and $(P', Q') \in \mathcal{S}$

2. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \overset{\hat{\alpha}}{\Rightarrow} P'$ and $(P', Q') \in \mathcal{S}$ ■

We can now define observation equivalence, or bisimilarity:

Definition 3.11. P and Q are *observation-equivalent* or *(weakly) bisimilar*, written $P \approx Q$, if $(P, Q) \in \mathcal{S}$ for some (weak) bisimulation $\mathcal{S}$. That is,

$$\approx \, = \bigcup \{\mathcal{S} : \mathcal{S} \quad \text{is a bisimulation}\}$$

■

3.4.2 Properties of Bisimilarity

We show a few simple properties of $\approx$ analogous with properties of $\sim$ on the following lines.

Proposition 3.13. *Assume that each $\mathcal{S}_i$ $(i = 1, 2, \ldots)$ is a bisimulation. Then the following are all bisimulations.*

(1)	Id_p	(3)	$\mathcal{S}_1\mathcal{S}_2$
(2)	$\mathcal{S}_i^{-1}$	(4)	$\bigcup_{i \in I} \mathcal{S}_i$

Proposition 3.14.

1. $\approx$ is the largest bisimulation.

2. $\approx$ is an equivalence relation. ■

We now turn to further properties of $\approx$. We start with the property which distinguishes $\approx$ from $\sim$ most sharply.

Proposition 3.15. $P \approx \tau.P$ ■

This is perhaps the source of all the power of bisimilarity. It allows τ to be ignored - to some extent - in investigating bisimilarity.

Proposition 3.16. *$P \sim Q$ implies $P = Q$, and $P = Q$ implies $P \approx Q$.* ■

Thus all our equational laws for $\sim$ and $=$ hold also for bisimilarity, though the opposite is not true. The next two propositions show how equality may be deduced from bisimilarity; they are crucial in applications.

Definition 3.12. P *is stable* if P has no τ-derivative. ■

Proposition 3.17. *If $P \approx Q$ and both P and Q are stable, then $P = Q$.* ■

Thus it appears that the difference between $\approx$ and $=$ is only a matter of the initial actions of agents. The next result underlines this fact.

Proposition 3.18. *If $P \approx Q$ then $\alpha.P = \alpha.Q$.* ■

3.5 The Theory of Observation Congruence

This is the last section of algebraic theory of CCS. We begin by introducing another way to define the notion of bisimulation. We then declare that bisimilarity is preserved by all the basic combinators, except Summation. At last, the long-awaited definition of equality, =, is given. In particular, it is shown that equality is indeed a congruence relation which justifies all the normal algebraic manipulations.

3.5.1 Experiments and Substitutivity

We begin by giving another characterization of bisimulation, in terms of the notion of *experiment*. We consider $P \overset{s}{\Rightarrow}$, where $s \in \mathcal{L}^*$ to be an experiment upon P; it consists of running P a 'little' and observing the action sequence $s = \ell_1 \ldots \ell_n$. If $s = \varepsilon$, we also write $P \Rightarrow P'$; if P is unstable then this may involve one or more (unobserved) τ actions. $P \Rightarrow P'$ is still an experiment and must play a part in our theory because P' may admit fewer experiments than P does.

Proposition 3.19. *$\mathcal{S}$ is a bisimulation if, for all $(P, Q) \in \mathcal{S}$ and $s \in \mathcal{L}^*$,*

1. *Whenever $P \overset{s}{\Rightarrow} P'$ then, for some Q', $Q \overset{s}{\Rightarrow} Q'$ and $P'\mathcal{S}Q'$*
2. *Whenever $Q \overset{s}{\Rightarrow} Q'$ then, for some P', $P \overset{s}{\Rightarrow} P'$ and $P'\mathcal{S}Q'$* ■

Proposition 3.20. *Bisimilarity is preserved by Composition, Restriction and Relabelling. It means that if $P \approx Q$ then $P|R \approx Q|R$, $P\backslash L \approx Q\backslash L$ and $P[S] \approx Q[S]$.* ■

3.5.2 Equality and its Basic Properties

We have seen that $\approx$ is not fully substitutive; $P \approx Q$ does not imply $P + R \approx Q + R$. We want a notion of equality, $P = Q$, which implies $P \approx Q$ and which is fully substitutive, we want a congruence relation. It should be the largest congruence relation included in $\approx$.

Definition 3.13. P and Q are *equal* or *(observation-) congruent*, written $P = Q$, if for all α

1. Whenever $P \xrightarrow{\alpha} P'$ then, for some Q', $Q \overset{\alpha}{\Rightarrow} Q'$ and $P' \approx Q'$
2. Whenever $Q \xrightarrow{\alpha} Q'$ then, for some P', $P \overset{\alpha}{\Rightarrow} P'$ and $P' \approx Q'$ ■

Note that the two clauses differ from those for $\approx$ only in one respect: $\overset{\alpha}{\Rightarrow}$ appears in place of $\overset{\hat{\alpha}}{\Rightarrow}$. Thus each action of P or Q must be matched by at least one action of the other.

We also extend equality to expressions, in the same way as bisimilarity.

Definition 3.14. Let the variables which occur in E or in F be $\tilde{X}$. Then $E = F$ if, for all $\tilde{P}$, $E\{\tilde{P}/\tilde{X}\} = F\{\tilde{P}/\tilde{X}\}$. ■

We can immediately obtain a characterization of equality which shows how close it is to bisimilarity.

Proposition 3.21. *Assume that $\mathcal{L}(P) \cup \mathcal{L}(Q) \neq \mathcal{L}$. Then $P = Q$ iff, for all R, $P + R \approx Q + R$.* ■

Proposition 3.22. *$P \sim Q$ implies $P = Q$, and $P = Q$ implies $P \approx Q$.* ■

And we pick the last two propositions of equality properties.

Proposition 3.23. *Equality is an equivalence relation.*

Proposition 3.24. *If $P = Q$ then $\alpha.P = \alpha.Q$, $P + R = Q + R$, $P|R = Q|R$, $P\backslash L = Q\backslash L$ and $P[f] = Q[f]$.*

Chapter 4

Architecture of the Authorization System

I was considering the architecture for a pretty long time. After some prior study of widely deployed systems and their access control models [25, 27], I turned my mind toward the direction of Task-based Authorization System and problems raising from its possible application [26]. Results solving some particular problems were published during the last year of my internal PhD. study [28, 29]. This chapter presents integrated solution of a general architecture of authorization system.

The first issue that has to be solved is establishment of general architecture of the authorization system as a whole. We have not found any general abstract conception or model of authorization system for strongly distributed systems in the literature. But we feel that specification of general framework is necessary basic step before we start to define parts of the general problem separately.

We are interested in heterogeneous distributed systems that are composed from autonomous information systems able to communicate among themselves. We are going to call those autonomous systems *s-nodes* (from system nodes).

Each s-node contains resources it is able to work with (files, peripheral devices, . . .), a system that manages work with resources (operational system, database management system) and access (or authorization) control model that is used for determining allowed accesses to the resources.[1] We generally do not assume any uniformity among s-nodes, particularly considering access control systems. Any restrictions stated for the access systems shall be specified explicitly in the text.

[1]Chapter 5 introduces another definition of s-nodes. We mention here only parts important for the chapter and other issues are omitted.

4.1 Basic Structure

It is useful to define basic layers of the authorization system based on the architecture of distributed systems. We are not interested in structure of s-nodes, we just identify necessary layers that are needed for successful implementation of global security model in heterogeneous environment.

We have said already that s-nodes contain (or use) access control models. This is the first layer in our architectural framework and we shall call it *Local Access Control.* We are not able to say too much about general properties of those controls. We assume, they are capable to perform access control to local resources and they should be able to create new users or user groups. They are able to assign resources with properties usable for management of access control. We need them also to provide mechanisms that allow authentication of more abstract systems (or its users) and following access to resources.

There are two general approaches for users in abstract systems. A new group or general user that is able to access local resources is created. It means that other layers of the distributed system create new hierarchy of users and their authorizations. The second way is that existing structure of users is exploited. We prefer (and use in the following description) the first alternative, where all resources accessible for distributed system are accessible for a new user or group and access control, is partially moved to another layer of the architecture.

Definition 4.1. (Local Access Control) Local Access Control $\mathcal{L}_i^0$ is the n-tuple $\mathcal{L}_i^0 = \{S_i, O_i, A_i, f_i\}$ that contains the set of users S_i, the set of resources (or objects) O_i the set of access modes A_i and the authorization function f_i that is defined as follows:

$$f_i : S_i \times O_i \times A_i \to \{True, False\}$$

The function f_i is able to decide access demands. ■

Because there are no common properties among Local Access Controls, there is neither any common set of privileges. The abstraction of access privileges among Local Access Controls may vary. It is not a problem, if we find a way to specify privileges on the global level.

The second layer of our architecture should be able to convert global definition of privileges into a form applicable in the Local Access Control and conversely. This layer shall be called *Conversion Layer.* This is the first element that creates some general (global) framework. We do not know yet how to specify privileges on the global level, but there is a place where it may be done. This layer has to be implemented with s-nodes means because it transforms some global information into local and vice versa. We are not able to change the lower side of this layer, but definition of the higher interface is only in our control.

Remember that this layer has two tasks. It has to determine set of subjects that are allowed to execute computational tasks and it may be needed to determine objects, falling into given categories.

Definition 4.2. (Conversion Layer) Conversion Layer represents three functions $\mathcal{L}_i^1 = (\sigma_i^r,\ \sigma_i^u,\ \sigma_i^c)$ that take the set of users and the set of resources from the underlying Local Access Control $\mathcal{L}_i^0 = \{S_i, O_i, A_i, f_i\}$, the set of resource categories C and the set of subjects S_i^c from Workflow Manager $\mathcal{L}^3$ (Definition 4.4) and returns subset of subjects that are authorized to access specified resources.

$$\sigma_i^u : C \times S_i^c \to S_i^c$$

uses σ_i^r to translate resource categories C into the set of resources O_i

$$\sigma_i^r : C \to O_i$$

and function σ_i^c that determine subjects in Local Access Control from subjects in Workflow Manager

$$\sigma_i^c : S_i^c \to S_i$$

■

The aim is to enable to use function f_i to determine e.g. the final subset of subjects with given categorization c allowed to access resources with the same categorization: $f_i(\sigma_i^c \circ \sigma_i^u(c, S_i^c),\ \sigma_i^r(c),\ a)$.

We get to the layer that allows specification of tasks executable on the particular s-node. There are generally two types of tasks for the distributed system important for current description.

- Tasks that are executed on one s-node and by one user and perform some basic, atomic actions from the distributed system's point of view. We shall call those tasks *atomic tasks*. The description of those tasks is dependent on s-node.
- Tasks that may be executed on several s-nodes and/or by several users. Those tasks' execution steps are defined only by other tasks. We are going to call such tasks *workflows*. Specification of workflows is independent on s-nodes.

We are now talking about the layer allowing to specify atomic tasks. Definitions of atomic tasks are described by means of the s-node, there are used operations that are executable by the s-node. We shall call this layer *Atomic Task Manager*.

Definition 4.3. (Atomic Task Manager) Atomic Task Manager $\mathcal{L}_i^2$ allows definition of tasks that are executable on s-node $\mathcal{S}_i$ and contains pool of defined task-templates. ■

Another, higher layer - *Workflow Manager* - manages definitions of workflows and all other computational tasks that need cooperation of more s-nodes to be completed. This layer is completely s-node independent. It allows to be implemented as platform independent and to be executed on various systems. (The ideal seems to be WWW environment and Java-language. WWW environment solves basic problems: it allows easy communication of s-nodes, offers user-friendly environment and some basic security properties.)

Workflow Manager may be split into two parts, one concerning *static authorizations* of workflows (authorizations that are specified in the workflow creation) and one concerning *dynamic authorizations*. The dynamic authorizations are very important in the moment the workflow is able to change its state according to the processed data; there has to be an instrument that allows restriction of data flow in the distributed system.

Definition 4.4. (Workflow Manager) The Workflow Manager layer $\mathcal{L}^3$ divides into static part $\mathcal{L}_s^3$ and dynamic part $\mathcal{L}_d^3$ (according to authorizations they are able to solve). It also contains pool of task-templates defined in the distributed system or at least their specifications. ■

I suppose that we have identified all basic layers of the authorization system. The following picture shows their position in the architecture and their mutual communication.

4.2 Communication of the Layers

There is a difference between authorization systems in centralized and distributed information system. We may call them as active and passive systems.

- Passive - authorization system waits for an access requirement and then decides whether to grant a privilege for access or not.

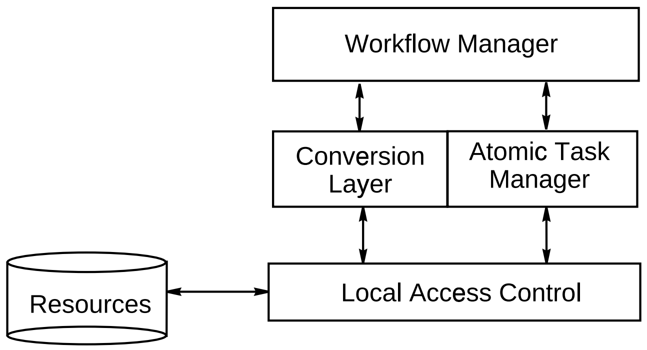

Figure 4.1: Communication among system layers

- Active - the system may be divided into two parts. First part creates set of possible subsequent access requirements and looks for s-nodes that are able to grant them. Second part is similar to passive systems.

Access control systems are implemented on centralized systems, so they are passive. Users ask for access and the system decides. Authorization system of any distributed system has to be active. The reason is fundamental. There is no central place, where all users of the distributed system are known.

S-nodes that have just completed a step of a task have to be able to determine, whether there are s-nodes with users that are allowed to continue the particular task. This is one of the crucial problems in cooperation between Local Access Control and Tasks (Workflows) Managers. This interface is also the most problematic for implementation.

How to perform a workflow? Let us imagine a that someone on s-node $\mathcal{S}_i$ has started a workflow and executed the first step (task) of it. We need to find all s-nodes that are potentially able to execute the next task of the workflow.

1. We have to address (all) s-nodes in the distributed system.

2. Each s-node identifies users that are able to continue the workflow and waits until any of the identified users initiates execution of the task.

3. When the *active* user is on the s-node $\mathcal{S}_j$, then $\mathcal{S}_j$ responds to the $\mathcal{S}_i$.

4. The s-node $\mathcal{S}_i$ recalls its request on all s-nodes except $\mathcal{S}_j$.

5. Execution of the workflow moves to the new s-node $\mathcal{S}_j$ and the *active* user may execute the task.

The most important is step (2). During this step, communication among layers of the authorization system on particular s-node is performed. The first impulse is received by the Workflow Manager. It has to ask the Atomic Task Manager if there is the particular task defined at all. In the case of success, the Atomic Task Manager has to determine (either directly with Local Access Control or in cooperation with Conversion Layer) set of users that possess necessary static authorizations to run the task. The same procedure, but assuming dynamic authorization, is guided by Workflow Manager, Conversion Layer and of course, Local Access Control. There are received two sets of users. Their intersection is the set of users authorized to execute next part (task step) of the workflow.

4.3 The Conversion Layer

We shall present a more detailed description of the Conversion Layer because it is the place, two considerably different models are in touch.

When thinking about possibilities that exist for general specification of authorization requirements, i.e. general property of subjects and resources, the following alternatives were identified.

1. Hierarchy in the organization.
2. Name of a subject (user).
3. Privilege for data access (analogy with MAC).
4. Reference to a common hierarchy (absolute or relative (from-to)).
5. Reference to another predefined role (group) structure.
6. Types of resources that have to be accessed.

The classification we shall use has to be very stable in time and must allow its application on all s-nodes. We have assumed that the most general and stable classification should be based on data (resources) that are accessed, on categorization of their content.[2]

Definition 4.5. (Resource categories) Resource categories is a non-hierarchical set of elements C that may depend on the environment and the application area. There are defined two set operations $\cup$ and $\subseteq$. ■

[2]Remember decimal classification of books. It is used for tens of years in libraries without need of re-classification of once classified books.

When using categories, we have no way to specify needed access modes. Therefore, when a subject is able to access particular resource, he is able to perform generally all possible operations over those resources. Restrictions in this direction are enforced through the functional tasks' definitions.

Resource categorization is the *fixed point* that allows global definition of workflows. All tasks' definitions use categorization (or classification) to specify security requirements for data access (access to resources and workflows). Categories are converted into form that the s-node's Local Access System is able to use to determine authorized users and to determine needed resources during execution of particular task step. This conversion is performed especially in Conversion Layer and partially by Atomic Task Manager.

We shall now define *Active authorization model* that constitutes basis of the authorization system for heterogeneous distributed systems.

Chapter 5

Active Authorization Model

This chapter contains description of Active authorization model (AAM) we are going to use further for specification of security in distributed systems. Section 5.1 proposes formalization of a distributed system that allows development of authorization model. Next section summarizes essential ideas of AAM and Section 5.3 brings full specification of Active authorization model.

5.1 Distributed System

Let us take a look on the context in which Active authorization model (AAM) is placed. Workflow management system, or more generally strongly distributed information system has been mentioned already several times and we should state its more precise definition.

Definition 5.1. (Strongly distributed system) Strongly distributed system $\mathcal{S}$ is a set of autonomous information systems (s-nodes) communicating with each other through communication channels $\chi_{i,j}$ that connect pairs of s-nodes ($\mathcal{S}_i$ and $\mathcal{S}_j$). $\mathcal{S} = (\{\mathcal{S}_{i \in I} \mid \mathcal{S}_i \text{ is an s-node}\}, \{\chi_{i,j \in I \times I} \mid i \neq j \wedge \chi_{i,j} \text{ is a channel}\})$. ■

Definition 5.2. (System node, s-node) S-node $\mathcal{S}_i$ is a 5-tuple $\mathcal{S}_i = (\mathcal{A}_i, \Omega_i, \mathcal{R}_i, \mathcal{U}_i, \mathcal{E}_i)$. S-node represents the administration unit $\mathcal{A}_i$, the authorization system Ω_i, the resource manager $\mathcal{R}_i$, the audit unit $\mathcal{U}_i$ and the system engine $\mathcal{E}_i$. ■

It is a complete view of s-node so there are even parts we are not interested in. $\mathcal{A}_i$ is the administration unit that enables to define and manage computational tasks and workflows. The authorization system Ω_i is responsible for authorizations and correct assignment of access rights. The resource manager $\mathcal{R}_i$ takes care of all resources in the distributed system and allows access to them. It should allow to assign classifications to resources, define subjects and again with their classifications. The audit unit $\mathcal{U}_i$ logs and analyzes all events in the given s-node $\mathcal{S}_i$ or even in the whole distributed system $\mathcal{S}$. The s-node engine $\mathcal{E}_i$ is the essential part that executes tasks prepared for processing on the s-node. Interface of the engine allows users to initiate new workflows and select tasks, they want to process.

The system engine $\mathcal{E}_i$ uses the remaining four parts to ensure secure, correct and immediate processing of tasks. It starts new tasks, ensures that necessary resources are allocated and proper access rights are granted. It frees used objects and prepares conditions for initialization of tasks, concurring the task termination.

S-node works with four basic types of entities: *resources*, *computational tasks*, *subjects* and *audit trails*. Resources can be divided into several basic types. We can talk about users, electronic information (files, objects, ...) and real resources (paper objects, physical devices and so on). We can also distinguish *atomic tasks*, *task steps* and *workflows* in computational tasks. We have not mentioned task steps yet, but the notion is necessary for the authorization model. Generally, task step consists from one or more atomic tasks and it is assigned security information that is atomic from user's point of view. Regarding subjects, there are users and tasks that may act as active entities.

What we are going to address is the authorization system and entities it works with. Other issues are mentioned only in the necessary volume.

We have already mentioned that computational tasks are assumed to be of two independent parts - functional and security.

Definition 5.3. (Task template) The computational task template (static definition of the task i) $\mathcal{T}_{i,0}$ is a couple consisting of the functional definition (Υ_i) itself and security related information (Ψ_i).

$$\mathcal{T}_{i,0} = (\Upsilon_i, \Psi_i)$$

■

Definition 5.4. (Task instance) The task instance $\mathcal{T}_{j,k}$ is a triplet

$$\mathcal{T}_{j,k} = (\mathcal{T}_{i,0}, \mathcal{P}_{j,k}, E_{j,k})$$

Where $\mathcal{T}_{i,0}$ is the appropriate task template, $\mathcal{P}_{j,k}$ is the actual protection state of the task instance and $E_{j,k}$ is the processing state. The indexes j and k of

$\mathcal{T}_{j,k}$ identify the running workflow (k) and actual branch (computational task) of the workflow (j). ■

Any task template is just a static definition of a task. It should be stored, in case of a workflow, on every s-node of the distributed system, or at least everywhere with at least one potential *executor* (subject that is able to initiate the task execution) of the task. Task templates of atomic tasks are stored on s-nodes, where they are defined. All s-nodes that are able to define new tasks and workflows should know specification of all existing atomic tasks' templates.

A task instance contains all information needed for its successful, correct and secure processing. There is no central place that can be used for the execution control. It means that every single task instance has to contain everything necessary for its processing.

The authorization system should be independent on implementation of distributed system, as on s-nodes, as on the way tasks are defined. To achieve this aim we have to stay on rather abstract level of modeling and develop simple, but very tiny connection between tasks and their authorization information. Other requirements encompass functional properties. The model should cover all necessary aspects of system's security:

- Synchronization - Time period of authorizations is synchronized with proper tasks.
- Authorization constrains - Capability to handle temporal constrains.
- Cooperation - Model has to be able to cooperate with DAC, MAC or RBAC models used for determination of needed resources and authorized subjects.
- Modeling - Modeling of authorizations in tasks and management of authorizations through tasks' life-cycles.
- Separation - Maintenance of separate protection states for each task.
- Control - Dynamic runtime check-in and check-out of permissions.

5.2 Basic Concepts of the Model

We want to specify basic concepts that are behind the idea of Active authorization model. We have already mentioned two of them - task and task template. We move to security aspects now and start from the beginning - task.

The basic notion in AAM is the *authorization unit*. Each workflow consists of well-defined tasks that perform simple actions - we shall call them task steps[1]. Authorization unit is able to describe all security information related to the given task step. It consists from four elements. The first element contains *initial privileges* that are used to determine the set of subjects (users or tasks that we are going to call initiators) that may start (initiate) the associated task. The second one is a set of *enabled privileges*. It means privileges that the executor of the task possesses. Usage of each privilege may be restricted by predicates. We use obligatory predicate defining maximum number of applications. Exact definition of privileges comes from the definition of authorization set - Def 5.8. The third element contains *executors* of the associated task step. The last data element defines number of authorization unit's *executions*.

The basic idea is to store all information needed for the task execution within its task instances (we are going to use the notion *flow unit*). This property is necessary for task migration. Task instances are transferred among s-nodes of the distributed system and no other information of them is known. In the moment the task is executed, additional security information has to be added to the 'functional' data needed for system engines $\mathcal{E}_i$. The structure containing those data is called *Protection data*. The content consists from all authorization units and some other information needed during the task and its authorization units' processing.

Protection data is a part of *flow unit* (it is representation of task instance $\mathcal{T}_{j,k}$) together with the data for task processing (it means task definition and processing state). The flow unit is complete 'bundle' of information needed for the correct and secure task processing.

The compact peace of information we have obtained in the flow unit may be migrated among s-nodes of the heterogeneous distributed system. On the other side, each authorization unit may be processed on the particular s-node independently. It defines protection state of the given task and its executor - it is crucial information for access control. If it contains identification of the corresponding task, it can also be separately audited (as well as stored or analyzed).

We are now to define basic notions of Active authorization model. This is another important step towards authorization system suitable for strongly distributed systems that has to be done.

[1]Task step is a computational task with one important property - it may be invoked by an extern subject. It means that we can define conditions for its initialization.

5.3 The Model Definition

This section is very compact and contains specification of Active authorization model. I want to mention that idea of the model has motivated formation of remaining parts of this thesis.

We assume following facts as implicit in the section. There is a set of resources (or objects, generally all passive elements in the system) O, a set of subjects (users and tasks) S and a set of all access rights A. Those three sets are finite. When the given set is used in context of an s-node, it may also have an index. Further suppose that $\mathbb{N}$ is the set of integer numbers. All definitions that follow are supposed to be stated in the context of a distributed system $\mathcal{S}$ or of s-nodes $\mathcal{S}_i$ it consists of. The difference shall be visible from indexes.

First of all, we have to define a few basic notions such as time mark, time interval, authorizations.

Definition 5.5. (Time marks - moments) The set of *time marks* $M_\mathcal{S}$ in the system $\mathcal{S}$, is defined as a set of nonnegative integer numbers. Let us define maximum value of a time mark $m_{max} > 0, m \in \mathbb{N}$.

$$M_\mathcal{S} = \{m | m \in \mathbb{N} \wedge m \in \langle 0, m_{max} \rangle\}$$

We may define time marks M_i for s-nodes $\mathcal{S}_i$ in the same way. ■

Definition 5.6. (Time intervals) If there is a set of time marks M then the set of *time intervals* in the context of M is defined as:

$$I_M \subseteq \{(m_1, m_2) \mid m_1, m_2 \in M \wedge m_1 < m_2\}$$

If there is a moment m such that $m \geq m_1 \wedge m \leq m_2$, we may write

$$m \in (m_1, m_2) \vee m \in I_M$$

and say that m is in interval I_M. Assuming s-nodes, we may denote relevant sets of intervals equivalently I_{M_i} or I_i. ■

Time marks do not express *real time moments*, but rather a sequence of events; conversion into *real time* differs with each task execution. There is one condition that should be preserved: order of actions in real time is the same as in time marks.

It is impossible to work with concrete resources in the authorization system because we do not know all of them and tasks have to be executable everywhere.

We therefore need some more abstract description of resources. The most suitable abstraction is resource separation into categories, according to their content (domain and sensitivity) and use of such a categorization. This is the reason we define notion of categorization of resources.

Definition 5.7. (Categorization) Assume that there is a non-hierarchical set of categories $C_{\mathcal{S}} = \{c_1, c_2, \ldots, c_n\}$. There are defined set operations $\cup$ and $\subseteq$ on $C_{\mathcal{S}}$.
There is also defined function $\Theta_{\mathcal{S}} : O \to C_{\mathcal{S}}$ that assigns each resource o_i in the system $\mathcal{S}$ an element $c_i \subseteq C_{\mathcal{S}}$. $\Theta_{\mathcal{S}}(o_i) = c_i$ is called categorization of the resource o_i.
Each s-node $\mathcal{S}_i$ has got defined function $\Theta_i \Leftrightarrow \Theta_{\mathcal{S}} \restriction O_i, O_i \subseteq O$. Values of Θ_i are from $C_i : C_i \subseteq C_{\mathcal{S}} \ \wedge \ \forall c \in C_i : \forall c_k \in C_{\mathcal{S}}$ such that $c \subseteq c_k,\ c_k \in C_i$. ■

A set of categories is defined for each s-node in the distributed system $\mathcal{S}$. Sets are chosen according to data the particular s-node is supposed to process. The set $C_{\mathcal{S}}$ should be created so generally that (ideally) no changes of once defined categories are needed during the life-cycle of the distributed system. On the other side, it is possible to easily add new elements[2] into $C_{\mathcal{S}}$.

The condition for subsets of categories in s-nodes is put for correct determination of authorized subjects. We may find a subject that is authorized for level higher then the one needed. We should have this level in set of classifications.

If the definition of function Θ were changed to $\Theta : O \to C \times \mathbb{N}$, we would obtain a classification function, as described in [16].

Definition 5.8. (Authorization set) Let us O_i be a set of all resources, S_i be a set of all subjects and A_i be a set of all access modes on an s-node. If there is defined $k \in \mathbb{N}$ then $U_i = \{u \mid u \in \mathbb{N} \vee u < k\}$ is a set of all possible repeating counts. Then the set of all authorizations P_i is defined as

$$P_i \subseteq S_i \times O_i \times A_i \times U_i$$

For that holds:

$$\forall\, p = (s, o, a, u) \in P_i : u > 0,\ \exists p' = (s, o, a, u - 1) \in P_i$$

■

[2]It implies existence of a mechanism for distribution of added categories among all s-nodes.

This is the first and the last time when subjects are directly associated with authorizations. The reason to define authorization sets is a need to express concrete privileges on s-nodes and to have a basis for deriving more abstract notions for authorizations. There is also practical reason - storing of information about authorizations applied to the system resources. We know that all users have to be authenticated in the system and that users are subjects of responsibility for actions in the system.

We are going to use association *task – privilege* from now on. This association is, from AAM point of view, implicit and is enforced by the task step containing corresponding *authorization template*. Authorization template is an authorization without specification of subject $(-, o, a, u)$. We need to define two types of authorization templates *abstract* that does not use objects, but categories of resources and *specific* that is used on a particular s-node - concrete resources may be used.

We need to create an association between security categories and subjects. We assume that when a subject is authorized to access one object with classification (or categorization) c, he is authorized to access all objects with classification c and resolution to particular resources is done according to task definitions. This axiom must be satisfied on all s-nodes.

Definition 5.9. (Abstract authorization template) Assume that P_i is an authorization set. We say that π^A is an abstract authorization template if the following condition holds.

$$\pi^A = (c, a, u) \text{ such that } \exists\ s \in S \wedge \exists\ o : \Theta_i(o) = c \text{ and } (s, o, a, u) \in P$$

Set of all abstract authorization templates in the distributed system $\mathcal{S}$ is denoted Π^A. ■

Definition 5.10. (Specific authorization template) Assume that P_i is an authorization set. We say that π^S is a specific authorization template if the following condition holds.

$$\pi^S = (o, a, u) \text{ such that } \exists\ s \in S :\ (s, o, a, u) \in P_i$$

Set of all specific authorization templates for s-node $\mathcal{S}_i$ is denoted Π_i^S. ■

Authorizations are applied when the appropriate objects are accessed. We define use of authorization to formalize this action.

Definition 5.11. (Authorization application) We define a function Λ that expresses application of an authorization. There are the same rules for authorizations, as well as for both types of authorization templates. Domains of the function are therefore P_i, Π^A and Π_i^S.

1. $\Lambda : P \mapsto P$,
   $p = (s, o, a, u) \in P \wedge u > 0$ then $\Lambda(s, o, a, u) = (s, o, a, u - 1)$
2. $\Lambda : \Pi^A \mapsto \Pi^A$
   $\pi^A = (c, a, u) \in \Pi^A \wedge u > 0$ then $\Lambda(c, a, u) = (c, a, u - 1)$
3. $\Lambda : \Pi^S \mapsto \Pi^S$
   $\pi^S = (o, a, u) \in \Pi^S \wedge u > 0$ then $\Lambda(o, a, u) = (o, a, u - 1)$ ■

We define initial authorizations to allow definition of conditions that subjects initializing tasks on the particular s-node have to satisfy.

Definition 5.12. (Initial authorizations) Let us assume that P_i is a set of authorizations and Π^A is the set of respective abstract authorization templates. Then let P_I^t be the set of initial authorizations (for the task step t).

$$P_I^t \subseteq \Pi^A$$

is the set of initial authorizations. ■

We should notice at least two facts. There is no constrain for sets of objects. It means that we may want a subject working on one s-node to have authorizations for objects on another s-node. The second fact is that there has to be at least one subject (in the system) that satisfies the initial authorizations. This condition is necessary for execution of the associated task.

All subjects that posses the initial authorizations are called initiators.

Definition 5.13. (Initiators) Let Π^A be the set of authorization templates, S_i be the set of subjects on s-node $\mathcal{S}_i$ and P_I^t be the set of initial authorizations chosen over Π^A and $S_0^{t,i} \subseteq S_i$. We define function Φ_i:

$$\Phi_i(S_i, P_I^t) \mapsto S_0^{t,i}$$

that returns a set of subjects $S_0^{t,i}$ on s-node $\mathcal{S}_i$ those privileges form superset to P_I^t. The set $S_0^{t,i}$ is then called *initiators* for P_I^t (on s-node $\mathcal{S}_i$). ■

The Φ_i function allows to combine Active authorization model with other access control models (DAC, MAC or RBAC) that are used for access control on particular s-nodes. We do not say how is the function implemented. We do not say what model is used for that purpose. It may be a function that finds all users that are specified in a given UNIX group or a function cooperating with RBAC model that uses much more sophisticated approaches for the purpose.

Definition 5.14. (Executor) Assume that task step t has got on s-node $\mathcal{S}_i$ set of initiators $S_0^{t,i}$. The subject $s \in S_0^{t,i}$ is called *executor of the task t* if it initiates the task step. We denote it as s_e^t. ■

Each task step that is invoked has its executor - the subject that processed the task step. Because we are working not directly with tasks, but with authorization information, we use *authorization unit* to express the notion. The *authorization unit* represents elementary set of access rights that may be granted to executor. It associates a set of initiators with sets of authorizations. Furthermore, it contains all information that is needed for successful processing and for finding successors of the unit.

Definition 5.15. (Authorization unit) We have the set of abstract authorization templates Π^A. Let P_I^t be the proper set of initial authorizations and $P_E^t \subseteq \Pi^A$ be the set of enabled authorizations. $S_e^t \subseteq S_0^t$ is the set of executors of the authorization unit:

$$S_e^t = \{s_e^t | \; \exists i, s_e^t \in S_i \wedge s_e^t \in \Phi_i(S_i, P_I^t)\}$$

Each task may be repeated several times, so we define number of initializations of the associated authorization unit $n \in \mathbb{N}$. The authorization unit ω_i is then defined as a 4-tuple:

$$\omega_i = (P_I^t, P_E^t, n, S_e^t)$$

The most important are the first three elements[3] of ω_i. ■

You see that authorization unit of a task consists from two sets of privileges that are derived from needs of the associated task (privileges needed for the task initiation and privileges enabled by that ω_i).

The original model contains definition of relations between authorization units. There are two possible ways to describe relations between tasks. It may be expressed in the authorization system and we may use a formalism for process description. Following chapters demonstrate former one by CCS, used as an apparatus for process description and its enriching with security properties. The following definition shows, how to define task relations in authorization model.

Definition 5.16. (Relations of authorization units) Let I, I_0 be two sets of time intervals in the distributed system $\mathcal{S}$ and $I \subset I_0$ is a set of time intervals

[3]We shall write authorization units as triples $\omega_i = (P_I^t, P_E^t, n)$ when not interested in issues of logging.

used by the workflow (represented by flow unit). Now choose any $i_1, i_2 \in I_0$. Further assume that ω_1 and ω_2 are authorization units of the workflow executed during time intervals $i_1 = (m_1^s, m_1^e)$ and $i_2 = (m_2^s, m_2^e)$ respectively.
We define the following relations between authorization units $\omega_1^{i_1}$ and $\omega_2^{i_2}$:

1. $\omega_1^{i_1} < \omega_2^{i_2} \overset{\text{def}}{\Longleftrightarrow} (i_1, i_2 \in I \wedge m_1^e < m_2^s)$
2. $\omega_1^{i_1} \# \omega_2^{i_2} \overset{\text{def}}{\Longleftrightarrow} (i_1 \in I \Rightarrow i_2 \notin I \wedge i_2 \in I \Rightarrow i_1 \notin I)$
3. $\omega_1^{i_1} \,\&\, \omega_2^{i_2} \overset{\text{def}}{\Longleftrightarrow} (i_1, i_2 \in I \Rightarrow m_1^s \in i_2 \vee m_2^s \in i_1)$

■

Initialization of an authorization unit (and execution of associated task step) decrements number of available instances of that authorization unit. When this number reaches zero, the authorization unit becomes invalid and can not be initialized any more inside the given workflow instance.

Definition 5.17. (Execution of authorization unit) Assume that there is given the authorization unit $\omega_i = (P_I^t, P_E^t, n, S_e^t)$ and Ω is the set of all authorization units in the distributed system $\mathcal{S}$. ω_i changes its state with each initialization. We denote k-th state of the authorization unit ω_i as $\omega_{i,k}$. The definition of unit's state is as follows:

1. The initial state ($k = 0$) is $\omega_{i,0} = (P_I^t, P_E^t, n_0, S_{e,0}^t)$, where $S_{e,0}^t = \emptyset$.
2. The k-th state ($k > 0$) is defined as $\omega_{i,k} = (P_I^t, P_E^t, n_k, S_{e,k}^t)$, where $S_{e,k}^t = S_{e,k-1}^t \cup s_e^t : s_e^t \in S_0^{t,i}$ (set of initiators on the actual s-node $\mathcal{S}_i$) and $n_k \geq 0 \wedge n_k = n_{k-1} - 1$.
3. The final state ($k = n_0$) is $\omega_{i,n} = (P_I^t, P_E^t, 0, S_{e,n}^t)$

We define function Δ that represents execution of the authorization unit:

$$\Delta : \Omega \mapsto \Omega \text{ and } \Delta(\omega_{i,k}) = \omega_{i,k+1}$$

according to the stated rules, for all $\omega_{i,k} \in \Omega \wedge k < n_0$. ■

Assuming execution of any authorization unit we are not interested in application of particular enabled privileges. This application is done according to Def 5.11. We shall denote authorization unit as $\omega_{i,k}^l$ after l-th application of enabled privileges.

Definition 5.18. (Validity of authorization unit) Assume, we have the authorization unit ω_i in the k-th state $\omega_{i,k} = (P_I^t, P_E^t, S_{e,k}^t, n_k)$. One says that ω_i is invalid if and only if $n_k = 0$. Otherwise ($n > 0$), the authorization unit is valid. ■

This is the skeleton of Active authorization model. The remaining definitions are stated to complete the construction of the model to the most abstract notions.

The following notion is the last one for us to be able to define *protection data*. It is a set that stores all applications of authorizations. This set may be archived for occasional control or security audit in the future.

Definition 5.19. (Authorization usage log) Authorization usage log is a set $\mathcal{U} = \{(p, id(\omega_i), m)\}$ that elements are triples consisting of applied authorizations from the relevant authorization set P ($p \in P$), identification of the authorization unit $id(\omega_i)$ and moment (from the set of time marks M) of the event (access to a resource) $m \in M$. ■

At last, we get to the definition of data structure that is able to store all information needed for authorization control during the execution of a workflow. There are information specifying sets of enabled privileges, information needed for synchronization of workflows and of course for auditing mechanisms.

Definition 5.20. (Protection data) Protection data is a 4-tuple:

$$\Psi_{i,k} = (\{\omega_i | \omega_i \in \Omega_{\mathcal{S}}\}, id(\omega_i), \Sigma, \mathcal{C})$$

where $\Omega_{\mathcal{S}}$ is a set of authorization units. $id(\omega_i)$ is identification of the last processed authorization unit in this $\Psi_{i,k}$. The last two elements of the 4-tuple are an authorization usage log ($\mathcal{U}$) and classification of the $\Psi_{i,k}$ ($\mathcal{C}$).

Through the last definition of protection data, we have returned to the beginning of this chapter, where *task template* has been defined (Def 5.3).

It has been a description of Active authorization model that we are going to incorporate into Calculus of communicating systems that is able to describe processes (computational tasks) and their dependencies. The reason we do it is to show possibility to use the security model in apparatus that is able to describe processes and their communication - functional description of tasks.

Chapter 6

Incorporating Security into CCS

Previous chapter introduced Active authorization model and an overview of CCS was also already given (Chapter 3). In the moment, we have got ideas about security in distributed systems and we have got formalism that has been created for description of communicating processes. What we are going to do is to combine those two into one system. We thus enrich formalism for processes with security considerations.

We are going to use the same notation as introduced in section 3.1. To define new properties for particular elements we shall use indexes distinguishing them from already defined symbols.

This chapter has several purposes. Firstly, it introduces basic approach for incorporating AAM into CCS. It also shows that some minor refinement of AAM need to be done. The second section of this chapter defines properties of secure systems. Those properties are divided according to two basic security controls *access control* and *flow control*. We are going to use this division to split security rules and properties between agents and transitions in CCS.

6.1 Refinement of AAM Principles

This section addresses three basic problems. The first problem is about distinguishing access and flow control. It is necessary for any other continuation of authorization system specification. The second problem is correct definition of classification/categorization. And the last issue is about changes in abstraction of CCS model used to define processes in distributed system.

6.1.1 Access Control and Flow Control

Authorization system specification, up to this moment, takes into account just access control. Also specification of Active authorization model in (Chapter 5) works only with access control - it is able to describe authorizations for particular accesses to resources, but it has no instruments for the control of data flow. But we have to consider both sides of the coin when solving problem of security control. The first side is about access control (discretionary access control) and the second is about flow control (mandatory access control). The notions in parenthesis are not exact because access control is determined by the authorization system and there is no way for users to make any decisions about it.

We have already talked about access control and have shown that it is bounded with agent constants and expressions that represent states of processes. The flow control is expressed as requirements of actions that represent communication between agents. Actions in the process calculus represent exchange of data, synchronization, all data flows that exist in the modeled system. You see, it is natural instrument to model and constrain data flow to define flow control.

To restrict data flow in extensive systems, we have to use general rules. Those rules are based on security differences between states of processes. Communication is assigned a value according to the place where it goes from and we say what security properties of the destination have to be. There are already defined security properties of states - authorization units. Authorization units contain initial authorizations and enabled authorizations. Which one should be used for the control of data flow? Initial authorizations are used do determine users that may start execution of the associated task, nothing more. On the other side, enabled authorizations say what the task is allowed to do. And we have got what we need, the latter is the information to be used for the flow control.

What we have to answer is the following question: Would it be useful to apportion enabled authorizations into two sets, one to access the s-node resources, and one for flow control, access to information received from other (communicating) processes? It is certain that communication between states need to be assigned security information and we should be able to say whether the consecutive task is authorized to process the received data.

We assume that it is very useful. In fact, it is necessary for covering security in systems based on communication. We therefore have to enrich Active authorization model that has been proposed without distinction between access and flow control. What we shall receive is a very powerful tool. E.g. by assigning definitions of tasks with classification for flow control, we are able to choose among several definitions of tasks in the run-time according to the data that have been generated during previous steps of workflows. It is dynamic change of workflow according to the previous execution's results!

We have discovered by analysis that only definition of authorization unit contains enabled privileges (see Def 5.15, page 53) and it is the only one definition that has to be changed.

Definition 6.1. (Authorization unit revisited) We have a set of abstract authorization templates Π^A. Let P_I^t be the proper set of initial authorizations and $P_E^t = (P_E^{A,t}, P_E^{F,t}) \subseteq \Pi^A \times \Pi^A$ be enabled authorizations consisting of the set of access control ($P^{A,t}$) and the set of flow control ($P^{F,t}$) authorizations. Further, let $S_e^t \subseteq S_0^t$ be a set of executors of the authorization unit:

$$S_e^t = \{s_e^t | \exists i, s_e^t \in S_i \wedge s_e^t \in \Phi_i(S_i, P_I^t)\}$$

Each task may be repeated several times, so we define number of initializations of the associated authorization unit $n \in \mathbb{N}$. The authorization unit ω_i is then defined as a 4-tuple:

$$\omega_i = (P_I^t, P_E^t, n, S_e^t)$$

The most important are the first three elements of ω_i. ■

In this place, we forego definition of the notion e-complete processes that denotes atomic processes and use the term in the following statement. Atomic processes are the most primitive from the security point of view.

Definition 6.2. (E-complete authorizations) Each e-complete process is assigned authorization information representing enabled authorizations. That information consists from an authorization that allows access to an s-node resource and an authorization that allows entry of data from other states.

$$P_{p_i^A, p_i^F}$$

where authorizations are elements of enabled authorizations of appropriate authorization unit $p_i^A \in P_E^{A,t}$ and $p_i^F \in P_E^{F,t}$. ■

6.1.2 Categorization of System Elements

There is still one problem we have not solved in previous definitions, yet. It is about security categorization (or classification) the transferred data and subjects shall be assigned. Each process that creates new data, applies all the data it receives (through communication as well as through direct access to s-node resources). Categorization of the output must respect properties of all input data. We have to go even further and to say that categorization of all data that are changed, created or stored on an s-node should look at categorization of all input data. This shall lead us to another security property of authorization system that has to be satisfied in case of data flow control.

We have used the word *respect* input data. Before we try to formalize this 'word' we have to define categorization. The following definition introduces exact specification of categorization (classification). The basic idea comes from BLP model [16] and lattice model for flow control [31, 32, 33].

Definition 6.3. (Classification) Each object has got assigned security classification that is defined on the set of data categories DC and on the set of security classes SC, $C = SC \times DC$. Classification is a couple $c_i = (sc_i, dc_i)$.

Each security class is an element of ordered set $SC = \{sc_1, \ldots, sc_k\}$ according to operation $\leq$ $(sc_1 \leq sc_2 \leq \cdots \leq sc_k)$.

Each category $dc_i \subseteq DC$ and categories constitute a partially ordered set of sets with minimal element $\emptyset$, maximum element DC and operation $\subseteq$.

Classifications constitute a lattice with minimum element $(sc_1, \emptyset)$ and maximum element (sc_k, DC). Elements are partially ordered:

$$(sc_1, dc_1) \leq (sc_2, dc_2) \Leftrightarrow (sc_1 \leq sc_2) \wedge (dc_1 \subseteq dc_2)$$

The lowest upper bound $\triangle$ is defined as

$$(sc_1, dc_1) \triangle (sc_2, dc_2) = (max(sc_1, sc_2), dc_1 \cup dc_2)$$

and greatest lower bound ∇ as

$$(sc_1, dc_1) \nabla (sc_2, dc_2) = (min(sc_1, sc_2), dc_1 \cap dc_2)$$

Each element c_i of security classification is called a *classification class*. ■

Definition 6.4. (Categorization) Assume, we have a classification C defined on the set of data categories DC and on the set of security classes SC. Assume that $SC = \{sc_1\}$. In other words, all data has got the same *security class*. We then call C a *categorization*. Each element of a given categorization is called *categorization class*. ■

When respecting security properties of input data, the classification of output data c is *join of classification classes* of all input data $c_1, \ldots, c_n$. It is defined as $c = c_1 \triangle \cdots \triangle c_n$.

Before we are ready to start with definition of properties of secure systems, a few preliminary definitions have to be stated. The first one is about secure actions.

Definition 6.5. (Secure action) Action in process calculus $(\alpha, \beta, \ldots)$ is denoted as follows:

$$\xrightarrow{\beta} P \xrightarrow{\alpha} P'$$

To specify a secure action, we have to assign it a classification class:

$$\xrightarrow{\beta_{c_\beta}} P_{c_P} \xrightarrow{\alpha_{c_\alpha}} P'_{c'_P}$$

Secure action α is assigned a classification class c_α computed as lowest upper bound of classification class c_P of resources that P is allowed to access and classification class c_β of action β.

$$c_\alpha = c_P \vartriangle c_\beta$$

■

Definition 6.6. (Classification function) We define classification function Θ that assigns each resource a classification $c_i \in C$, where C is a data classification.

$$\Theta : O \to C$$

The classification class assigned to a resource can not be changed. ■

Definition 6.7. (State classification) There is a function Γ that assigns a classification class to each state. Let's say that $\mathcal{P}$ is the set of all states, $C = \{c_1, c_2, \ldots\}$ is the classification ($(sc_1, \emptyset)$ is the minimum element of C), p_i enabled authorizations and ω_j are authorization units.

$$p_i = ((o_{i,A},\, a_{i,A},\, u_{i,A}),\, (c_{i,F},\, a_{i,F},\, u_{i_F}))$$
$$\omega_j = (P_I^t,\, (\{(o_{i,A},\, a_{i,A},\, u_{i,A})\}_{i \in I},\, \{(c_{j,F},\, a_{j,F},\, u_{j,F})\}_{i \in J}), S_e^t,\, n)$$

$\Gamma : \mathcal{P} \to C$ is defined as:

$$\Gamma(P_{p_i}) = \Theta(o_{i,A}) \vartriangle c_{i,F}$$
$$\Gamma(P_{\omega_j}) = \vartriangle_{i \in I} \Theta(o_{i,A}) \;\; \vartriangle_{i \in J} c_{j,F} \vartriangle (sc_1, \emptyset)$$
$$\Gamma(P_{P_I}) = (sc_1, \emptyset)$$
$$\Gamma(P_{\sigma_i \cup \sigma_j}) = \Gamma(P_{1,\sigma_i}) \vartriangle \Gamma(P_{2,\sigma_j})$$

■

6.1.3 Abstraction and Specialization of Authorizations

First problem we have to solve is total independence of *process calculus* on abstraction used to model a system. This abstraction is not so clear in Active authorization model. AAM works with basic entities - *authorization units* ω_i, that are defined as $\omega_i = (P_I^t, P_E^t, n, S_e^t)$ (see Def 5.15). This is the basic authorization element. It specifies requirements to initiate execution of a task step t (initial authorizations P_I^t) and set of privileges the executor receives (enabled authorizations P_E^t). Processing of an authorization unit may be viewed as application of single authorizations from P_E^t or as execution of the authorization unit as a whole (n is decreased). Subjects can see just execution of a task step. The authorization model needs to make difference among three basic abstraction levels for process modeling.

1. E-complete - Atomic processes on s-node level. Processes are *complete* to perform some atomic action defined by system (e.g. operation system) that is installed on a particular s-node. (An example may be to read a given file or to write a record in a database.)

2. A-complete - Atomic processes on the authorization system level. They are *complete* to be granted basic set of authorizations (expressed in an authorization unit by enabled authorizations) and to perform an atomic action defined in the context of the distributed system. (We may state examples like changing balance of an account or filing an insurance claim.)

3. W-complete - Composite tasks (workflows) that may be initiated by users. Those tasks represent the layer of a distributed system that is managed by users (users may decide whether initiate a workflow). With a-complete processes, users may decide when, but necessity to run a process is stated by the system.

Processes (or *states* in the process calculus) on the finest resolution, *a-complete processes*, change authorization unit ω_i by removing elements from the set of enabled authorizations of the authorization unit. Assume that $P_{E,i}^t = \{p_1, \ldots, p_j, \ldots, p_k\}$. $P_{E,i+1}^t = \{p_1, \ldots, p_j', \ldots, p_n\}$, where $p_j' = \Lambda(p_j)$.

$P_{E,i}^t, P_{E,i+1}^t$ are enabled authorizations from consecutive states of an authorization unit and p_j is the authorization applied in the former state. We are not interested in initial authorizations on this level of abstraction.

The second level of abstraction works with execution of authorization units (see Def 5.17). This level works with P_E^t as with a primitive entity. The set of initial authorizations is used here, but no users are allowed to initiate the task on this level of abstraction by themselves. Authorization units are parts of task steps of a workflow and the distributed system determines whether (and sometimes when) to execute them and searches for appropriate subjects when needed.

The highest level of abstraction is represented by tasks that may be initiated by users. To initiate a task, we need to define conditions for it - initial authorizations. Those authorizations are defined accordingly to the needs of application environment. The initial authorizations defined for particular steps t_j of the workflow are independent on the initial authorization of the workflow.

We do not need to assume sets of enabled authorizations on this abstraction level. The same may hold for the number of possible initializations n.

To demonstrate previous paragraphs, assume the following example described by the means of enriched process calculus. We have got defined a workflow P_{ω_p}. This is the highest level of abstraction. Looking into the workflow, we may see more detailed states of the process P.

$$P_{\omega_p} \overset{\text{def}}{=} P_{1,\omega_1} \xrightarrow{\alpha_{1,c_1}} P_{2,\omega_2} \xrightarrow{\alpha_{2,c_2}} \cdots$$

Each authorization unit ω_i enables other set of authorizations and defines, through its initial authorizations, other set of subjects able to run the associated task step.

The most detailed level of abstraction may be expressed as:

$$P_{i,\omega_i} \overset{\text{def}}{=} P_{i_1,\omega_i^1} \xrightarrow{\alpha_{i_1,c_i^1}} P_{i_2,\omega_i^2} \xrightarrow{\alpha_{i_2,c_i^2}} \cdots$$

This time, only one authorization unit (ω_i) is used and each processing state P_{i_k,ω_i^k} is characterized by one state (ω_i^k) of the authorization unit ω_i. Positions in the same unit's state (Def 5.17) differ only in the set of enabled authorizations P_E^t that decreases.

We have specified three different levels of abstraction needed for the authorization model. The basic question is whether it is possible to define some general rules for moving through levels of abstraction of the model.

To define simple rules for secure communication between states, we have to state properties that are important for different levels of abstraction.

1. E-complete processes shall be assigned authorizations needed for their execution. It means that processes need not be denoted by authorization units, but only with specific authorization templates (enabled privileges).

2. A-complete processes shall be assigned with authorization units as defined in Def 5.15. Each authorization unit must satisfy condition saying

that the set of enabled authorizations P_E^t is equal to the summation of all authorization templates assigned to underlying e-complete processes.

3. W-complete processes shall be assigned only authorizations needed for their initialization.

   There is no general rule that would specify which processes shall be assigned authorization unit or initial authorizations. It depends entirely on the designer of the model.

We should state some general definition, saying how to transform security information when moving to more or less detailed model. There are defined six transition rules in CCS (see Page 22): Prefix, Summation, Composition, Restriction, Relabelling and Constant. We can see that states may be in sequence (there are actions between pairs of them), there may be choice *1 of n* (see Summation) and states may represent parallel processes (Composition). It means that we have to define three different types of abstracting agents into agent expressions or Constants. But beforehand, we define operations for enabled privileges.

Definition 6.8. (Authorization operators) Assume that there are sets of authorizations $P_1 = \{p_{1,i} \mid i \in I \wedge p_{1,i} = (c_{1,i}, a_{1,i}, n_{1,i})\}$ and $P_2 = \{p_{2,j} \mid j \in J \wedge p_{2,j} = (c_{2,j}, a_{2,j}, n_{2,j})\}$[1]. We define operators $\oplus$ and $\otimes$ over $\Pi^A \times \Pi^A \rightarrow \Pi^A$:

1. $P_1 \oplus P_2 = \{p_i \mid i \in \begin{cases} I - J & \Rightarrow p_i = (c_{1,i},\ a_{1,i},\ n_{1,i}) \\ J - I & \Rightarrow p_i = (c_{2,i},\ a_{2,i},\ n_{2,i}) \\ I \cap J & \Rightarrow p_i = (c_{1,i},\ a_{1,i},\ n_{1,i} + n_{2,i}) \end{cases} \}$

2. $P_1 \otimes P_2 = \{p_i \mid i \in \begin{cases} I - J & \Rightarrow p_i = (c_{1,i},\ a_{1,i},\ n_{1,i}) \\ J - I & \Rightarrow p_i = (c_{2,i},\ a_{2,i},\ n_{2,i}) \\ I \cap J & \Rightarrow p_i = (c_{1,i},\ a_{1,i},\ \max(n_{1,i}, n_{2,i})) \end{cases} \}$

To shorten notation of $\oplus$ and $\otimes$ with many authorizations, we shall write $\bigoplus_{i \in I} P_i$, resp. $\bigotimes_{i \in I} P_i$ for $\oplus$, resp. $\otimes$ of all P_i, where $i \in I$. ■

We have defined three basic types of processes. It means three forms of authorization information. When not knowing which abstraction is used we obtain fourth type of process and form of authorizations. That does not specify enabled authorizations, neither authorization units nor initial authorizations, but has to preserve those information from underlying states.

[1]It is possible that authorizations are defined as specific templates. We shall use function Θ_i defined on the appropriate s-node to obtain abstract authorization templates as needed, in this case.

To make it easier, we start with enabled authorizations allowing only access control. The basic principle used for access control is that access authorizations are bound with states where defined and it is not possible to *move* it to joined states established from several less abstract states. This is the reason we have to use *counted privileges* P_E^+ and P_I^+ to retain information for correct construction of authorization units.

$a_i = (p_{e,i},\ \emptyset,\ \emptyset,\ p_{e,i},\ \emptyset)$	for e-complete process
$a_i = (\emptyset,\ \omega,\ \emptyset,\ P_E^+,\ P_I^+)$	for a-complete process
$a_i = (\emptyset,\ \emptyset,\ P_{I,i},\ P_E^+,\ P_I^+)$	for w-complete process
$a_i = (\emptyset,\ \emptyset,\ \emptyset,\ P_E^+,\ P_I^+)$	for all other processes

The flow control brings into account another principle. We have to accumulate privileges for flow control to ensure feasibility of tasks[2]. This time, enabled authorizations are composed from two elements - privileges for access and flow control. We receive slightly different four formats of security information.

Definition 6.9. (Authorization information) Each state is assigned the authorization information a_i determining access and flow control. This information may be in one of four forms.

1. For e-complete processes $a_i = ((p_{e,i}^A, P_{E,i}^F),\ \emptyset,\ \emptyset,\ (p_{e,i}^A, P_{E,i}^F),\ \emptyset)$.
2. For a-complete processes $a_i = ((\emptyset, P_{E,i}^{F+}),\ \omega,\ \emptyset,\ P_E^+,\ P_I^+)$.
3. For w-complete processes $a_i = ((\emptyset, P_{E,i}^{F+}),\ \emptyset,\ P_{I,i},\ P_E^+,\ P_I^+)$.
4. For all other processes $a_i = ((\emptyset, P_{E,i}^{F+}),\ \emptyset,\ \emptyset,\ P_E^+,\ P_I^+)$.

The set of all authorization information shall be denoted as $\mathcal{A}_\mathcal{I}$. ■

Joining of several states into one general notion and it also allows joining of parallel processes. Such processes are independent and their authorization information shall stay independent, too. It means that each state is assigned a set of authorization information. That fact is preserved in the following definition.

Definition 6.10. (Authorization abstraction - $\triangle$ operator) Assume that there are agent expressions E_{1,a_1}, E_{2,a_2} and $a_1 = \{a_{1,1}, a_{1,2}, \dots a_{1,m}\}$, $a_2 = \{a_{2,1}, a_{2,2}, \dots a_{2,n}\}$ are sets of authorization information. P_E^+, $P_{E,\_}^+$ are sets of *counted enabled authorizations* and P_I^+, $P_{I,\_}^+$ are sets of *counted initial authorizations.* All counted authorizations are initialized to empty sets. A_{a_0} is an agent constant with $a_0 = a_1 \triangle a_2$. The operator $\triangle$: $\mathcal{A}_\mathcal{I} \times \mathcal{A}_\mathcal{I} \rightarrow \mathcal{A}_\mathcal{I}$ is defined as follows:

[2]This principle allows to model flow control in processes on any abstraction level.

1. $A_{a_0} = E_{1,a_1} \xrightarrow{\alpha} E_{2,a_2}$ implies a_0 with $P_E^+ = \bigoplus_{i\in\{1,..,m\}}(P_{E,1,i} \oplus P_{E,1,i}^+) \oplus \bigoplus_{j\in\{1,..,n\}}(P_{E,2,j} \oplus P_{E,2,j}^+)$ and $P_I^+ = \bigoplus_{i\in\{1,..,m\}} P_{I,1,i}^+ \oplus \bigoplus_{j\in\{1,..,n\}} P_{I,2,j}^+$. We may interpret it as synchronization of parallel processes and their merging into one resulting process.
   (a) $a_0 = \{(\emptyset,\ \omega,\ \emptyset,\ P_E^+,\ P_I)\}$ if defined $\omega = (P_I,\ P_E^+,\ n)$ such that $P_I^+ \subseteq P_I$ for A.
   (b) $a_0 = (\emptyset,\ \emptyset,\ P_{I,0},\ P_E^+,\ P_I^+)$ if defined only initial authorizations $P_{I,0}$ for A.
   (c) $a_0 = (\emptyset,\ \emptyset,\ \emptyset,\ P_E^+,\ P_I^+)$ otherwise.
2. $A_{a_0} = E_{1,a_1} \mid E_{2,a_2}$ implies a_0 as follows[3]:
   $a_0 = \{a_{1,1},\ \ldots,\ a_{1,m},\ a_{2,1},\ \ldots,\ a_{2,n}\}$.
3. $A_{a_0} = E_{1,a_1} + E_{2,a_2}$ implies a_0 where $P_E^+ = \bigotimes_{i\in\{1,..,m\}}(P_{E,1,i} \otimes P_{E,1,i}^+) \otimes \bigotimes_{j\in\{1,..,n\}}(P_{E,2,j} \otimes P_{E,2,j}^+)$ and $P_I^+ = \bigotimes_{i\in\{1,..,m\}} P_{I,1,i}^+ \otimes \bigotimes_{j\in\{1,..,n\}} P_{I,2,j}^+)$
   (a) $a_0 = (\emptyset,\ \omega,\ \emptyset,\ P_E^+,\ P_I)$ if defined $\omega = (P_I,\ P_E^+,\ n)$ such that $P_I^+ \subseteq P_I$ for A.
   (b) $a_0 = (\emptyset,\ \emptyset,\ P_{I,0},\ P_E^+,\ P_I^+)$ if defined only the set of initial authorizations $P_{I,0}$ for A.
   (c) $a_0 = (\emptyset,\ \emptyset,\ \emptyset,\ P_E^+,\ P_I^+)$ otherwise.

Authorization transformation from agent expressions into agent constant shall be called *authorization abstraction* and counter-transformation shall be called *authorization specialization*. ■

6.2 Definition Of Secure System

In this section, properties whose compliance ensures the security of a given system are stated. We also have to formalize them to be used as basis for proving security of the proposed authorization system.

Security properties are divided into two sections, access control and flow control. There may be systems those tasks do not allow communication or communication is not important feature. Such systems define categorization of data, but all users are allowed (generally) to access all data sent among task steps of workflows and tasks are used only to define their job. In that case

[3]Concurrent systems may be naturally expressed as (restricted) compositions. It means that there should remain separation of security information for all parallel processes.

only access control is what we are interested in. On the other side, flow control makes the system generally secure and should enable definition of rules that correspond with principles of mandatory access control.

We state properties necessary for systems to be called secure. They are divided into two groups, related to access control and flow control.

6.2.1 Access Control of Secure System

We have already several times mentioned that authorization unit is the basic information to define security properties of tasks. This is confirmed in the following items that use the notion to define properties of secure system.

1. Each workflow definition defines maximum number of initializations for all authorization units it contains.

2. Authorization unit defines, through enabled authorizations, set of authorizations that may be granted to a subject (a task or a user).

3. A subject may be granted just the minimum set of privileges necessary to execute the authorization unit. This set is specified by enabled authorizations.

4. A subject is granted authorizations defined in an authorization unit only for the time he executes it. The necessary condition for the execution is satisfaction of initial authorizations defined in the executed authorization unit.

5. A subject may execute only one task at a time.

6. Application of any authorization from enabled authorizations causes decreasing of available initializations of given authorization unit[4].

As you can see, properties of access control correspond with properties of Active authorization system. Property (1) is implied by Def 6.1 as well as Property (2). Property (3) is expressed in definition of $\triangle$ operator (Def 6.10). Property (6) is enforced by definition of authorization application - Def 5.11 and Def 5.18 (validity of authorization unit). Requirements (4) and (5) are directed towards properties of Workflow engine.

[4]It holds even for fail of the unit execution. I understand that there may be different opinions on this question. I can imagine rules that define number of possible initialization with fail, but for simplicity we use this rule.

6.2.2 Flow Control of Secure System

To define flow control we need classification (Def 6.3) and its elements, classification classes, or categorization (Def 6.4).

1. Each data in the system is assigned a classification class. This class is defined as static or dynamic according to type of the given resource. The static class does not change during the data existence. The dynamic class is affected by flow control (by classes of all input data).

2. There is defined a *flow relation* on pairs of classification classes. Given two classes c_1 and c_2, the relation $c_1 \rightarrow c_2$ is valid if, and only if, information in c_1 is allowed to flow into c_2 ($c_1 \leq c_2$, according to Def 6.3).

3. There is defined operation $\triangle$ for *classification class combination* that defines classification class resulting from interference of two classification classes: $c_1 \triangle c_2 = c_3$.

4. There does not exist a sequence of operations that violates the flow relation. If a value $f(a_1, a_2, \ldots, a_n)$ flows into an object with class b, then the relation $a_1 \triangle a_2 \triangle \ldots \triangle a_n \rightarrow b$ must be satisfied.

Item (1) specifies property of Workflow engine and is necessary for all other flow control properties. Item (2) states when a data flow is correct. Next property shall be used for classification of output ports (output data) and item (4) puts new requirement on subsequent task steps.

Access control is stated by AAM. When showing security of a new task, all we need to do is to show compliance with properties of flow control.

Chapter 7

Secure CCS

This chapter finishes explanation that began by defining Architecture of the Authorization System (Chapter 4). Proposition of authorization system is incorporated into process calculus that allows formalization of communication processes and their simulation.

We shall begin this chapter by introducing foundations of the calculus - *action*, *state*, *port*, . . . that are changed the way that enables definition of relevant authorization information. The resulting instrument is suitable for modeling security properties of distributed systems.

The skeleton of this chapter is grounded on specification of process calculus [45] that is extended as necessary. Except definition of secure model, we have a secondary target - make as few changes of original process calculus as possible. It results in similarity with the original specification, but the result is **secure**.

Previous chapter has divided authorization system into access control and flow control. Access control is able to decide access requests to static resources according to tasks' definitions. We have said that privileges can not be joined, they are closely connected with atomic tasks. It means that access control is provided by s-nodes and requirements for secure system are ensured by definition and usage of secure states. On the other side, flow control secures movement of dynamic resources that rise and die within tasks' processing. CCS is about communication and that is why we are going to talk just about flow control in this chapter.

We assume that all notions defined in Chapter 3 are known.

7.1 Action and State

We have discussed some basic properties that have to be satisfied for modeling of access and flow control. Let us start with actions that represent communication[1] between processes. Assume, we have an infinite set $\mathcal{A}$ of names and we use $a, b, c, \ldots \in \mathcal{A}$ as names. Those names are used for communication between agents.

Definition 7.1. (Names) Assume, there is a set of names $\mathcal{A}$ and data classification $C = \{c_1, c_2, \ldots, c_n\}$ defined on DC and SC (see Def 6.3 on page 60). Each element $a \in \mathcal{A}$ is assigned a classification $c_i \in C$. We denote a name as a_{c_i}. ■

Definition 7.2. (Co-names) Assume, there is a set of names $\mathcal{A}$. We denote by $\bar{\mathcal{A}}$ the set of co-names. We say that $\bar{a}_{c_i} \in \bar{\mathcal{A}}$ is complement of $a_{c_i} \in \mathcal{A}$. The classification class c_i of $\bar{a}$ is the same as the classification class of a. ■

Definition 7.3. (Labels) Labels is union of sets of names and co-names. $\mathcal{L} = \mathcal{A} \cup \overline{\mathcal{A}}$. ■

States in the system are represented as agents (agent constants, agent expressions, ...). We have said that each state possesses its security information. The type of information depends on the abstraction the state represents (e-complete, a-complete or w-complete process). Each state possesses generally a set of tuples of static security information (they are used for access control). Each tuple may contain enabled authorizations $P_E^t = (P_{E,t}^A, P_{E,t}^F)$ where $P_E^{A,t} = \{p_{e,t}^A\} \vee \emptyset$ and either authorization unit ω_t or a set of initial authorizations P_I^t. The allowed forms of authorization information are:

$$\begin{array}{ll}
a_i = ((p_{e,i}^A, P_{E,i}^F),\ \emptyset,\ \emptyset,\ (p_{e,i}^A, P_{E,i}^F),\ \emptyset) & \text{for e-complete process} \\
a_i = ((\emptyset, P_{E,i}^{F+}),\ \omega,\ \emptyset,\ P_E^+,\ P_I^+) & \text{for a-complete process} \\
a_i = ((\emptyset, P_{E,i}^{F+}),\ \emptyset,\ P_I^i,\ P_E^+,\ P_I^+) & \text{for w-complete process} \\
a_i = ((\emptyset, P_{E,i}^{F+}),\ \emptyset,\ \emptyset,\ P_E^+,\ P_I^+) & \text{for all other processes}
\end{array}$$

To shorten notation of that information, we denote agents with σ_t that represents all authorization information assigned to the state. We may write a transition as:

$$P_{\sigma_i} \xrightarrow{\ell_c} Q_{\sigma_j}$$

[1]We should talk about synchronization because we are going to work with basic calculus that does not offer value passing. We have got two reasons for that. Basic calculus is simpler for development of theory and CCS offers simple way for reducing full calculus into the basic one.

P_{σ_i} and Q_{σ_j} are agents, $\ell_c \in \mathcal{L}$ is a label.

We still need to introduce one last action. It is action that arises each time when b_{c_i} and $\overline{b}_{c_i}$ (complementary actions) are invoked simultaneously. This action is internal for a composite agent, i.e. the same action arises from any pair of complementary actions.

Definition 7.4. (Perfect actions) The action that denotes simultaneous invocation of any complementary actions $a_c, \overline{a}_c$ is called *perfect* or *completed action* and is denoted τ_c. This action has got assigned the same classification as actions it represents - c.
τ_* denotes set of all perfect actions. ■

All perfect actions have the same visible effects - none. We need to preserve classification of the original actions and that is the only difference among perfect actions.

With τ_* actions, we may define set of all actions $Act = \mathcal{L} \cup \tau_*$.

Perfect actions play very important role because they allow us to ignore all internal (perfect) actions of composite systems. This is enabled by the fact that τ_* actions do not represent any potential communication and they are therefore not directly observable. But because each τ_c represents two actions it is reasonable to apply restriction on the composite system. It should not use the *hidden* actions.

Definition 7.5. (Port) Each state has ports denoted by names from $\mathcal{L}$. The ports denoted by any $a_{c_i} \in \mathcal{A}$ are able to receive (input ports), ports denoted by $\overline{a}_{c_i} \in \overline{\mathcal{A}}$ are able to transmit (output ports).
To define an action a_{c_i} from agent P_{σ_i} to Q_{σ_j}, there must be defined ports a_{c_i} and $\overline{a}_{c_i}$ in P_{σ_i} and Q_{σ_j}, respectively. ■

Talking about ports, they have to satisfy security properties defined for secure system. It is especially item 2 and item 4 of flow control.

Definition 7.6. (Secure port) Secure port is a port that satisfies conditions for secure system.

1. *Input port* - classification class of the input port $a_{c_i} \in \mathcal{A}$ is the same as of the corresponding output port $\overline{a}_{c_j} \in \mathcal{A}$.

$$c_i = c_j$$

2. *Output port* - classification class of each output port $\overline{a}_{i,c_i} \in \mathcal{A}$ for the given state P_σ is equal or greater than *combination* of all input ports

$a_{j,c_j} \in \mathcal{A}, \forall j \in J$ of the state P_σ and classification of all static resources accessed in P_σ.

$$\forall j \in J: \ \Gamma(P_\sigma) \leq c_j$$

Input ports in the definition are an abstraction that also represents places to receive all *data flows* from local resources (access control).[2] ■

Lemma 7.1. *Secure port preserves security of distributed system.*

Proof. We use Def 7.5 declaring that action and input and output ports it connects, are all assigned with the same classification class. We are talking about data flow, so we have to prove fulfillment of property 2 for input ports and property 4 for output ports (page 68).

1. A transition represents a data flow, so *flow relation* $\rightarrow$ must hold. Assume, we have a transition $P_{\sigma_p} \overset{a_{c_a}}{\rightarrow} Q_{\sigma_q}$ that connects ports a_{c_a} and $\overline{a}_{c_a}$. It implies flow relation $c_a \rightarrow c_a$. Because $c_a \leq c_a$ by Def 6.3, the flow relation is valid and property is satisfied.
2. A state represents operations and therefore inference of classification classes that are represented by classification classes of all input ports $a_{i \in I}$ of the given state. The relation is

   $$\triangle_{i \in I} \, c_i \rightarrow c_j \Leftrightarrow c'_j \rightarrow c_j$$

   According to Def 7.6 (2), $c'_j \leq c_j$. It means that by Def 6.3, the flow relation is valid and property is satisfied.

□

Definition 7.7. (Secure state) The state that contains only secure ports and that satisfies all relevant requirements of access control for secure system is called *secure state.* ■

Remember definition of state classification (Def 6.7). Now, when ports have been introduced, we could redefine classification of state using its ports (flow control) and inner properties (access control). The resulting classification of the state would be combination of input ports' classification classes (it covers also classification of resources, the state is allowed to access).

[2]The inequality for output ports is very important for inheritance of processes defined in Chapter 8. It would not be possible if just equality were used.

Definition 7.8. (State classification in CCS) There is a function Γ that assigns a classification class to each state. Let's say that $\mathcal{P}$ is the set of all states, $C = \{c_1, c_2, \ldots\}$ is the classification (let $(sc_1, \emptyset)$ be the minimum element of C) and P_σ is a secure state and $\{a_{1,c_1}, \ldots, a_{n,c_n}\}$ is the set of all input ports in P_σ. $\Gamma : \mathcal{P} \to C$ is defined as:

$$\Gamma(P_\sigma) = \triangle_{i=1..n}\, c_i \;\triangle\; (sc_1, \emptyset)$$

In case of no input ports in P_σ $(n = 0)$, then holds $\Gamma(P_\sigma) = (sc_1, \emptyset)$. ■

7.2 The Basic Language

We are going to use the syntax defined in the previous section plus $\alpha_{c_i}, \beta_{c_j}, \ldots \in$ *Act*, $\ell_{c_i}, \ell'_{c_j} \in \mathcal{L}$, $K, L \subset \mathcal{L}$. A *relabelling function* $f : \mathcal{L} \to \mathcal{L}$, such that $f(\overline{\ell}_{c_i}) = \overline{f(\ell_{c_i})}$ and $f(\tau_{c_i}) = \tau_{c_i}$.

We also introduce a set $\mathcal{X}$ of *agent variables* $(X_{\sigma_x}, Y_{\sigma_y}, \ldots \in \mathcal{X})$ and $\mathcal{K}$ of *agent constants* $(A_{\sigma_a}, B_{\sigma_b}, \ldots \in \mathcal{K})$[3]. At last, there is the set $\mathcal{E}$ of *agent expressions*. Let $E_{\sigma_e}, F_{\sigma_f}, \ldots$ range over $\mathcal{E}$. $\mathcal{E}$ is the smallest set containing $\mathcal{X}$ and $\mathcal{K}$ and the following expressions, assuming $E, E_i \in \mathcal{E}$.

1. $F_{\sigma_f} = \alpha_{c_\alpha}.E_{\sigma_e}$, a *Prefix* $(\alpha \in Act)$, $c_\alpha \triangle \sigma_e \leq \sigma_f$
2. $F_{\sigma_f} = \Sigma_{i \in I} E_{i,\sigma_i}$, a *Summation* ($I$ an indexing set), $\triangle_{i \in I}\, \sigma_i \leq \sigma_f$
3. $F_{\sigma_f} = E_{1,\sigma_1} | E_{2,\sigma_2}$, a *Composition*, $\sigma_1 \triangle \sigma_2 \leq \sigma_f$
4. $F_{\sigma_f} = E_{\sigma_e} \backslash L$, a *Restriction*$(L \subseteq \mathcal{L})$, $\sigma_f = \sigma_e$
5. $F_{\sigma_f} = E_{\sigma_e}[f]$, a *Relabelling*($f$ a relabelling function), $\sigma_f = \sigma_e$

Inequality in expressions (1)-(3) is introduced again to allow secure inheritance of processes.

The expression (2) is written more generally because of possible future need for infinite sum or empty sum $(\mathbf{0} = \sum_{i \in \emptyset} E_i)$.

We also define *agents*. Agents are agent expressions without variables[4]. We denote them $\mathcal{P}$ and we shall let P_{σ_p}, $Q_{\sigma_q}, \ldots$ range over agents.

[3] We are not going to fully use distinction between agent constants and variables. The main reason of this separation is introduction of recursion.

[4] When we move to the value-passing calculus, the definition shall be changed from *no variables* to *no free variables*.

To decrease number of parenthesis, we adopt the convention that the combinators have decreasing binding power in the following order: Restriction and Relabelling (tightest), Prefix, Composition, Summation - it is the same as in the original CCS.

We shall use the general notion of a *labelled transition system*

$$(S, T, \{\xrightarrow{t} : \ t \in T\})$$

which consists of a set S of states, a set T of transition labels, and a secure transition relation $\xrightarrow{t} \subseteq S \times S$ for each $t \in T$.

For our purposes, we take S to be $\mathcal{E}$ and T to be *Act*. We wish to define secure transitions of each composite agent in terms of the transitions of its component agent(s). Remember transition rules defined on Page 22. What we need to do is set conditions to ensure security of a labelled transition system.

It is impossible to preserve certain *vagueness* of CCS in the sense of abstraction. We therefore define secure transition rules with the following conditions. Agents $P_{\sigma_p}, P'_{\sigma'_p}, Q_{\sigma_q}, \ldots$ connected with action ℓ_{c_ℓ} must have input/output ports ℓ_{c_ℓ} ($\bar{\ell}_{c_\ell}$ resp.). When there is a perfect action τ_{c_τ} then agents have to have defined original ports. It is ensured by e-complete processes, but it need not hold for *joined* states[5].

$$\textbf{Act}\ \frac{\Gamma(P_{\sigma_p}) \geq c_\alpha}{\alpha_{c_\alpha}.P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P_{\sigma_p}}$$

$$\textbf{Sum}_j\ \frac{P_{j,\sigma_j} \xrightarrow{\alpha_{c_\alpha}} P'_{j,\sigma'_j} \quad \Gamma(P'_{j,\sigma'_j}) \geq c_\alpha \geq \Gamma(P_{j,\sigma_j})}{\sum_{i \in I} P_{i,\sigma_i} \xrightarrow{\alpha_{c_\alpha}} P'_{j,\sigma'_j}}\ (j \in I)$$

$$\textbf{Com}_1\ \frac{P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \quad \Gamma(P'_{\sigma'_e}) \geq c_\alpha \geq \Gamma(P_{\sigma_p})}{P_{\sigma_p}|Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}|Q_{\sigma_q}}$$

$$\textbf{Com}_2\ \frac{Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q} \quad \Gamma(Q'_{\sigma'_f}) \geq c_\alpha \geq \Gamma(Q_{\sigma_q})}{P_{\sigma_p}|Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} P_{\sigma_p}|Q'_{\sigma'_q}}$$

[5]You have to define e-complete processes on another level. It would imply change of data flows and their classification.

$$\textbf{Com}_3 \frac{P_{\sigma_p} \xrightarrow{\ell_{c_\ell}} P'_{\sigma'_p} \quad Q_{\sigma_q} \xrightarrow{\bar{\ell}_{c_\ell}} Q'_{\sigma'_q} \quad \Gamma(P'_{\sigma'_e}),\ \Gamma(Q'_{\sigma'_f}) \geq c_\ell \geq \Gamma(P_{\sigma_p}),\ \Gamma(Q_{\sigma_q})}{P_{\sigma_p}|Q_{\sigma_q} \xrightarrow{\tau_{c_\ell}} P'_{\sigma'_p}|Q'_{\sigma'_q}}$$

$$\textbf{Res} \frac{P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \quad \Gamma(P'_{\sigma'_e}) \geq c_\alpha \geq \Gamma(P_{\sigma_p})}{P_{\sigma_p}\backslash L \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}\backslash L} \ (\alpha_{c_\alpha}, \bar{\alpha}_{c_{\bar{\alpha}}} \notin L)$$

$$\textbf{Rel} \frac{P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \quad \Gamma(P'_{\sigma'_e}) \geq c_\alpha \geq \Gamma(P_{\sigma_p})}{P_{\sigma_p}[f] \xrightarrow{f(\alpha_{c_\alpha})} P'_{\sigma'_p}[f] \quad \Gamma(P_{\sigma_p}[f]) = \Gamma(P_{\sigma_p})}$$

$$\textbf{Con} \frac{P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \quad \Gamma(P'_{\sigma'_p}) \geq c_\alpha \geq \Gamma(P_{\sigma_p})}{A_{\sigma_a} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \quad \sigma_a = \sigma_p} \ (A_{\sigma_a} \overset{\text{def}}{=} P_{\sigma_p})$$

For example, the rule for summation can be read as follows: if any one summand P_{j,σ_j} of the sum $\sum_{i\in I}$ has an action and the action satisfies the *flow relation*, then the whole sum has also that action.

Whenever $E_{\sigma_e} \overset{\alpha_{c_\alpha}}{\to} E'_{\sigma'_e}$ and α_{c_α} is a *secure action* (see Def 6.5), we call the pair $(\alpha_{c_\alpha}, E'_{\sigma'_e})$ a *secure immediate derivative of* E_{σ_e} and $E'_{\sigma'_e}$ an α_{c_α}*-secure derivative* of E_{σ_e}.

Analogously, whenever $E_{\sigma_e} \xrightarrow{\alpha_{1,c_1}} \cdots \xrightarrow{\alpha_{n,c_n}} E'_{\sigma'_e}$ and $\alpha_{1,c_1}, \alpha_{2,c_2}, \ldots$ are all secure actions, we call $(\alpha_{1,c_1} \cdots \alpha_{n,c_n}, E'_{\sigma'_e})$ a *secure derivative* of E_{σ_e}, $\alpha_{1,c_1} \cdots \alpha_{n,c_n}$ a *secure action-sequence* of E_{σ_e} and $E'_{\sigma'_e}$ an $\alpha_{1,c_1} \cdots \alpha_{n,c_n}$*-secure derivative* of E_{σ_e}. In the case $n = 0$, we have that ϵ_c is a secure action-sequence of E_{σ_e} and E_{σ_e} is a derivative of itself.

Of course, we need the notion of *sort*. The definition is analogous to Def 3.1 introduced in subsection 3.1.5.

7.3 The Expansion Law

We only briefly resume the expansion law that is to be needed in the following text. The principle stays the same, we only add security information to all actions and states.

Proposition 7.2. ***(The expansion law)***
Let $P_{\sigma_0} \equiv (P_{1,\sigma_1}[f_1] \mid \ldots \mid P_{n,\sigma_n}[f_n])\backslash L$, with $n \geq 1$. Then

$$\begin{aligned} P_{\sigma_0} = \sum \Big\{ & f_i(\alpha_c).(P_{1,\sigma_1}[f_1] \mid \ldots \mid P'_{i,\sigma'_i}[f_i] \mid \ldots \mid P_{n,\sigma_n}[f_n])\backslash L \mid \\ & P_{i,\sigma_i} \xrightarrow{\alpha_c} P'_{i,\sigma'_i},\ f_i(\alpha_c) \notin L \cup \overline{L}, \sigma'_i = \sigma_i \ \triangle \ c, \Big\} \\ + \sum \Big\{ & \tau_c.(P_{1,\sigma_1}[f_1] \mid \ldots \mid P'_{i,\sigma'_i}[f_i] \mid \ldots \mid P'_{j,\sigma'_j}[f_j] \mid \ldots \mid P_{n,\sigma_n}[f_n])\backslash L \mid \\ & P_{i,\sigma_i} \xrightarrow{\ell_{1,c}} P'_{i,\sigma'_i}, P_{j,\sigma_j} \xrightarrow{\ell_{2,c}} P'_{j,\sigma'_j}, f_i(\ell_{1,c}) = \overline{f_j(\ell_{2,c})},\ i < j \\ & \sigma'_i = \sigma_i \triangle c,\ \sigma'_j = \sigma_j \triangle c \Big\},\ \sigma_0 = \triangle_{1..n}\, \sigma_i \triangle c \end{aligned}$$

We are going to use special case of the expansion law for $n = 1$ for definition of inheritance of secure processes.

Corollary 7.3. *We say that the following special cases of the expansion law are valid and secure.*

1. $(\alpha_c.Q_{\sigma_q})\backslash L = \begin{cases} \mathbf{0} & \text{if} \quad \alpha_c \in L \cup \bar{L} \\ \alpha_c.Q_{\sigma_q}\backslash L & \text{otherwise} \end{cases}$
2. $(\alpha_c.Q_{\sigma_q})[f] = f(\alpha_c).Q_{\sigma_q}[f]$
3. $(Q_{\sigma_q} + R_{\sigma_r})\backslash L = Q_{\sigma_q}\backslash L + R_{\sigma_r}\backslash L$
4. $(Q_{\sigma_q} + R_{\sigma_r})[f] = Q_{\sigma_q}[f] + R_{\sigma_r}[f]$

Proof. The security of equations is ensured because there are no changes of security information. The only change is in the first item (inactive agent). The action α_c is forbidden and we therefore can not reach state Q_{σ_q} as well as its security information. The same holds for the action. The inactive agent $\mathbf{0}$ has therefore no security information (to be precise there should be tuple of empty sets).

1. When using the expansion law without Relabelling (it is identity) with $n = 1$ then $P_{\sigma_p} \equiv \alpha_c.Q_{\sigma_q}$. The second summation contains no terms because there are no distinct components to communicate - yields $\mathbf{0}$. But because there is only one action α_c the result is $\mathbf{0}$ or $\alpha_c.Q_{\sigma_q}$ when the action is or is not in $L \cup \overline{L}$.

2. We use empty restriction set and by applying the expansion law for $n = 1$ and $P_{\sigma_p} \equiv \alpha_c.Q_{\sigma_q}$, the result is immediate.

3. We again set $n = 1$ and $f_1 = Id$. We use the fact that $(Q_{\sigma_q} + R_{\sigma_r}) \overset{\alpha_c}{\rightarrow} P_{\sigma_p}$ iff $Q_{\sigma_q} \overset{\alpha_c}{\rightarrow} P_{\sigma_p}$ or $R_{\sigma_r} \overset{\alpha_c}{\rightarrow} P_{\sigma_p}$.

4. We set $n = 1$ and $L = \emptyset$, the rest is the same as for the previous item.

□

7.4 Strong Bisimulation

In this section, we shall set up the notion of equivalence between secure agents. We have already introduced definition of strong bisimulation in process calculus (see Section 3.3.1). We are going to proceed in the same way and highlight changes made in order to define relation correct for secure agents.

Definition 7.9. A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over secure agents is a *strong secure bisimulation* if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ implies, for all secure actions $\alpha_{c_\alpha} \in Act$,

1. Whenever $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\rightarrow} P'_{\sigma'_p}$ then, for some $Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\rightarrow} Q'_{\sigma'_q}$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$

2. Whenever $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\rightarrow} Q'_{\sigma'_q}$ then, for some $P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\rightarrow} P'_{\sigma'_p}$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$

Secure states (agents) imply secure ports. It means that all communication between states is secure. ■

The reader may have noticed that we do not require the same classification for any two corresponding states as it is in the case of corresponding actions. This relaxation of the definition may be regarded as an omission, but there is a reason behind. We wish to leave open the inference process of action classifications. Only the fact that action classification is equal to the classification of the output ports was explicitly stated. However, it does not necessary follow that we would need to define further restrictions. Ours is the simple model without this specific choice enabling us to leave open the other possibilities.

This assumption is further supported by the fact that the system administrator defines data flows between particular tasks. He is, however, not able to see definitions of those tasks since these lie in the responsibility area of *s-node administrators*.

We have defined *secure ports* as an abstraction of 'transmission places' in states and classification of output ports and necessary classification of corresponding action. It is basis for the following lemma.

Lemma 7.4. *If $\mathcal{S}$ is a strong bisimulation and $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ then classifications of output ports in P_{σ_p} and Q_{σ_q} equal.*

Proof. The proof follows from the definition of secure action. If $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} \cdots$, then classification c_α of action α is equal to classification of output port.

If $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$, then

1. $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \Rightarrow \exists Q'_{\sigma'_q} : Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q} \Longrightarrow \Gamma(\alpha_{c_\alpha}) = \Gamma(\widehat{\alpha}_{c_\alpha}) = c_\alpha$ for both actions as well as output ports.
2. $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q} \Rightarrow \exists P'_{\sigma_p} : P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \Longrightarrow \Gamma(\alpha_{c_\alpha}) = \Gamma(\widehat{\alpha}_{c_\alpha}) = c_\alpha$ for both actions as well as output ports.

In fact, any of the equations is sufficient. □

Proposition 7.5. *Assume that each $\mathcal{S}_i (i = 1, 2, \ldots)$ is a strong secure bisimulation. Then the following relations are all strong secure bisimulations:*

$$\begin{array}{llll} (1) & Id_p & (3) & \mathcal{S}_1\mathcal{S}_2 \\ (2) & \mathcal{S}_i^{-1} & (4) & \bigcup_{i \in I} \mathcal{S}_i \end{array}$$

Proof. (1) and (2) are easy to prove by substituing P_{σ_p} for Q_{σ_q} (1) and exchanging P_{σ_p} with Q_{σ_q} (2). For (3), suppose that $(P_{\sigma_p}, R_{\sigma_r}) \in \mathcal{S}_1\mathcal{S}_2$. Then for some Q_{σ_q} we have $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}_1$ and $(Q_{\sigma_q}, R_{\sigma_r}) \in \mathcal{S}_2$

Now let $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. Then for some $Q'_{\sigma'_q}$ we have, since $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}_1$,

$$Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q} \text{ and } (P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}_1$$

Also since $(Q_{\sigma_q}, R_{\sigma_r}) \in \mathcal{S}_2$ we have for some $R'_{\sigma'_r}$,

$$R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} R'_{\sigma'_r} \text{ and } (Q'_{\sigma'_q}, R'_{\sigma_r}) \in \mathcal{S}_2$$

Hence $(P'_{\sigma'_p}, R'_{\sigma'_r}) \in \mathcal{S}_1\mathcal{S}_2$. By similar reasoning if $R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} R'_{\sigma'_r}$ then we can find $P'_{\sigma'_p}$ such that $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and $(P'_{\sigma'_p}, R'_{\sigma'_r}) \in \mathcal{S}_1\mathcal{S}_2$. □

We are now ready to define strong secure equivalence.

Definition 7.10. P_{σ_p} and Q_{σ_q} are *strongly securely equivalent* or *strongly securely bisimilar*, written $P_{\sigma_p} \sim^s Q_{\sigma_q}$, if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ for some strong bisimulation $\mathcal{S}$. This may be equivalently expressed as follows:

$$\sim^s \; = \; \bigcup \{\mathcal{S} : \mathcal{S} \text{ is a strong secure bisimulation}\}$$

■

Proposition 7.5 introduced properties of bisimulation that allow us to bring the following properties of strong secure equivalence.

Proposition 7.6. *We say that*

1. *$\sim^s$ is the largest strong secure bisimulation.*
2. *$\sim^s$ is an equivalence relation.*

Proof. The first proposition is implied by Def 7.10. By Proposition 7.5(4), $\sim^s$ is a strong secure bisimulation and includes any other such.

To prove, that $\sim^s$ is an equivalence relation we have to prove:
Reflexivity: According to Proposition 7.5(1), for any P_{σ_p}: $(P_{\sigma_p}, P_{\sigma_p}) \in \mathcal{S}$, where $\mathcal{S}$ is a strong secure bisimulation. It implies $P_{\sigma_p} \sim^s P_{\sigma_p}$.
Symmetry: If $P_{\sigma_p} \sim^s Q_{\sigma_q}$, then $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ for some strong secure bisimulation $\mathcal{S}$. Hence $(Q_{\sigma_q}, P_{\sigma_p}) \in \mathcal{S}^{-1}$, and so $Q_{\sigma_q} \sim^s P_{\sigma_p}$ by Proposition 7.5(2).
Transitivity: If $P_{\sigma_p} \sim^s Q_{\sigma_q}$ and $Q_{\sigma_q} \sim^s R_{\sigma_R}$ then $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}_1$ and $(Q_{\sigma_q}, R_{\sigma_r}) \in \mathcal{S}_2$ for strong secure bisimulations $\mathcal{S}_1$, $\mathcal{S}_2$. So $(P_{\sigma_p}, R_{\sigma_r}) \in \mathcal{S}_1\mathcal{S}_2$, and so $P_{\sigma_p} \sim^s R_{\sigma_r}$ by Proposition 7.5(4). □

The definition 7.9 uses word 'imply'. This is not exactly what we want. We need a definition saying, anytime the conditions are satisfied, it is a strong secure bisimulation! We want to replace 'implies' with 'iff'. For the purpose, we first define a new relation $\sim'$.

Definition 7.11. $P_{\sigma_p} \sim' Q_{\sigma_q}$ iff, for all $\alpha_{\alpha_c} \in Act$,

1. Whenever $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ then, for some $Q'_{\sigma'_q}$, $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$
2. Whenever $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ then, for some P'_{σ_p}, $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$

Proposition 7.5(1) tells that $\sim^s$ is a strong bisimulation and we deduce from Def 7.9 and Def 7.11 that

$$P_{\sigma_p} \sim^s Q_{\sigma_q} \;\Rightarrow\; P_{\sigma_p} \sim' Q_{\sigma_q}$$

It remains to show the second direction of equivalence.

Lemma 7.7. *The relation $\sim'$ is a strong bisimulation.*

Proof. Let $P_{\sigma_p} \sim' Q_{\sigma_q}$ and let $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. We have to find Q' so that $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \sim' Q'_{\sigma'_q}$. And by the definition of $\sim'$ we can indeed find Q' so that $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$; also $P'_{\sigma'_p} \sim' Q'_{\sigma'_q}$ and we are done. □

At last we can replace implication by 'iff' for relation $\sim^s$:

Proposition 7.8. *$P_{\sigma_p} \sim^s Q_{\sigma_q}$ iff, for all $\alpha_{c_\alpha} \in Act$,*

1. *Whenever $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\to} P'_{\sigma'_p}$ then, for some $Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\to} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$*

2. *Whenever $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\to} Q'_{\sigma'_q}$ then, for some $P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\to} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$*

This procedure is also used to deduce the required definition of weak and observation bisimulation. We always start with the desired definition and with properties of bisimulations the correct definition is derived.

Proposition 7.9. *We have a secure system $\mathbb{S}$ with initial state P_{σ_p}. We have a system $\mathbb{T}$ with initial state Q_{σ_q}. If $P_{\sigma_p} \sim^s Q_{\sigma_q}$, then $\mathbb{T}$ is also secure.*

Proof. We prove it by induction through all states.
(1) $P_{\sigma_p} \sim^s Q_{\sigma_q}$ implies Q_{σ_q} to be secure by Def 7.9.
(2) when $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, then $\exists Q'_{\sigma'_q} : Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ for all actions α_{c_α} of P_{σ_p} and Q_{σ_q} and $P'_{\sigma'_p} \sim^s Q'_{\sigma'_q}$ □

We know that strong bisimulation preserves security of systems! With this assumption, we may state secure monoid laws.

Proposition 7.10. ***(Secure monoid laws)*** *We are ready to prove the following monoid laws.*
(1) $P_{\sigma_p} + Q_{\sigma_q} \sim^s Q_{\sigma_q} + P_{\sigma_p}$
(2) $P_{\sigma_p} + (Q_{\sigma_q} + R_{\sigma_r}) \sim^s (P_{\sigma_p} + Q_{\sigma_q}) + R_{\sigma_r}$
(3) $P_{\sigma_p} + P_{\sigma_p} \sim^s P_{\sigma_p}$
(4) $P_{\sigma_p} + \mathbf{0} \sim^s P_{\sigma_p}$

Proof. We shall use semantic rules $\mathbf{Sum}_j$ for all four laws.
When proving (2), we suppose that $P_{\sigma_p} + (Q_{\sigma_q} + R_{\sigma_r}) \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. Then by the semantic rules $\mathbf{Sum}_j$, either $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ or $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ or $R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. For each case we easily show that $(P_{\sigma_p} + Q_{\sigma_q}) + R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \sim^s P'_{\sigma'_p}$. The reverse direction is similar.
Laws (1) and (3) are proved similarly.
Law (4) is also simple. Assuming $P_{\sigma_p} + \mathbf{0} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, it is always $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ because $\mathbf{0}$ has no action. □

7.5 Bisimulation and Observation Equivalence

Previous section introduced notion of strong bisimulation. Now, we are going to relax the introduced definition. We shall not want exact match of agents' actions if they are τ_{c_i} actions, we merely require that each perfect action be matched by *zero or more* perfect actions.

The first issue we have to solve is to formulate new definition of secure action-sequence in order to compare action sequences. This new definition shall ignore τ actions. In the moment we start to consider classification of actions, there arises a problem with the original definition from process calculus. Assume the following secure action-sequences $t_{1,c_t^1} = a_{c_a^1}\tau_{c_1}\tau_{c_2}b_{c_b^1}$ and $t_{2,c_t^2} = a_{c_a^2}b_{c_b^2}$. When we ignore actions τ_{c_1} and τ_{c_2} in t_{1,c_t^1} we obtain new secure action-sequence $t'_{1,c_t^1} = a_{c_a^1}b_{c_b^1}$. It means that we ignore two actions τ_{c_1} and τ_{c_2} and also two states that probably accessed some resources. Classification of actions, however, can not be changed.

It means that classification of t'_1 still differs from classification of t_2 with high probability. We will see that it narrows possibility for bisimulation, but in a natural way that is correct when considering security of tasks. We may intuitively say that bisimulation may be found when ignoring parallel branches in tasks, but it is not possible to ignore some actions in sequences.

We may reason about possibility to arbitrary change classification of actions next to the neglected τ_{c_i} actions, but it would be inexcusable change of task definition that would violate security of the task and the whole system.

There are only two situations when classification of an action may *change*. But we show that classification stays, in fact, the same. It is classification of perfect actions during their decomposition and merging. Any perfect action is an abstraction for actions that are not important from our point of view, agents connected with τ_{c_i} actions are viewed as one closed entity. It means that we may replace sequence of $\tau_{c_i}, i \in I$ actions with empty sequence ϵ_c : $c = \triangle_{i\in I} c_i$. In the opposite direction, the empty sequence ϵ_c may by replaced with a sequence of actions $\tau_{c_i}, i \in I$, so that $c = \triangle_{i\in I} c_i$.

There are two situations that may occur when ignoring a whole sequence of perfect actions.

- Secure action-sequence of τ_{c_i} actions is followed by an action $a_{c_a} \in \mathcal{L}$.
- Secure action-sequence of τ_{c_i} is not followed by any other action for the given secure action-sequence.

Removing of τ_{c_i} actions in the latter case may be solved by approach from the previous paragraph. The former case may be solved by removing the whole sequence of τ_{c_i} because of the following theorem.

Theorem 7.11. *Removing subsequence $\alpha_{i,c_i} \dots \alpha_{j,c_j}, j < n$ from a secure action-sequence $t_{c_t} = \alpha_{1,c_1}\alpha_{2,c_2} \dots \alpha_{n,c_n}$ does not change classification t_{c_t} of the sequence as a whole.*

Proof. Having secure action-sequence $t_{c_t} = \alpha_{1,c_2}\alpha_{2,c_2} \dots \alpha_{n,c_n}$, there exist agents $P_{1,\sigma_1}, \dots, P_{n+1,\sigma_{n+1}}$ such that $P_{1,\sigma_1} \xrightarrow{\alpha_{1,c_1}} P_{2,\sigma_2} \xrightarrow{\alpha_{2,c_2}} \cdots \xrightarrow{\alpha_{n,c_n}} P_{n+1,\sigma_{n+1}}$. Definition of secure action says that $c_i = c_{i-1} \vartriangle \Gamma(P_{i,\sigma_i})$. It implies $c_i \leq c_j, \forall j \geq i$ (see Def 6.3). It again implies that combination of classifications of two consecutive actions $c_i = (sc_i, dc_i)$ and $c_i = (sc_{i+1}, dc_{i+1})$ is

$$c_i \vartriangle c_{i+1} = (max(sc_i, sc_{i+1}), dc_i \cup dc_{i+1}) = (sc_{i+1}, dc_{i+1}) = c_{i+1}$$

The classification of the secure action-sequence c_t is equal to classification of the last action.

$$c_t = c_1 \vartriangle \dots \vartriangle c_n = c_2 \vartriangle \dots \vartriangle c_n = \cdots = c_{n-1} \vartriangle c_n = c_n$$

It means that removing of any subsequence from t_{c_t} that does not contain the last action α_{n,c_n} does not change classification $c_t = c_n$ of the secure action-sequence. $\square$

Previous theorem is based on the assumption that classification of actions in defined task/forkflow can not be changed. We assume that it is more secure not to change security properties of already defined tasks. In fact, it would be very difficult to change properties of tasks defined on particular s-nodes.

We are moving toward *secure bisimulation*. We need to change definition for removing perfect actions from secure action-sequences. The original definition of CCS is insufficient because it changes classification of the result.

Definition 7.12. If $t_{c_t} = \alpha_{1,c_1} \dots \alpha_{n,c_n} = t'_{c'_t}\alpha_{n,c_n} \in Act^*$ is a secure action-sequence with classification, $c_t = \triangle_{i=1..n} c_i = c_n$, then:

1. if $\alpha_{n,c_n} \in \mathcal{L}$, $\widehat{t}_{c_{\widehat{t}}} = \alpha_{1,c_1} \dots \alpha_{m,c_m}\alpha_{n,c_n} \in \mathcal{L}^*$
2. if α_{n,c_n} is a perfect action, $\widehat{t}_{c_{\widehat{t}}} = \alpha_{1,c_1} \dots \alpha_{m,c_m}\epsilon_{c_n} \in \mathcal{L}^*$

is the sequence gained by deleting all occurrences of perfect actions from $t'_{c'_t}$, with $c_{\widehat{t}} = \triangle_{i=1..m,n} c_i = c_n$. $\blacksquare$

Here, we can sharply see one of the basic differences between original process calculus and our modification. We still assume perfect actions as only one symbol (assuming its visible effects), but we must be careful with its security classification. Although the action is not 'visible', it still holds important security information.

Note that $\widehat{\tau^n_{c_{i\in I}}} = \epsilon_c$ with $c = \triangle_{i \in I} c_i$ (the empty sequence). It is an abstraction of data flow from P_{σ_p} to itself. But because it represents some existing secure action-sequence we have to look at its classification. We shall write secure sequence of perfect actions as $(\tau_{\_})^*$ or $\tau^n_{\_}$.

Theorem 7.12. *Classification of secure action-sequence does not change with removing of perfect actions:*

$$t_{c_t} = \widehat{s}_{c_s} \Rightarrow c_t = c_s$$

Proof. Proof results from Theorem 7.11 that says that classification of action-sequence does not change when removing any but last action or sequence of actions. Assuming $s_{c_s} = \alpha_{1,c_1} \dots \alpha_{n,c_n}$, there are two cases in Def 7.12.
1. $\alpha_{n,c_n} \in \mathcal{L}$ - we are not removing the last action.
2. $\alpha_{n,c_n} \in \tau_*$ - α_{n,c_n} is replaced with ε_{c_n} and we are not removing the last action - empty action. □

Definition 7.13. If $t_{c_t} = \alpha_{1,c_1} \dots \alpha_{n,c_n} \in Act^*$, then we write $E_{\sigma_1} \xrightarrow{t_{c_t}} E'_{\sigma_2}$ if $E_{\sigma_1} \xrightarrow{\alpha_{1,c_1}} \cdots \xrightarrow{\alpha_{n,c_n}} E'_{\sigma_2}$. We shall also write $E_{\sigma_1} \xrightarrow{t_{c_t}}$ to mean that $E_{\sigma_1} \xrightarrow{t_{c_t}} E'_{\sigma_2}$ for some E'_{σ_2}. ■

We can define a new labelled transition system

$$(\mathcal{E}, \mathcal{L}^*, \{\overset{s}{\Rightarrow} : s \in \mathcal{L}^*\})$$

over agent expressions, in which transition relations $\overset{s}{\Rightarrow}$ are defined as follows.

Definition 7.14. If $t_{c_t} = \alpha_{1,c_1} \dots \alpha_{n,c_n} \in Act^*$, then $E_{\sigma_1} \overset{t_{c_t}}{\Longrightarrow} E'_{\sigma_2}$ if

$$E_{\sigma_1} (\xrightarrow{\tau_{\_}})^* \xrightarrow{\alpha_{1,c_1}} (\xrightarrow{\tau_{\_}})^* \cdots (\xrightarrow{\tau_{\_}})^* \xrightarrow{\alpha_{n,c_n}} (\xrightarrow{\tau_{\_}})^* E'_{\sigma_2}$$

We should notice that $c_n \leq c_t$, but it is equal to the classification of the last action in the unwrapping (perfect action or α_{c_α}).
We shall also write $E_{\sigma_1} \overset{t_{c_t}}{\Longrightarrow}$ to mean that $E_{\sigma_1} \overset{t_{c_t}}{\Longrightarrow} E'_{\sigma_2}$ for some E'. ■

Next notion we are going to introduce is analogous to secure derivative.

Definition 7.15. (Descendant) If $t_{c_t} \in Act^*$, then $E'_{\sigma'}$ is a *secure* t_{c_t}*-descendant* of E_σ iff $E_\sigma \overset{\widehat{t}_{c_t}}{\Longrightarrow} E'_{\sigma'}$. ■

Note that $t_{c_t} \in \mathcal{L}^*$ just means that $E_\sigma \overset{t_{c_t}}{\Longrightarrow} E'_{\sigma'}$ and if $E'_{\sigma'}$ is τ_{c_t}-descendant, $E \overset{\tau^n}{\Longrightarrow} E'_{\sigma'}$ $E'_{\sigma'} \equiv E_\sigma$. If $n = 0$, then $E'_{\sigma'} \equiv E_\sigma$. Also note that $\widehat{\alpha}_{c_\alpha}$ does not mean that classification of α is c_α. It is classification of the last action in the 'unwrapping'.

In the moment, we have defined three relations between agents. Each specifies an action-sequence with exactly the same observable content as t, but the possibilities for intervening perfect actions differ.

$\xrightarrow{t_{c_t}}$ specifies exactly actions occurring in t_{c_t}.

$\overset{t_{c_t}}{\Longrightarrow}$ specifies at least perfect actions occurring in t_{c_t}.

$\overset{\widehat{t}_{c_t}}{\Longrightarrow}$ specifies nothing about the perfect actions occurring in t_{c_t}.

We are now going to introduce a notion of *observation equivalence.*

> We say that P_{σ_p} and Q_{σ_q} are observation-equivalent iff, for every action α_{c_α}, every α_{c_α}-derivative of P_{σ_p} is observation-equivalent to some α_{c_α}-descendant of Q_{σ_q}, and similarly with P_{σ_p} and Q_{σ_q} interchanged.

Definition 7.16. (Secure bisimulation) A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over secure agents is a *(weak) secure bisimulation* if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ implies, for all $\alpha_{c_\alpha} \in Act$,

1. Whenever $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ then, $\exists\, Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$

2. Whenever $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ then, $\exists\, P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$

And the following definition is about bisimilarity.

Definition 7.17. P_{σ_p} and Q_{σ_q} are *securely observation-equivalent* or *(weakly) bisimilar*, written $P_{\sigma_p} \approx^s Q_{\sigma_q}$, if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ for some (weak) bisimulation $\mathcal{S}$. That is

$$\approx^s = \bigcup\{\mathcal{S} : \mathcal{S} \text{ is a bisimulation}\}$$

■

7.5.1 Basic Properties of Bisimilarity

The basic properties of secure bisimilarity are analogous to the strong secure bisimilarity.

Proposition 7.13. *Assume that each $\mathcal{S}_i = (i = 1, 2, \ldots)$ is a secure bisimulation. Then the following are all secure bisimulations:*

(1) Id_p (3) $\mathcal{S}_1\mathcal{S}_2$

(2) $\mathcal{S}_i^{-1}$ (4) $\bigcup_{i \in I} \mathcal{S}_i$

Proof. Similar to Proposition 7.5; in (3), we also need the auxiliary result that if $(Q_{\sigma_q}, R_{\sigma_r}) \in \mathcal{S}_i$ and $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ then, for some R',$R_{\sigma_r} \overset{\widehat{\alpha}_{c_\alpha}}{\Rightarrow} R'_{\sigma'_r}$ and $(Q'_{\sigma'_q}, R'_{\sigma'_r}) \in \mathcal{S}_i$. □

Proposition 7.14. *We say that*

1. $\approx^s$ is the largest secure bisimulation.

2. $\approx^s$ is an equivalence relation.

Proof. Based on Proposition 7.13, just as Proposition 7.6 is based on Proposition 7.5. □

We continue the analogy by showing that $\approx^s$ satisfies property (*) of strong secure bisimulation. We define a new relation $\approx'$ in terms of $\approx^s$.

Definition 7.18. $P_{\sigma_p} \approx' Q_{\sigma_q}$ iff, for all $\alpha_{c_\alpha} \in Act$,

1. Whenever $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ then, $\exists\, Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$

2. Whenever $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ then, $\exists\, P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$

Proposition 7.14 (1) says that $\approx^s$ is a secure bisimulation. We deduce from definitions 7.16 and 7.18 that $P_{\sigma_p} \approx^s Q_{\sigma_q}$ implies $P_{\sigma_p} \approx' Q_{\sigma_q}$. We only have to show that $P_{\sigma_p} \approx' Q_{\sigma_q}$ implies $P_{\sigma_p} \approx^s Q_{\sigma_q}$.

Lemma 7.15. *The relation $\approx'$ is a weak secure bisimulation.*

Proof. Analogous to Lemma 7.7. □

We are now ready to state the following proposition - final version of secure bisimulation.

Proposition 7.16. *$P_{\sigma_p} \approx^s Q_{\sigma_q}$ iff, for all $\alpha_{c_\alpha} \in Act$,*

1. *Whenever $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ then, $\exists\, Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$*

2. *Whenever $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ then, $\exists\, P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$*

We would like to show that bisimulation preserves security as well as strong bisimulation.

Proposition 7.17. *Assume, there is a secure system $\mathbb{S}$ with initial state P_{σ_p}. There is also a system $\mathbb{T}$ with initial state Q_{σ_q} such that $P_{\sigma_p} \approx^s Q_{\sigma_q}$. The system $\mathbb{T}$ is then also secure.*

Proof. We prove it by induction through all states.
(1) $P_{\sigma_p} \approx^s Q_{\sigma_q}$ implies Q_{σ_q} to be secure by Def 7.16.
(2) when $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, then $\exists Q'_{\sigma'_q} :\ Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ for all actions α_{c_α} of P_{σ_p} and all action-sequences $\widehat{\alpha}_{c_\alpha}$ of Q_{σ_q} and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$ □

We conclude our analogy by introducing the notion of bisimulation up to $\approx^s$. This is even more important than *strong* bisimulation. In realistic systems we find many states which are bisimilar, but not identical. We often write $P_{\sigma_p}\mathcal{R}\, Q_{\sigma_q}$ to mean $(P, Q) \in \mathcal{R}$, for any binary relation $\mathcal{R}$. Note that $\approx^s \mathcal{S} \approx^s$ is a composition of binary relations such that $P_{\sigma_p} \approx^s \mathcal{S} \approx^s Q_{\sigma_q}$ means that for some $P'_{\sigma'_p}$ and $Q'_{\sigma'_q}$ we have $P_{\sigma_p} \approx^s P'_{\sigma'_p}$, $P'_{\sigma'_p}\mathcal{S}\, Q'_{\sigma'_q}$ and $Q'_{\sigma'_q} \approx^s Q_{\sigma_q}$.

Definition 7.19. $\mathcal{S}$ is a *(weak) secure bisimulation up to* $\approx^s$ if $P_{\sigma_p}\mathcal{S}\ Q_{\sigma_q}$ implies, for all α_{c_α},

1. Whenever $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ then, $\exists\, Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \approx^s \mathcal{S} \approx^s Q'_{\sigma'_q}$

2. Whenever $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ then, $\exists\, P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \approx^s \mathcal{S} \approx^s Q'_{\sigma'_q}$

Lemma 7.18. *If $\mathcal{S}$ is a secure bisimulation up to $\approx^s$, then $\approx^s \mathcal{S} \approx^s$ is a secure bisimulation.*

Proof. Let $P_{\sigma_p} \approx^s \mathcal{S} \approx^s Q_{\sigma_q}$ and $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. It is enough to show that $P'_{\sigma'_p} \approx^s \mathcal{S} \approx^s Q'_{\sigma'_q}$.
First note that for some $P_{1,\sigma_{p_1}}$ and $Q_{1,\sigma_{q_1}}$, $P_{\sigma_p} \approx^s P_{1,\sigma_{p_1}}\mathcal{S}\, Q_{1,\sigma_{q_1}} \approx^s Q_{\sigma_q}$, further $\widehat{\widehat{\alpha}}_{c_\alpha} = \widehat{\alpha}_{c_\alpha}$ (by definition). Now, we can fill the following three equations from the definition of secure bisimulation:

1. $P_{\sigma_p} \approx^s P_{1,\sigma_{p_1}} \wedge P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \Rightarrow \exists\, P'_{1,\sigma'_{p_1}} : P'_{\sigma'_p} \approx^s P'_{1,\sigma'_{p_1}} \wedge P_{1,\sigma_{p_1}} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{1,\sigma'_{p_1}}$ from definition of bisimulation.

2. $P_{1,\sigma_{p_1}} \mathcal{S}\, Q_{1,\sigma_{q_1}} \wedge P_{1,\sigma_{p_1}} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} P'_{1,\sigma'_{p_1}} \Rightarrow \exists\, Q_{1,\sigma_{q_1}} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{1,\sigma'_{q_1}} \wedge$
$\wedge\, P'_{1,\sigma'_{p_1}} \approx^s \mathcal{S} \approx^s Q'_{1,\sigma'_{q_1}}$ from definition of bisimulation up to $\approx^s$.

3. $Q_{1,\sigma_{q_1}} \approx^s Q_{\sigma_q} \wedge Q_{1,\sigma_{q1}} \xrightarrow{\widehat{\alpha}_{c_\alpha}} Q'_{1,\sigma'_{q_1}} \Rightarrow \exists\, Q'_{\sigma'_q} : Q'_{1,\sigma'_{q_1}} \approx^s Q'_{\sigma'_q} \wedge Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ from definition of bisimulation.

What we have to do is to put all three sets of equations together and use transitivity of bisimulation.

$$P_{\sigma_p} \approx^s P_{1,\sigma_{p_1}} \mathcal{S}\, Q_{1,\sigma_{q_1}} \approx^s Q_{\sigma_q} \;\wedge\; P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p} \text{ implies}$$
$$P'_{\sigma'_p} \approx^s P'_{1,\sigma'_{p_1}} \approx^s \mathcal{S} \approx^s Q'_{1,\sigma'_{q_1}} \approx^s Q'_{\sigma'_q} = P'_{\sigma'_p} \approx^s \mathcal{S} \approx^s Q'_{\sigma'_q} \wedge$$
$$\wedge\; Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$$

□

Proposition 7.19. *If $\mathcal{S}$ is a secure bisimulation up to $\approx^s$ then $\mathcal{S} \subseteq \approx^s$.*

Proof. Since by Lemma 7.18 $\approx^s \mathcal{S} \approx^s$ is a bisimulation, we have got that $\approx^s \mathcal{S} \approx^s \subseteq \approx^s$ by definition of $\approx^s$. But $\mathcal{S} \subseteq \approx^s \mathcal{S} \approx^s$, so we are done. □

It means that to prove $P_{\sigma_p} \approx^s Q_{\sigma_q}$, we have to find a bisimulation up to $\approx^s$ which contains $(P_{\sigma_p}, Q_{\sigma_q})$

7.5.2 Further Properties of Bisimilarity

At this place, I want to introduce just one property of bisimilarity that shows difference from strong bisimilarity.

Proposition 7.20. $\varepsilon_c.P_{\sigma_p} \approx^s \tau_c.P_{\sigma_p}$

Proof. We use $\approx'$ that has been shown identical with $\approx^s$.
Consider any action of P_{σ_p} such that $\varepsilon_c.P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. $\tau_c.P_{\sigma_p} \xrightarrow{\tau_c} P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, so $\tau_c.P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$.
For the second direction consider $\tau_c.P_{\sigma_p} \xrightarrow{\tau_c} P_{\sigma_p}$ is matched by the null action $\varepsilon_c.P_{\sigma_p} \xrightarrow{\varepsilon_c} P_{\sigma_p}$. So $\varepsilon_c.P_{\sigma_p} \approx' \tau_c.P_{\sigma_p}$ and also $\varepsilon_c.P_{\sigma_p} \approx^s \tau_c.P_{\sigma_p}$. □

7.6 The Theory of Observation Congruence

There is given another definition of bisimulation. We prove that all the basic combinators (except Summation) preserve bisimilarity. The definition of equality is given that allows to prove all propositions we shall state.

We begin by introducing another characterization of bisimulation, in terms of the notion of *experiment*. We consider $P_{\sigma_p} \overset{s_{c_s}}{\Longrightarrow} P'_{\sigma'_p}$, where $s_{c_s} \in \mathcal{L}^*$ to be an experiment upon P_{σ_p}. It means that we run P_{σ_p} and observe the action sequence $s_{c_s} = \ell_{1,c_1} \cdots \ell_{n,c_n} \varepsilon_{c_\tau}$. If $s_{c_s} = \varepsilon_{c_\tau}$, we also write $P_{\sigma_p} \overset{\varepsilon_{c_\tau}}{\Longrightarrow} P'_{\sigma'_p}$. It may involve one or more (unobserved) τ_{c_i} actions ($c_\tau = \triangle_{i\in I} c_i$). It is, however, still an experiment - $P'_{\sigma'_p}$ may admit fewer experiments than P_{σ_p} does.

Proposition 7.21. *$\mathcal{S}$ is a secure bisimulation iff, for all $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ and $s_{c_s} \in \mathcal{L}^*$,*

1. *Whenever $P_{\sigma_p} \overset{s_{c_s}}{\Longrightarrow} P'_{\sigma'_p}$ then, $\exists Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{s_{c_s}}{\Longrightarrow} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \mathcal{S}\, Q'_{\sigma'_q}$.*
2. *Whenever $Q_{\sigma_q} \overset{s_{c_s}}{\Longrightarrow} Q'_{\sigma'_q}$ then, $\exists P'_{\sigma'_p}$, $P_{\sigma_p} \overset{s_{c_s}}{\Longrightarrow} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \mathcal{S}\, Q'_{\sigma'_q}$.*

Proof. ($\Longrightarrow$) Assume that $\mathcal{S}$ is a bisimulation. Let $P_{\sigma_p} \overset{s_{c_s}}{\Longrightarrow} P'_{\sigma'_p}$, so that $P_{\sigma_p} \xrightarrow{t_{c_t}} P'_{\sigma'_p}$, where $t_{c_t} = \alpha_{1,c_1} \ldots \alpha_{n,c_n} \in Act^*$ and $\widehat{t}_{c_t} = s_{c_s}, c_t = c_s$. Then $P_{\sigma_p} \xrightarrow{\alpha_{1,c_1}} P_{1,\sigma_p^1} \xrightarrow{\alpha_{2,c_2}} \cdots \xrightarrow{\alpha_{n,c_n}} P'_{\sigma'_p}$, and since $\mathcal{S}$ is a bisimulation we also have $Q_{\sigma_q} \overset{\widehat{\alpha}_{1,c_1}}{\Longrightarrow} Q_{1,\sigma_q^1} \overset{\widehat{\alpha}_{2,c_2}}{\Longrightarrow} \cdots \overset{\widehat{\alpha}_{n,c_n}}{\Longrightarrow} Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \mathcal{S}\, Q'_{\sigma'_q}$. Hence $Q_{\sigma_q} \overset{\widehat{t}_{c_t}}{\Longrightarrow} Q'_{\sigma'_q}$, i.e. $Q_{\sigma_q} \overset{s_{c_s}}{\Longrightarrow} Q'_{\sigma'_q}$.

($\Longleftarrow$) Assuming (1), let $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. If $\alpha_{c_\alpha} = \tau_{c_\alpha}$ then $P_{\sigma_p} \overset{\varepsilon_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$, so by (1) $Q_{\sigma_q} \overset{\varepsilon_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \mathcal{S}\, Q'_{\sigma'_q}$. But $\widehat{\alpha}_{c_\alpha} = \tau_{c_\alpha}$, so $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ as required. If $\alpha_{c_\alpha} = \ell_{c_\alpha}$ then $P_{\sigma_p} \overset{\ell_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and by (1) $Q_{\sigma_q} \overset{\ell_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \mathcal{S}\, Q'_{\sigma'_q}$. But $\widehat{\ell}_{c_\alpha} = \ell_{c_\alpha}$, so $Q_{\sigma_q} \overset{\widehat{\ell}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ as required. Similarly for P_{σ_p} and Q_{σ_q} interchanged. □

Without assuming the empty experiment $P_{\sigma_p} \overset{\varepsilon_\tau}{\Longrightarrow} P'_{\sigma'_p}$, one could easily show that $a.\mathbf{0} + \tau.\mathbf{0}$ is bisimilar with $a.\mathbf{0}$, which is false by the original definition and bisimilarity would not be preserved by Prefix and Composition.

Proposition 7.22. *Secure bisimilarity is preserved by Composition, Restriction and Relabelling. That is, if $P_{\sigma_p} \approx^s Q_{\sigma_q}$ than $P_{\sigma_p}|R_{\sigma_r} \approx^s Q_{\sigma_q}|R_{\sigma_r}$, $P_{\sigma_p}\backslash L \approx^s Q_{\sigma_q}\backslash L$ and $P_{\sigma_p}[f] \approx^s Q_{\sigma_q}[f]$.*

Proof. Let's start with Composition. It will be enough to show that

$$\mathcal{S} = \{(P_{\sigma_p}|R_{\sigma_r}, Q_{\sigma_q}|R_{\sigma_r}) :\ P_{\sigma_p}, Q_{\sigma_q}, R_{\sigma_r} \in \mathcal{P} \wedge P_{\sigma_p} \approx^s Q_{\sigma_q}\}$$

is a bisimulation. So let $P_{\sigma_p}|R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}|R'_{\sigma'_r}$ (every derivative must have this form).

Case 1: $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and $R_{\sigma_r} \equiv R'_{\sigma'_r}$. Then $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ for some $Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$, so $Q_{\sigma_q}|R_{\sigma_r} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}|R_{\sigma_r}$, and $(P'_{\sigma'_p}|R_{\sigma_r}, Q'_{\sigma'_q}|R_{\sigma_r}) \in \mathcal{S}$ as required.

Case 2: $R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} R'_{\sigma'_r}$ and $P_{\sigma_p} \equiv P'_{\sigma'_p}$. Then also $Q_{\sigma_q}|R_{\sigma_r} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q_{\sigma_q}|R'_{\sigma'_r}$, and $(P_{\sigma_p}|R'_{\sigma'_r}, Q_{\sigma_q}|R'_{\sigma'_r}) \in \mathcal{S}$ as required.

Case 3: $\alpha_{c_\alpha} = \tau_{c_\alpha}$, $P_{\sigma_p} \xrightarrow{\ell_{c_\alpha}} P'_{\sigma'_p}$ and $R_{\sigma_r} \xrightarrow{\overline{\ell}_{c_\alpha}} R'_{\sigma'_r}$. Then $Q_{\sigma_q} \overset{\ell_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ for some $Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$, so $Q_{\sigma_q}|R_{\sigma_r} \overset{\tau_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}|R'_{\sigma'_r}$, and $(P'_{\sigma'_p}|R'_{\sigma'_r}, Q'_{\sigma'_q}|R'_{\sigma'_r}) \in \mathcal{S}$ as required. Hence by symmetric argument, $\mathcal{S}$ is a bisimulation.

Restriction $\mathcal{S} = \{(P_{\sigma_p}\backslash L, Q_{\sigma_q}\backslash L) :\ P_{\sigma_p}, Q_{\sigma_q} \in \mathcal{P}\ \wedge\ P_{\sigma_p} \approx^s Q_{\sigma_q}\}$
If $\alpha_{c_\alpha} \in L$ then there is no derivative $P'_{\sigma'_p}$. Otherwise, for $\alpha_{c_\alpha} \notin L$ we obtain $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$. It implies $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ for some $Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$, so $Q_{\sigma_q}\backslash L \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}\backslash L$, and $(P'_{\sigma'_p}\backslash L, Q'_{\sigma'_q}\backslash L) \in \mathcal{S}$ as required.

Relabelling $\mathcal{S} = \{(P_{\sigma_p}[f], Q_{\sigma_q}[f]) :\ P_{\sigma_p}, Q_{\sigma_q} \in \mathcal{P}\ \wedge\ P_{\sigma_p} \approx^s Q_{\sigma_q}\}$
$P_{\sigma_p}[f] \xrightarrow{f(\alpha_{c_\alpha})} P'_{\sigma'_p}[f]$. It implies $Q_{\sigma_q}[f] \overset{\widehat{f(\alpha_{c_\alpha})}}{\Longrightarrow} Q'_{\sigma'_q}[f]$ for some $Q'_{\sigma'_q}$ with $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$, so $Q_{\sigma_q}[f] \overset{\widehat{f(\alpha_{c_\alpha})}}{\Longrightarrow} Q'_{\sigma'_q}[f]$, and $(P'_{\sigma'_p}[f], Q'_{\sigma'_q}[f]) \in \mathcal{S}$ as required.
When using symmetry for relabelling and restriction we obtain that $\mathcal{S}$ is in both cases bisimulation. □

We have seen that $\approx^s$ is not fully substitutive ($P_{\sigma_p} \approx^s Q_{\sigma_q}$ does not imply $P_{\sigma_p} + R_{\sigma_r} \approx^s Q_{\sigma_q} + R_{\sigma_r}$). We want a notion of secure equality, $P_{\sigma_p} =^s Q_{\sigma_q}$, which implies $P_{\sigma_p} \approx^s Q_{\sigma_q}$ and which *is* fully substitutive - congruence relation.

Definition 7.20. (Secure equality) P_{σ_p} and Q_{σ_q} are *securely equal* or *securely (observation-) congruent*, written $P_{\sigma_p} =^s Q_{\sigma_q}$, if for all α_{c_α}

1. Whenever $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$, then $\exists\ Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$.

2. Whenever $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$, then $\exists\ P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ and $P'_{\sigma'_p} \approx^s Q'_{\sigma'_q}$. ■

The difference from secure bisimilarity is that first actions of P_{σ_p} and Q_{σ_q} must be matched by *at least* one action of the other.

Proposition 7.23. ***(Secure τ laws)***
(1) $\alpha_{c_\alpha}.\tau_c.P_{\sigma_p} =^s \alpha_{c_\alpha}.\varepsilon_c.P_{\sigma_p}$
(2) $P_{\sigma_p} + \tau_c.P_{\sigma_p} =^s \tau_c.P_{\sigma_p}$
(3) $\alpha_{c_\alpha}.(P_{\sigma_p} + \tau_c.Q_{\sigma_q}) + \alpha_{c_\alpha}.\varepsilon_c.Q_{\sigma_q} =^s \alpha_{c_\alpha}.(P_{\sigma_p} + \tau_c.Q_{\sigma_q})$

Proof. Proof is done directly from the definition of secure equality. We also use Proposition 7.20 for equation (1). □

We may add null action ε_c without *visible* change of systems. Systems stay secure according to the left sides of equations.

Proposition 7.24. *Assume that $\mathcal{L}(P_{\sigma_p}) \cup \mathcal{L}(Q_{\sigma_q}) \neq \mathcal{L}$. Then $P_{\sigma_p} =^s Q_{\sigma_q}$ iff, for all R_{σ_r}, $P_{\sigma_p} + R_{\sigma_r} \approx^s Q_{\sigma_q} + R_{\sigma_r}$.*

Proof. ($\Longrightarrow$) It is easy to show that the following is a bisimulation:

$$\{(P_{\sigma_p}+R_{\sigma_r}, Q_{\sigma_q}+R_{\sigma_r}) :\ P_{\sigma_p}, Q_{\sigma_q}, R_{\sigma_r} \in \mathcal{P} \wedge P_{\sigma_p} =^s Q_{\sigma_q}\}$$

Case 1: $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and $R_{\sigma_r} \equiv R'_{\sigma'_r}$. Then $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ (see that $Q_{\sigma_q}! \equiv Q'_{\sigma'_q}$) and $Q_{\sigma_q} + R_{\sigma_r} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$
Case 2: With symmetry for $R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} R'_{\sigma'_r}$ is obtained $(R'_{\sigma'_r}, R'_{\sigma'_r}) \in \mathcal{S}$

($\Longleftarrow$) We prove the contrapositive. Suppose that $P_{\sigma_p} \neq^s Q_{\sigma_q}$. Then there are e.g. α_{c_α} and $P'_{\sigma'_p}$ such that $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, but whenever $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ then $P'_{\sigma'_p}! \not\approx^s Q'_{\sigma'_q}$. Now choose $R_{\sigma_r} \equiv \ell_{c_\ell}.\mathbf{0}$, where ℓ_{c_ℓ} is not in the sort of P_{σ_p} nor Q_{σ_q}. Clearly $P_{\sigma_p} + R_{\sigma_r} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ so we must show that whenever $Q_{\sigma_q} + R_{\sigma_r} \xrightarrow{\widehat{\alpha}_{c_\alpha}} Q'_{\sigma'_q}$ then $P'_{\sigma'_p}! \not\approx^s Q'_{\sigma'_q}$. If $\alpha_{c_\alpha} = \tau_{c_\alpha}$ and $Q'_{\sigma'_q} \equiv Q_{\sigma_q} + R_{\sigma_r}$, then $P'_{\sigma'_p}! \not\approx^s Q'_{\sigma'_q}$ since $Q'_{\sigma'_q}$ has a ℓ_{c_ℓ} action and $P'_{\sigma'_p}$ has none. Otherwise $Q_{\sigma_q} + R_{\sigma_r} \overset{\alpha_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ and hence $Q_{\sigma_q} \overset{\widehat{\alpha}_{c_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ so again $P'_{\sigma'_p}! \not\approx^s Q'_{\sigma'_q}$. □

We can now show that secure equality lies between strong secure bisimilarity and secure bisimilarity.

Proposition 7.25. *$P_{\sigma_p} \sim^s Q_{\sigma_q}$ implies $P_{\sigma_p} =^s Q_{\sigma_q}$, and $P_{\sigma_p} =^s Q_{\sigma_q}$ implies $P_{\sigma_p} \approx^s Q_{\sigma_q}$.*

Proof. $P_{\sigma_p} \sim^s Q_{\sigma_q}$ implies $P_{\sigma_p} \approx^s Q_{\sigma_q}$ can be checked directly from definitions. To see that $P_{\sigma_p} \sim^s Q_{\sigma_q}$ implies $P_{\sigma_p} =^s Q_{\sigma_q}$ use property (*) of strong bisimilarity together with the definition of equality. Further, we have that $P_{\sigma_p} =^s Q_{\sigma_q}$ implies $P_{\sigma_p} + \mathbf{0} \approx^s Q_{\sigma_q} + \mathbf{0}$, but $P_{\sigma_p} + \mathbf{0} \sim^s P_{\sigma_p}$ so $P_{\sigma_p} + \mathbf{0} \approx^s P_{\sigma_p}$ and $P_{\sigma_p} \approx^s Q_{\sigma_q}$ follows. □

7.6.1 Axioms for Finite Agents

Definition 7.21. (Finite agent) A secure agent expression is *finite* if it contains only secure finite Summations and no Constants (or Recursions). ■

Definition 7.22. (Serial agent) A secure agent expression E is *serial* if it contains no Composition, Restriction or Relabelling, and also the defining equation of any secure Constant in E contains no Composition, Restriction or Relabelling. ■

Thus by expansion every secure finite agent can be equated to a finite secure serial agent. The only laws we have that involve only Summation (including **0**, the empty sum) and Prefix are those of Propositions 3.2 and 3.3. We therefore consider two axiom systems:

Axioms $\mathcal{A}_1$

A1 $P_{\sigma_p} + Q_{\sigma_q} =^s Q_{\sigma_q} + P_{\sigma_p}$

A2 $P_{\sigma_p} + (Q_{\sigma_q} + R_{\sigma_r}) =^s (P_{\sigma_p} + Q_{\sigma_q}) + R_{\sigma_r}$

A3 $P_{\sigma_p} + P_{\sigma_p} =^s P_{\sigma_p}$

A4 $P_{\sigma_p} + \mathbf{0} =^s P_{\sigma_p}$

The second axiom system contains also the τ laws:

Axioms $\mathcal{A}_2$

A1-A4

A5 $\alpha_{c_\alpha}.\tau_{c_1}.P_{\sigma_p} =^s \alpha_{c_\alpha}.\varepsilon_{c_1}.P_{\sigma_p} =^s \alpha_{c_\alpha}.P_{\sigma_p}$

A6 $P_{\sigma_p} + \tau_{c_1}.P_{\sigma_p} =^s \tau_{c_1}.P_{\sigma_p}$

A7 $\alpha_{c_\alpha}.(P_{\sigma_p} + \tau_{c_1}.Q_{\sigma_q}) + \alpha_{c_\alpha}.Q_{\sigma_q} =^s \alpha_{c_\alpha}.(P_{\sigma_p} + \tau_{c_1}.Q_{\sigma_q})$

For the following text, we shall use $P_{\sigma_p} =^s Q_{\sigma_q}$ to mean that P_{σ_p} and Q_{σ_q} are equal by our definition (Def 7.20), while $\mathcal{A} \vdash P_{\sigma_p} =^s Q_{\sigma_q}$ shall mean that the equality can be proved by equational reasoning from axioms $\mathcal{A}$.

Definition 7.23. P_{σ_p} is a *standard form*, or is *in standard form*, if

$$P_{\sigma_p} \equiv \sum_{i=1}^{m} \alpha_{i,c_i}.P_{i,\sigma_i} \tag{7.1}$$

where each P_{i,σ_i} is also in standard form. ■

Two facts: **0** is a standard form, by taking $m = 0$ and the order and grouping of terms in $\alpha_{1,c_1}.P_{1,\sigma_1} + \cdots + \alpha_{n,c_n}.P_{n,\sigma_n}$ may be ignored by virtue of axioms **A1**, **A2**.

Lemma 7.26. *For any P_{σ_p}, there is a standard form $P'_{\sigma'_p}$ such that*

$$\mathcal{A} \vdash P_{\sigma_p} =^s P'_{\sigma'_p}$$

Proof. By use of $\mathcal{A}_1$, the term $\mathbf{0}$ may be eliminated from any Summation $\cdots + \mathbf{0} + \cdots$, and this results in a standard form. □

The first result shows that axioms $\mathcal{A}_1$ are both sound and complete for strong congruence.

Proposition 7.27. $P_{\sigma_p} \sim^s Q_{\sigma_q}$ *iff* $\mathcal{A}_1 \vdash P_{\sigma_p} = Q_{\sigma_q}$

Proof. ($\Longleftarrow$) All we need to observe is that all of the axioms **A1**-**A4** are true if we replace $=^s$ by $\sim^s$ - see Prop 7.10.
($\Longrightarrow$) This is the completeness part. Assume $P_{\sigma_p} \sim^s Q_{\sigma_q}$ and also (according to Lemma 7.26) that P_{σ_p} and Q_{σ_q} are both in standard form ($P_{\sigma_p} \equiv \sum_{i=1}^{m} \alpha_{i,c_i}.P_{i,\sigma_i}$ and $Q_{\sigma_q} \equiv \sum_{j=1}^{n} \alpha_{j,c_j}.P_{j,\sigma_j}$). We prove the result by induction on the maximum depth of P_{σ_p} and Q_{σ_q}, where the depth of P_{σ_p} is defined to be the maximum number of nested Prefixes in P_{σ_p}.
If the maximum depth is 0 then P_{σ_p} and Q_{σ_q} are both $\mathbf{0}$ and equational reasoning ensures that $\mathcal{A}_1 \vdash \mathbf{0} = \mathbf{0}$.
Otherwise, let $\alpha_{c_\alpha}.P'_{\sigma'_p}$ be a summand of P_{σ_p}. Then $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$, so since $P_{\sigma_p} \sim^s Q_{\sigma_q}$ there is some $Q'_{\sigma'_q}$ such that $\mathcal{A}_1 \vdash Q'_{\sigma'_q} \sim^s P'_{\sigma'_p}$. But Q_{σ_q} is a standard form, so $\alpha_{c_\alpha}.Q'_{\sigma'_q}$ is a summand of Q_{σ_q}, and by induction $\mathcal{A}_1 \vdash P'_{\sigma'_p} =^s Q'_{\sigma'_q}$, so that summand $\alpha_{c_\alpha}.P'_{\sigma'_p}$ of P_{σ_p} can be proved equal to a summand of Q_{σ_q}. Similarly, every summand $\beta_{c_\beta}.Q'_{\sigma'_q}$ of Q_{σ_q} can be proved equal to a summand of P_{σ_p}. It follows that $\mathcal{A}_1 \vdash P_{\sigma_p} =^s Q_{\sigma_q}$, by using **A3** to eliminate duplicate summands and **A1** and **A2** to reorder and regroup summands as necessary. □

Now, we strengthen notion of standard form as follows.

Definition 7.24. P_{σ_p} is a *full standard form* if

1. $P_{\sigma_p} \equiv \sum_{i=1}^{m} \alpha_{i,c_i}.P_{i,\sigma_i}$, where each P_{i,σ_i} is in full standard form
2. Whenever $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ then $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\rightarrow} P'_{\sigma'_p}$ ■

The rest of the chapter introduces notions used somehow for definition of inheritance.

Lemma 7.28. ***(Saturation)*** *If* $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$, *then* $\mathcal{A}_2 \vdash P_{\sigma_p} =^s P_{\sigma_p} + \alpha_{c_\alpha}.P'_{\sigma'_p}$.

Proof. By induction on the structure of P_{σ_p}. Consider three cases for $P_{\sigma_p} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$:
Case 1 $\alpha_{c_\alpha}.P'_{\sigma'_p}$ is a summand of P_{σ_p}. We can use **A3**
Case 2 $\alpha_{c_\alpha}.Q_{\sigma_q}$ is a summand of P_{σ_p} and $Q_{\sigma_q} \overset{\tau_c}{\Longrightarrow} P'_{\sigma'_p}$. Then by induction $\mathcal{A}_2 \vdash Q_{\sigma_q} =^s Q_{\sigma_q} + \tau_c.P'_{\sigma'_p}$, so $\mathcal{A}_2 \vdash P_{\sigma_p} =^s P_{\sigma_p} + \alpha_{c_\alpha}.Q_{\sigma_q} =^s P_{\sigma_p} + \alpha_{c_\alpha}.(Q_{\sigma_q} + \tau_c.P'_{\sigma'_p}) =^s P_{\sigma_p} + \alpha_{c_\alpha}.(Q_{\sigma_q} + \tau_c.P'_{\sigma'_p}) + \alpha_{c_\alpha}.P'_{\sigma'_p} = P_{\sigma_p} + \alpha_{c_\alpha}.P'_{\sigma'_p}$
Case 3 $\tau_c.Q_{\sigma_q}$ is a summand of P_{σ_p} and $Q_{\sigma_q} \overset{\alpha_{c_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$. Then by induction $\mathcal{A}_2 \vdash Q_{\sigma_q} =^s Q_{\sigma_q} + \alpha_{c_\alpha}.P'_{\sigma'_p}$, so $\mathcal{A}_2 \vdash P_{\sigma_p} =^s P_{\sigma_p} + \tau_c.Q_{\sigma_q} =^s P_{\sigma_p} + \tau_c.Q_{\sigma_q} + Q_{\sigma_q} =^s P_{\sigma_p} + \tau_c.Q_{\sigma_q} + Q_{\sigma_q} + \alpha_{c_\alpha}.P'_{\sigma'_p} = P_{\sigma_p} + \alpha_{c_\alpha}.P'_{\sigma'_p}$

□

Lemma 7.29. *For any standard form P_{σ_p} there is a full standard form $P'_{\sigma'_p}$ of equal depth, such that $\mathcal{A}_2 \vdash P_{\sigma_p} =^s P'_{\sigma'_p}$.*

Proof. By induction on the structure of P_{σ_p}. For the base case, $P_{\sigma_p} \equiv \mathbf{0}$ is a full standard form.
We assume each summand $\beta_{c_\beta}.Q_{\sigma_q}$ of P_{σ_p} is already converted by $\mathcal{A}_2$ into a full standard form, without depth increase. Let us consider all pairs $(\alpha_{i,c_i}, P_{i,\sigma_i}), 1 \leq i \leq k$ such that $P_{\sigma_p} \overset{\alpha_{i,ci}}{\Longrightarrow} P_{i,\sigma_i}$, but not $P_{\sigma_p} \xrightarrow{\alpha_{i,ci}} P_{i,\sigma_i}$. Each P_{i,σ_i} must be in full standard form since it must be a subexpression of some summand $\beta_{c_\beta}.Q_{\sigma_q}$

$$P'_{\sigma'_p} \equiv P_{\sigma_p} + \alpha_{1,c_1}.P_{1,\sigma_1} + \cdots + \alpha_{k,c_k}.P_{k,\sigma_k}$$

is a full standard form of equal depth to P_{σ_p}, and by Lemma 7.28 $\mathcal{A}_2 \vdash P_{\sigma_p} =^s P'_{\sigma'_p}$. □

The last proposition is introduced without proof that may be found in [45].

Proposition 7.30. $P_{\sigma_p} =^s Q_{\sigma_q}$ *iff* $\mathcal{A}_2 \vdash P_{\sigma_p} =^s Q_{\sigma_q}$.

Chapter 8

Inheritance of Behavior

This chapter is devoted to inheritance of behavior or inheritance of processes. Inheritance is one of the key issues of object-oriented modeling. It allows definition of subclasses that inherit features of some superclasses. In contrast to static objects, there is no general agreement on the meaning of inheritance when considering the dynamic behavior of objects. The grounds for this chapter are some of latest articles of Twan Basten and Wil van den Aalst [15, 7, 8, 6] that have addressed the issue.

The goal of object-oriented design is the *reuse* of system components. We think that this goal is very important for processes, too. When implemented, it allows to repeatedly use already existing and validated processes (or tasks) and via inheritance add new features that are not present in the base definition.

Let us consider the question when one process extends another process in some more detail, or in other words: *When is one process a subclass of another process?* Process calculus assumes interaction of a process with environment. The process therefore contains actions that interact with environment - external actions. On the other side, there may be internal actions that change only internal state of the process and do not contribute to the external behavior of the process. We are able to abstract such processes with perfect actions.

The cited authors find two answers to the question. Assume that P_{σ_p} and Q_{σ_q} are two processes. The first answer is as follows:

> If it is not possible to distinguish the external behavior of P_{σ_p} and Q_{σ_q} when only methods of P_{σ_p} that are also present in Q_{σ_q} are executed, then P_{σ_p} is a subclass of Q_{σ_q}.

Intuitively, this basic form of inheritance conforms to *blocking* calls to methods new in P_{σ_p}. We shall say that P_{σ_p} inherits the *protocol* of Q_{σ_q} and the resulting form of inheritance is referred to as *protocol inheritance*. The second answer is as follows:

> If it is not possible to distinguish the external behavior of P_{σ_p} and Q_{σ_q} when arbitrary methods of P_{σ_p} are executed, but when only the effects of methods that are also present in Q_{σ_q} are considered, then P_{σ_p} is a subclass of Q_{σ_q}.

This form conforms to *hiding* the effect of methods new in P_{σ_p}. Process P_{σ_p} inherits the *projection* of the life cycle of P_{σ_p} onto methods of Q_{σ_q}. The resulting form of inheritance is called *projection inheritance*.

We are going to introduce extension of secure CCS to allow specification of inheritance. Other section shall be devoted to the definition of inheritance in secure process calculus. The last part of the chapter introduces inheritance rules.

8.1 Extension of Secure CCS

Inheritance of secure processes is a little more complicated than inheritance without considering security. The way we go has been partially determined by van den Aalst and Basten. We are to introduce new set(s) of axioms that allow us to extend equivalence of processes. It is the first problem that has to be solved - introduce new axioms that are definitely distinct from the original ones defined in ACP. We have some advantage because CCS contains restriction operator that can be used to define one form of inheritance and there is also a relabelling function that can be used for ignoring effects of some actions we want to hide.

But we have to go further, there is the second problem - equality of secure processes. Assume an example. We have definition of a process - task. We create a subprocess (from now on, the notion is used for inherited process) by adding new branches, new parallel states and actions. In the moment we start to hide those new actions (or their effects) we can not do the same with the all related security information. The result is that we obtain two equal processes, but only from the functional point of view. Security properties of them are different.

To solve problem with equality of processes, all possibilities were explored: some changes in axioms, changes in definitions of inheritance, ignoring security properties to some extent. Finally, the solution has been identified in minor changes in inheritance definitions, but primarily in introducing new bisimulation.

8.1.1 Axioms

We are going to extend set of axioms for finite state agents (Section 7.6.1). The basis shall be set of axioms $\mathcal{A}_2$. We need τ actions to allow abstraction. This is because of the definition of inheritance that is partially based on *ignoring* actions new in subprocess.

The first task is to find a way to express encapsulation and abstraction. Basten and Van den Aalst use ACP [17] process algebra and they introduce two new operators and use special actions τ and δ for results. We know τ action in the process calculus, but we do not use δ actions (inaction). We assume that it is possible to use inactive agent $\mathbf{0}$ instead. The first idea was to add new special auxiliary axiom for $\mathbf{0}$, but it seems that it is not necessary because of the expansion law (see Corollary 7.3 on page 76) that is sufficient for the purpose.

We have two basic types of inheritance: protocol and projection. We therefore define two new sets of axioms for each type and the third one for their composition. We obtain axioms for protocol inheritance, projection inheritance and the last set contains axioms for both types of inheritance.

Axioms $\mathcal{A}_3$ $\quad L \subseteq Act$

A1-A7

D1 $\alpha_{c_\alpha} \notin L \cup \overline{L} \Rightarrow (\alpha_{c_\alpha}.P_{\sigma_p})\backslash L =^s \alpha_{c_\alpha}.P_{\sigma_p}\backslash L$

D2 $\alpha_{c_\alpha} \in L \cup \overline{L} \Rightarrow (\alpha_{c_\alpha}.P_{\sigma_p})\backslash L =^s \mathbf{0}$

D3 $(P_{\sigma_p} + Q_{\sigma_q})\backslash L =^s P_{\sigma_p}\backslash L + Q_{\sigma_q}\backslash L$

For the projection inheritance, we use a relabelling function. We may call it τ_L, where $L \cup \overline{L}$ is the set of labels that are to be hidden.

Axioms $\mathcal{A}_4$ $\quad L \subseteq Act$

A1-A7

I1 $\alpha_{c_\alpha} \notin L \cup \overline{L} \Rightarrow \tau_L(\alpha_{c_\alpha}) =^s \alpha_{c_\alpha}$

I2 $\alpha_{c_\alpha} \in L \cup \overline{L} \Rightarrow \tau_L(\alpha_{c_\alpha}) =^s \tau_{c_\alpha}$

I3 $(P_{\sigma_p} + Q_{\sigma_q})[\tau_L] =^s P_{\sigma_p}[\tau_L] + Q_{\sigma_q}[\tau_L]$

I4 $(\alpha_{c_\alpha}.P_{\sigma_p})[\tau_L] =^s \tau_L(\alpha_{c_\alpha}).P_{\sigma_p}[\tau_L]$

Axioms $\mathcal{A}_5$

A1-A7, D1-D3, I1-I4

8.1.2 Equality

Next task that has to be solved is new definition of observation bisimulation that allows a little different treatment of authorization information. The definition of secure equality (Def 7.20) uses previously defined notion of observation equivalence. Both relations are equivalence relations. We need to introduce new notions that allow to define inheritance with one relaxation. Authorization information of states and actions in a subprocess is greater or equal to respective authorization information in the base process. The resulting relation is not equivalence because it lacks symmetry of security information, but preorder. We shall call the new relation *inheritance bisimulation*.

Definition 8.1. (Inheritance bisimulation) A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over secure agents is an *inheritance bisimulation* if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ implies, for all $\alpha_{c_\alpha} \in Act$,

1. Whenever $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ then, $\exists\ Q'_{\sigma'_q}$, $Q_{\sigma_q} \overset{\widehat{\alpha}_{c'_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$ such that $c'_\alpha \leq c_\alpha$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$

2. Whenever $Q_{\sigma_q} \xrightarrow{\alpha_{c_\alpha}} Q'_{\sigma'_q}$ then, $\exists\ P'_{\sigma'_p}$, $P_{\sigma_p} \overset{\widehat{\alpha}_{c'_\alpha}}{\Longrightarrow} P'_{\sigma'_p}$ such that $c_\alpha \leq c'_\alpha$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}$ ■

We continue with the definition of *inheritance bisimilar* processes.

Definition 8.2. P_{σ_p} and Q_{σ_q} are *inheritance bisimilar*, written $P_{\sigma_p} \preceq^s Q_{\sigma_q}$, if $(P_{\sigma_p}, Q_{\sigma_q}) \in \mathcal{S}$ for some inheritance bisimulation $\mathcal{S}$. That is

$$\preceq^s = \bigcup \{\mathcal{S} : \ \mathcal{S} \text{ is an inheritance bisimulation}\}$$

■

With those basic definitions we may state first properties of inheritance bisimulation.

Proposition 8.1. *Assume that each $\mathcal{S}_i (i = 1, 2, \ldots)$ is inheritance bisimulation. Then the following are all inheritance bisimulations:*

(1) Id_p (3) $\bigcup_{i \in I} \mathcal{S}_i$

(2) $\mathcal{S}_1 \mathcal{S}_2$

Proof. (1) is proved by substitution of Q_{σ_q} with P_{σ_p}. And $c_\alpha \leq c_\alpha$. We may interpret it as statement that any process is subprocess of itself.
(2) There exists agent R_{σ_r} such that $P_{\sigma_p} \mathcal{S}_1 R_{\sigma_r} \mathcal{S}_2 Q_{\sigma_q}$. Let $P_{\sigma_p} \xrightarrow{\alpha_{c_\alpha}} P'_{\sigma'_p}$ and for some $R'_{\sigma'_r} : R_{\sigma_r} \overset{\widehat{\alpha}_{c'_\alpha}}{\Longrightarrow} R'_{\sigma'_r}$ and for some $Q'_{\sigma'_q} : Q_{\sigma_q} \overset{\widehat{\alpha}_{c''_\alpha}}{\Longrightarrow} Q'_{\sigma'_q}$. It implies that $c_\alpha \leq c'_\alpha \leq c''_\alpha$ and $(P'_{\sigma'_p}, Q'_{\sigma'_q}) \in \mathcal{S}_1 \mathcal{S}_2$. Interpretation is again intuitive; child of a subprocess is also subprocess.
(3) It is union of all subprocesses of a given process. □

8.2 Definition of Inheritance

This section shall characterize basic properties of inheritance of behavior. The base are two types of inheritance identified in the introduction and we define four different inheritance relations. We show that equational laws preserve all inheritance relations.

The first definition shall formalize inheritance of a process by another process. We have identified *protocol inheritance* and *projection inheritance* that correspond to encapsulation and abstraction. The subtle difference between the two forms of inheritance introduced so far is that under projection inheritance are actions executed without taking into account their effect, whereas under protocol inheritance they are not executed at all.

Definition 8.3. (Inheritance relations)

1. Protocol inheritance: For any processes P_{σ_p} and Q_{σ_q}, P_{σ_p} is a subprocess of Q_{σ_q} under *protocol inheritance*, iff there exists $K \subseteq Act$ such that $\mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q}$.

   $P_{\sigma_p} \leq_{pt} Q_{\sigma_q}$

2. Projection inheritance: For any processes P_{σ_p} and Q_{σ_q}, P_{σ_p} is a subprocess of Q_{σ_q} under *projection inheritance*, iff there exists $L \subseteq Act$ such that $\mathcal{A}_4 \vdash (P_{\sigma_p}[\tau_L]) \preceq^s Q_{\sigma_q}$.

   $P_{\sigma_p} \leq_{pj} Q_{\sigma_q}$

3. Total inheritance: For any processes P_{σ_p} and Q_{σ_q}, P_{σ_p} is a subprocess of Q_{σ_q} under *total inheritance*, iff there exists $K \subseteq Act$ such that $\mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q}$ and there exists $L \subseteq Act$ such that $\mathcal{A}_4 \vdash (P_{\sigma_p}[\tau_L]) \preceq^s Q_{\sigma_q}$.

   $P_{\sigma_p} \leq_{t} Q_{\sigma_q}$

4. Life-cycle inheritance: For any processes P_{σ_p} and Q_{σ_q}, P_{σ_p} is a subprocess of Q_{σ_q} under *life-cycle inheritance*, iff there exist $K, L \subseteq Act : (K \cup \overline{K}) \cap (L \cup \overline{L}) = \emptyset$ such that $\mathcal{A}_5 \vdash (P_{\sigma_p} \backslash K)[\tau_L] \preceq^s Q_{\sigma_q}$.

   $P_{\sigma_p} \leq_{lc} Q_{\sigma_q}$

There is given a reasonable requirement that K and L, it means set of actions that are to be encapsulated and abstracted (hidden), are disjoint. This requirement should ensure that order of projection and protocol *operators* application can be interchanged without change of the result.

When reasoning about security properties of a process and its subprocesses we find out that security requirements on subjects to perform subprocesses are at least equal to the requirements for the base process. It is implied by definition of inheritance bisimulation.

We are going to use more extensively the notion of sort. We say that L is a sort of P_σ and write it as $P_\sigma : L$. We may also write $L = \mathcal{L}(P_\sigma)$ (for details see Subsection 3.1.5). We are also going to use standard forms of processes.

Before we try to formulate some useful ideas we state that inheritance using empty sets results the original process.

Lemma 8.2. *For any process P_{σ_p} and sorts K, L*

1. $\mathcal{L}(P_{\sigma_p}) \cap (K \cup \overline{K}) = \emptyset \Longrightarrow P_{\sigma_p} \backslash K =^s P_{\sigma_p}$
2. $\mathcal{L}(P_{\sigma_p}) \cap (L \cup \overline{L}) = \emptyset \Longrightarrow P_{\sigma_p}[\tau_L] =^s P_{\sigma_p}$

Proof. While P_{σ_p} is a standard form, it may be written as $\sum_{i \in I} \alpha_{i,c_i}.P_{i,\sigma_i}$ where all P_{i,σ_i} are also a standard form.

$$\begin{aligned} P_{\sigma_p} = \sum_{i \in I} \alpha_{i,c_i}.P_{i,\sigma_i} \backslash K &=^s \sum_{i \in I} \Big((\alpha_{i,c_i}.P_{i,\sigma_i}) \backslash K \Big) && \textbf{D3} \\ &=^s \sum_{i \in I} \Big(\alpha_{i,c_i}.P_{i,\sigma_i} \backslash K \Big) && \textbf{D1} \end{aligned}$$

And we repeat the approach on each P_{i,σ_i} until the end.
The second statement is proved by the same way, but axioms I3, I4 and I1 are used instead. □

It is possible to formulate the following lemma about strength of different types of inheritance.

Lemma 8.3. *The following statements about process and its subprocess hold:*

1. $P_{\sigma_p} \leq_t Q_{\sigma_q} \Longrightarrow P_{\sigma_p} \leq_{pt} Q_{\sigma_q} \wedge P_{\sigma_p} \leq_{pj} Q_{\sigma_q}$
2. $P_{\sigma_p} \leq_{pt} Q_{\sigma_q} \Longrightarrow P_{\sigma_p} \leq_{lc} Q_{\sigma_q}$
3. $P_{\sigma_p} \leq_{pj} Q_{\sigma_q} \Longrightarrow P_{\sigma_p} \leq_{lc} Q_{\sigma_q}$

Proof. Statement (1) is implied directly from the definition of total inheritance.
(2) $P_{\sigma_p} \leq_{pt} Q_{\sigma_q}$ means that $\mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q}$. We need to show that there is a sort L such that $\mathcal{A}_5 \vdash (P_{\sigma_p} \backslash K)[\tau_L] \preceq^s Q_{\sigma_q}$. But we showed that when $L = \emptyset$ then process does not change - $\mathcal{A}_5 \vdash (P_{\sigma_p} \backslash K)[\tau_\emptyset] \preceq^s Q_{\sigma_q} \iff \mathcal{A}_5 \vdash (P_{\sigma_p} \backslash K) \preceq^s Q_{\sigma_q}$ and set of axioms $\mathcal{A}_5 \supseteq \mathcal{A}_3$. And it is true because of the presumption.
(3) is proved in the same way as item (2). □

So, life-cycle inheritance is the most general one. Protocol and projection inheritance are extensions of the life-cycle inheritance and total inheritance is composition of the protocol and projection inheritance together.

8.3 Properties of Inheritance

It is necessary to show some basic properties of inheritance relations. To do so, we use properties introduced in [15]. First of all we introduce two lemmas about alphabets of subprocesses and composition of inheritance relations.

Lemma 8.4. *For any standard form P_{σ_p} and sorts $K, L \subseteq \mathcal{L}(P_{\sigma_p})$,*

1. $\mathcal{L}(P_{\sigma_p} \backslash K) \cap K = \emptyset$
2. $\mathcal{L}(P_{\sigma_p}[\tau_L]) \cap L = \emptyset$

Proof. We start with encapsulation.
Assume that $\mathcal{L}(P_{\sigma_p} \backslash K) \cap K \neq \emptyset$. It means that there is an action $\alpha_c \in K$ in the reduced process. But P_{σ_p} is a standard form and therefore there must be a summand $\alpha_c.P_{i,\sigma_i}$. But according to axiom **D2**, such a summand is reduced to **0**. It is conflict with presumption.
Abstraction is proved the same way, but with use of axiom **I2**. □

Lemma 8.5. *For any standard form P_{σ_p} and sorts $K, K', L, L' \subseteq \mathcal{L}(P_{\sigma_p})$,*

1. $\mathcal{A}_3 \vdash P_{\sigma_p} \backslash (K \cup K') =^s P_{\sigma_p} \backslash K \backslash K'$
2. $\mathcal{A}_4 \vdash P_{\sigma_p}[\tau_{L \cup L'}] =^s (P_{\sigma_p}[\tau_L])[\tau_{L'}]$

Proof. Both statements are already introduced in restriction (Proposition 3.9) and relabelling (Proposition 3.10) laws. They can be showed by application of axioms **D** and **I** on parts of standard form. □

The following property exhibits how to easily determine sorts to show existence of inheritance relation.

Proposition 8.6. *For any standard forms $P_{\sigma_p}, Q_{\sigma_q}$ and sorts $K = \mathcal{L}(P_{\sigma_p}) \backslash \mathcal{L}(Q_{\sigma_q})$ and L, L' such that $L \cup L' = K \wedge L \cap L' = \emptyset$,*

1. $P_{\sigma_p} \leq_{pt} Q_{\sigma_q} \Longleftrightarrow \mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q}$
2. $P_{\sigma_p} \leq_{pj} Q_{\sigma_q} \Longleftrightarrow \mathcal{A}_4 \vdash P_{\sigma_p}[\tau_K] \preceq^s Q_{\sigma_q}$
3. $P_{\sigma_p} \leq_{t} Q_{\sigma_q} \Longleftrightarrow \mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q} \wedge \mathcal{A}_4 \vdash P_{\sigma_p}[\tau_K] \preceq^s Q_{\sigma_q}$
4. $P_{\sigma_p} \leq_{lc} Q_{\sigma_q} \Longleftrightarrow \mathcal{A}_5 \vdash (P_{\sigma_p} \backslash L)[\tau_{L'}] \preceq^s Q_{\sigma_q}$

Proof. We demonstrate the proof for protocol inheritance. Remaining statements are proved in a similar way.
Direction from right to left is implied by the definition of inheritance relations for all of them.
Definition of protocol inheritance says that if P_{σ_p} is a subprocess of Q_{σ_q} then there exists a sort K such that $\mathcal{A}_3 \vdash P_{\sigma_p} \backslash K \preceq^s Q_{\sigma_q}$. Because of Lemma 8.2 we may restrict K to a subset of $\mathcal{L}(P\sigma_p)$. We need to examine following three possibilities

- $K \subset \mathcal{L}(P_{\sigma_p}) \backslash \mathcal{L}(Q_{\sigma_q})$ The sort of reduced P_{σ_p} is then superset to the sort of Q_{σ_q} and we can find an action $\alpha_c : \alpha_c \in \mathcal{L}(P_{\sigma_p} \backslash K) \wedge \alpha_c \notin \mathcal{L}(Q_{\sigma_q})$. It means that processes are not inheritance bisimilar.
- $K \supset \mathcal{L}(P_{\sigma_p}) \backslash \mathcal{L}(Q_{\sigma_q})$ By the same procedure we may find $\alpha_c : \alpha_c \notin \mathcal{L}(P_{\sigma_p} \backslash K) \wedge \alpha_c \in \mathcal{L}(Q_{\sigma_q})$ and processes are neither bisimilar.
- $K = \mathcal{L}(P_{\sigma_p}) \backslash \mathcal{L}(Q_{\sigma_q})$. And because there must exist a sort K, this is it.

□

Any inheritance relation should satisfy two basic properties - reflexivity and transitivity. The above defined inheritance relations satisfy those two properties because they are based on inheritance bisimulation that satisfies them. To be precise, inheritance bisimulation between base process and its subprocess after reduction and/or relabelling is equivalent with appropriate inheritance relation.

Definition of process inheritance over ACP introduces a notion of context. We may understand it as a process definition with a hole that can be filled up by definition of some other process. I suppose that we may substitute it by mechanism that replaces an agent constant with an agent expression.

The original proposal can be demonstrated by the following example. Assume that $C = a + \_$ is a context, where $\_$ is the hole and a is an action. We can now define substitution $C[p]$ of hole in the context C by p. When $p = a.c + b$, the result of substitution $C[a.c + b]$ is $a + (a.c + b)$. This way is not possible in CCS, but we may use another one to perform the same idea. Why are we introducing the concept at all? We wish to replace elements of one process by their subprocesses. The whole issue complicates a little more because of our requirement of security of resulting processes. The most important is the fact that security correctness of a process assumes particular security properties of all its elements. We can not do any changes recognizable from that point of view because it would require changes in considerable part of the original process definition.

Definition 8.4. (Inheritance substitution) Let P_{σ_p} be a standard form that contains other standard forms P_{i,σ_i} for all $i \in I$. Substitution inheritance $C(P_{\sigma_p}, P_{i,\sigma_i}, Q_{\sigma_i})$ replaces in P_{σ_p} agent P_{i,σ_i} with Q_{σ_i}. ■

Lemma 8.7. *Having a standard form Q_{σ_q} and a standard form P_{σ_p} containing P_{i,σ_i}, then result of $C(P_{\sigma_p}, P_{i,\sigma_i}, Q_{\sigma_i})$ is also a standard form.*

Proof. By induction on the structure of $C(P_{\sigma_p}, P_{i,\sigma_i}, Q_{\sigma_i})$. It is the same as for P_{σ_p}, but on certain level, we obtain $P'_{\sigma'_p} \equiv \sum_{i=1}^{m-1} \alpha_{i,c_i}.P_{i,\sigma_i} + \alpha_{m,c_m}.Q_{\sigma_q}$. And because Q_{σ_q} is in standard form, $C(P_{\sigma_p}, P_{i,\sigma_i}, Q_{\sigma_i})$ is also in standard form. □

We are now able to introduce requirements for application of inheritance substitution while preserving inheritance relations.

Proposition 8.8. *Let P_{σ_p}, Q_{σ_q}, R_{σ_r} be standard forms as well as S_{σ_s} and its element $S'_{\sigma'_s}$, further let $C(\_,\_,\_)$ be inheritance substitution. Let $K, K', L, L' \subseteq (\mathcal{L}(P_{\sigma_p}) \cup \mathcal{L}(Q_{\sigma_q}) \cup \mathcal{L}(R_{\sigma_r}) \cup \mathcal{L}(S_{\sigma_s}))$ be sorts.*

1. *If $P_{\sigma_p} \leq_{pt} Q_{\sigma_q}$ with K such that $\mathcal{A}_3 \vdash P_{\sigma_p}\backslash K \preceq^s Q_{\sigma_q}$ and $C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q}) \leq_{pt} R_{\sigma_r}$, then*
   $\mathcal{L}(R_{\sigma_r}) \cap K = \emptyset \Rightarrow C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}) \leq_{pt} R_{\sigma_r}$
2. *If $P_{\sigma_p} \leq_{pj} Q_{\sigma_q}$ with L such that $\mathcal{A}_4 \vdash P_{\sigma_p}[\tau_L] \preceq^s Q_{\sigma_q}$ and $C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q}) \leq_{pj} R_{\sigma_r}$, then*
   $\mathcal{L}(R_{\sigma_r}) \cap L = \emptyset \Rightarrow C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}) \leq_{pj} R_{\sigma_r}$
3. *If $P_{\sigma_p} \leq_{t} Q_{\sigma_q}$ with K, L such that $\mathcal{A}_3 \vdash P_{\sigma_p}\backslash K \preceq^s Q_{\sigma_q}$ and $\mathcal{A}_4 \vdash P_{\sigma_p}[\tau_L] \preceq^s Q_{\sigma_q}$ and $C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q}) \leq_{t} R_{\sigma_r}$, then*
   $\mathcal{L}(R_{\sigma_r}) \cap (L \cup K) = \emptyset \Rightarrow C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}) \leq_{t} R_{\sigma_r}$
4. *If $P_{\sigma_p} \leq_{lc} Q_{\sigma_q}$ with K, L such that $\mathcal{A}_5 \vdash (P_{\sigma_p}\backslash K)[\tau_L] \preceq^s Q_{\sigma_q}$ and $C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q}) \leq_{lc} R_{\sigma_r}$ since $\mathcal{A}_5 \vdash (C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})\backslash K')[\tau_{L'}] \preceq^s R_{\sigma_r}$, then*
   $\mathcal{L}(R_{\sigma_r}) \cap (L \cup K) = (K \cup K') \cap (L \cup L') = \emptyset \Rightarrow C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}) \leq_{lc} R_{\sigma_r}$

Proof. Let us start with proposition 1. Assume, that K' has been used to restrict $C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})$, i.e. $\mathcal{A}_3 \vdash C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})\backslash K' \preceq^s R_{\sigma_r}$. We use properties of restriction operator and definition of inheritance substitution in the following derivation.

$$\begin{aligned} C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p})\backslash(K \cup K') &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p})\backslash K\backslash K' \\ &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}\backslash K\backslash K')\backslash K\backslash K' =^s C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q}\backslash K\backslash K')\backslash K\backslash K' \\ &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})\backslash K\backslash K' =^s R_{\sigma_r}\backslash K' =^s R_{\sigma_r} \end{aligned}$$

It shows that $C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p}) \leq_{pt} R_{\sigma_r}$.
Proposition 2 is proven similarly and proposition 3 is composition of the previous twos.
Proof of proposition 4 is done by derivation that uses distributivity of restriction and relabelling over all elements of agents and the restrictive condition $(K \cup K') \cap (L \cup L')$.

$$\begin{aligned} C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p})\backslash(K \cup K')[\tau_{L\cup L'}] &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p})\backslash K\backslash K'[\tau_L] \circ [\tau_{L'}] \\ &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, P_{\sigma_p})\backslash K[\tau_L]\backslash K'[\tau_{L'}] =^s C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})\backslash K[\tau_L]\backslash K'[\tau_{L'}] \\ &=^s C(S_{\sigma_s}, S'_{\sigma'_s}, Q_{\sigma_q})\backslash K'[\tau_{L'}]\backslash K[\tau_L] =^s R_{\sigma_r}\backslash K[\tau_L] =^s R_{\sigma_r} \end{aligned}$$

□

At the end of this section we can repeat some of results of van den Aalst and Basten. Definitions of the four inheritance relations are sound and all inheritance relations are preorder. The conditions necessary for inheritance of processes seem reasonable. Because of the security requirements, we have been forced to introduced new definition of bisimilarity. It is not necessary for theoretical reasons, but I suppose that it is very useful for practical applications.

8.4 Further Remarks

This section contains some remarks about security of inheritance and also a few examples of inheritance relations. There were defined some typical cases in ACP that have been stated as axioms of inheritance. I am unfortunately not in a position to call the stated relations as axioms.

We have not especially considered security of processes in inheritance relations. The reason is very simple, we are using constructs and mechanisms already described in Chapter 7. We have introduced only one new notion of inheritance substitution for which the security is shown.

I would like to mention necessity for usage of non-equality relations in definitions of bisimulations in Secure CCS. We were in doubt, whether to demand equality of classification of consecutive states and actions. Our decision to define inheritance of processes solved the question very quickly. When allowing *lower or equal*, we reached much more realistic properties of inherited processes with preserved security properties.

Flow control in inherited or base processes is ensured with security information (classifications) that stays preserved. Those information may be just split into several actions or put together from several subsequent 'ignored' actions according to realized inheritance.

Any process (task) may be changed in three ways. We may change behavior of a process - replacing part of a task with a subprocess of that part. We also may insert some behavior in between sequential parts of the original process and we also may put a new behavior in parallel with a part of the process. We are now to introduce some examples and show why there is satisfied inheritance relation.

Lemma 8.9. *For any processes P_{σ_p} and Q_{σ_q} and $\alpha_{c_1}, \beta_{c_2} \in Act$ such that $\alpha_{c_1}, \beta_{c_2} \notin \mathcal{L}(P_{\sigma_p})$.*

$$\alpha_{c_1}.(P_{\sigma_p} + \beta_{c_2}.Q_{\sigma_q}) \leq_{pt} \alpha_{c_1}.P_{\sigma_p}$$

Proof. Let $K = \{\beta_{c_2}\}$

$$\mathbf{A_3} \vdash \alpha_{c_1}.(P_{\sigma_p} + \beta_{c_2}.Q_{\sigma_q})\backslash K \overset{D1,D3}{=^s} \alpha_{c_1}.(P_{\sigma_p}\backslash K + (\beta_{c_2}.Q_{\sigma_q})\backslash K)$$
$$\overset{D2}{=^s} \alpha_{c_1}.(P_{\sigma_p}\backslash K + \mathbf{0}) =^s \alpha_{c_1}.(P_{\sigma_p}) =^s \alpha_{c_1}.P_{\sigma_p}$$

We also should show that security information of the left side is higher or equal to the security information of the right side. We omit this proof because of its simplicity. If you remember, the security information is equal to the properties of the last state in the process.
You may notice that under protocol inheritance we can add new parallel branches with new end states. □

Action β_{c_2} may be seen as a guard. When blocking the action, process Q_{σ_q} is *prohibited* and can not be chosen. To ensure the blocking function of β_{c_2} it must not be in the sort of P_{σ_p}, otherwise blocking of Q_{σ_q} would change behavior of P_{σ_p}.

Lemma 8.10. *For any processes P_{σ_p} and Q_{σ_q} and $\alpha_{c_1}, \beta_{c_2} \in Act$ such that $\alpha_{c_1}, \beta_{c_2} \notin \mathcal{L}(P_{\sigma_p}) \cup \mathcal{L}(Q_{\sigma_q})$.*

$$\alpha_{c_1}.(\beta_{c_2}.(P_{\sigma_p} + Q_{\sigma_q}) + Q_{\sigma_q}) \leq_{pj} \alpha_{c_1}.(P_{\sigma_p} + Q_{\sigma_q})$$

Proof. Let $L = \{\beta_{c_2}\}$

$$\begin{aligned}
\mathbf{A_4} &\vdash (\alpha_{c_1}.(\beta_{c_2}.(P_{\sigma_p} + Q_{\sigma_q}) + Q_{\sigma_q}))[\tau_L] \overset{I4,I1}{=^s} \alpha_{c_1}.(\beta_{c_2}.(P_{\sigma_p} + Q_{\sigma_q}) + Q_{\sigma_q})[\tau_L] \\
&\overset{I3}{=^s} \alpha_{c_1}.((\beta_{c_2}.(P_{\sigma_p} + Q_{\sigma_q}))[\tau_L] + Q_{\sigma_q}[\tau_L]) \overset{I4,I1}{=^s} \alpha_{c_1}.(\tau_{c_2}.(P_{\sigma_p} + Q_{\sigma_q}) + Q_{\sigma_q}) \\
&\overset{A6,A5}{=^s} \alpha_{c_1}.(P_{\sigma_p} + Q_{\sigma_q})
\end{aligned}$$

We have shown possibility to add new actions into existing sequences and new *paths* to existing states. □

Lemma 8.11. *For any process P_{σ_p} and $\alpha_{c_1}, \beta_{c_2} \in Act$ such that $\beta_{c_2} \notin \mathcal{L}(P_{\sigma_p})$.*

$$\alpha_{c_1}.(\beta_{c_2}.P_{\sigma_p} + P_{\sigma_p}) \leq_t \alpha_{c_1}.P_{\sigma_p}$$

Proof. We have to prove that right side is derivable with protocol as well as with projection inheritance. So let $K = \{\beta_{c_2}\}$ and $L = \{\beta_{c_2}\}$.

$$\begin{aligned}
\mathbf{A_3} &\vdash (\alpha_{c_1}.(\beta_{c_2}.P_{\sigma_p} + P_{\sigma_p}))\backslash K \overset{D1,D3}{=^s} \alpha_{c_1}.((\beta_{c_2}.P_{\sigma_p}\backslash K + P_{\sigma_p}\backslash K) \\
&\overset{D2}{=^s} \alpha_{c_1}.(\mathbf{0} + P_{\sigma_p}) \overset{A4}{=^s} \alpha_{c_1}.P_{\sigma_p} \\
\mathbf{A_4} &\vdash (\alpha_{c_1}.(\beta_{c_2}.P_{\sigma_p} + P_{\sigma_p}))[\tau_L] \overset{I4,I1}{=^s} \alpha_{c_1}.(\beta_{c_2}.P_{\sigma_p} + P_{\sigma_p})[\tau_L] \\
&\overset{I3,I2}{=^s} \alpha_{c_1}.(\tau_{c_2}.P_{\sigma_p} + P_{\sigma_p}) \overset{A6}{=^s} \alpha_{c_1}.\tau_{c_2}.P_{\sigma_p} \overset{A5}{=^s} \alpha_{c_1}.P_{\sigma_p}
\end{aligned}$$

This example shows that total inheritance allows to add new *ways* to already existing states. □

Lemma 8.12. *For any processes P_{σ_p} and Q_{σ_q} and $\alpha_{c_1}, \beta_{c_2}, \gamma_{c_3} \in Act$ such that $\beta_{c_2}, \gamma_{c_3} \notin \mathcal{L}(P_{\sigma_p}) \cup \mathcal{L}(Q_{\sigma_q})$.*

$$\alpha_{c_1}.\beta_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_q}) \leq_{lc} \alpha_{c_1}.P_{\sigma_p}$$

Proof. To show that right side is the base process for the left side we have to use combination of both types of inheritance. Assume that $K = \{\beta_{c_2}\}$ and $L = \{\gamma_{c_3}\}$.

$$\begin{aligned}
\mathbf{A_5} &\vdash (\alpha_{c_1}.\beta_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_p}))[\tau_L]\backslash K \overset{I4,I1}{=^s} (\alpha_{c_1}.(\beta_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_p}))[\tau_L])\backslash K \\
&\overset{I4,I2}{=^s} (\alpha_{c_1}.\tau_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_p})[\tau_L])\backslash K \overset{I4,I1}{=^s} (\alpha_{c_1}.\tau_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_p}))\backslash K \\
&\overset{D1,D1}{=^s} \alpha_{c_1}.\tau_{c_2}.(P_{\sigma_p} + \gamma_{c_3}.Q_{\sigma_p})\backslash K \overset{D3,D2}{=^s} \alpha_{c_1}.\tau_{c_2}.(P_{\sigma_p} + \mathbf{0}) \overset{A5,A4}{=^s} \alpha_{c_1}.P_{\sigma_p}
\end{aligned}$$

As you can see, we are able to eliminate as new branches as new actions in sequences. □

Chapter 9

Conclusions and Directions

The area of security (but not only security) in heterogeneous information systems is generally very problematic. The reason is complexity of each single task that has to be solved. When one describes a problem generally with some theoretic apparatus it is very difficult to materialize results in an efficient implementation. On the other side, existing solutions of particular problems in particular environment can not be usually used for some general proposition.

Security is unique from any other aspect of information system because of its requirements for correctness. Having an information system that has some functional errors, you can still use it with success. But security model with just one fault, one inconsistency is more or less useless. It is just a matter of time when a hacker, a competitor, or an agency invests time and money to find the error and attack the system.

There is more and more attention given to security, but systems are more and more complex and their security mechanisms become more and more complicated. And the more complicated mechanism the higher probability to introduce errors.

This thesis works with very complex systems. What we are trying to, is to introduce security model (authorization model) that allows to define and prove security of computational tasks. We use already existing apparatus for description of processes and extend it with security properties for tasks and their relations.

9.1 Conclusions

When starting to write the thesis, I had one basic target. I wanted to introduce a complex authorization system, or at least its basic ideas, that would be suitable for strongly distributed information systems. It means systems that are heterogeneous that consist of a number of *local* information systems with their own administration and systems that are able to process tasks that need cooperation of users and access to resources on more *local* information systems. The results achieved in the thesis can be seen in four basic areas.

There is given a general idea to solve security requirements for strongly distributed information systems (Chapter 4). Elements of such systems (s-nodes) have their own administration, own sets of users. Resources are accessible through their own environment (operation system, DBMS). There is introduced a new global level that allows definition of tasks - workflows as mentioned above. The problem of cooperation between those two levels is solved through a conversion layer that is based on categorization of resources and subjects. All resources have a categorization that describes their content and subjects are qualified by the same way according to their authorization. This approach is based on the fact that properties of data do not change. Categorization of data does not change until the content is changed. When we start to use new data, we add new subsets into set of categorizations, but the already existing elements do not change.

The second area involves design of Active authorization model. Chapter 5 describes Active authorization model that is suitable for secure workflows. The word *active* is used because the model is able to synchronize authorizations with execution of tasks. This synchronization is based on a new concept of protection states (sets of authorizations) associated with small parts of workflows - task steps. It means that there is not one protection state that exists for the whole life of the system, but appropriate protection state is activated with initialization of a task step. There are also introduced other two very important ideas:

- Strict sepration of access and flow control - it starts already by different definition of privileges.

- General mechanism for transformation of authorizations between workflow and s-node.

There is introduced CCS as a formal apparatus for definition of communicating processes. This process algebra is enriched with security properties for processes and their communication. The approach is described in Chapter 6 that features basic notions necessary for the following description of Secure CCS. Active authorization model is taken as a basis that is in an intuitive way incorporated into process algebra. I want to mention just one moment that is very strongly expressed in Secure CCS.

Assuming access control, all privileges are connected with particular processes. When we change abstraction of the model to a higher level, the privileges are hidden. It means that access control authorizations are not combined into more abstract processes. It implies that it is not necessary for a subject authorized to initiate some process to obtain access rights for s-node resources. Totally different situation comes with flow control. The privileges are combined and any subject must be authorized to work with dynamic objects - objects created during the task (workflow) processing. This distinction between access and flow control is kept very strictly.

The last contribution of the thesis is introduction of basic notions of inheritance of processes for formalism for secure processes. The idea comes from Aalst and Basten that introduced inheritance for dynamic objects in ACP algebra and also in Petri Nets. Chapter 8 incorporates their ideas into Secure CCS. What is it good for? We are able to easily create new workflows by using already existing ones, while preserving abidance of security properties. It should help in administration of large systems - it is easier and more secure. The second issue is about possible dynamic changes of workflows. We are able to create subprocess for a task step of a workflow. When keeping exactly defined conditions, we are able to replace the given task step with a new one that do not have to be known id advance. Regarding security of the workflow, we just have to check properties of the subprocess.

9.2 Future research

I would say that the thesis is some sort of basic work in the area of security in strongly distributed systems. When reading the same chapter (Conclusion) in the thesis of W. K. Huang [39], we can find the following areas: supporting dynamic task dependency, schema-based vs. instance-based workflow specification, extension of workflows into heterogeneous distributed environment and web-based applications. Regarding those subjects, we may see this thesis as continuation of the mentioned work as extension into heterogeneous distributed environment. Although this work does not use approaches that has been introduced there. The rest of ideas stays for the future.

9.2.1 Further Development of Authorization System

Despite all my effort, this thesis covers just basic properties of authorization system and many questions stay unresolved. The introduced solution is a concept of one man and I am certain that somebody else would have proposed some things slightly changed, some completely different. On the other side, the solution proposed is a solid framework for further effort in the area.

The model, as proposed, puts a lot of responsibility to the design of task steps and workflows. It is assumed that workflow templates should undergo careful check before run for the first time. I suppose, there is needed a lot more discussion about dependence between enabled and initial authorizations. The tighter the relation the lower importance of correctness of tasks' design. It is a part of one more general question. What should be enforced automatically and what should be decided by system administrators. Humans make mistakes, but computers may do a lot of mess from one mistake of a human.

Another question is enforcement of processing order of task steps. When implementing AAM into process calculus, the question has been left just on CCS. Is it correct? Is it sufficient for secure processing of the task?

We can find many questions that need more discussion and should take our attention, e.g. what about to define relabelling function also for classification of system elements? I do not know if it is useful, but it can be defined. That is why I say that the thesis is just a corner-stone for a lot of possible future work.

9.2.2 Dynamic Task Dependency

Workflow applications are usually based on a static, schema-based workflow specification. This has been one of assumptions for the thesis. This approach is relatively simple and suitable for many environments that perform *several* workflows. However, there are situations when there is a lot of workflows only slightly changing along one definition. With the former approach, we would need to make number of workflow definitions that could be run just several times or even once.

Such environments require another approach for definition of workflows that would allow some changes of workflows during their execution. There has been mentioned possibility to use security classification for deciding which actions to perform next, but the idea has not been developed. It is an area that is very promising and it would be very interesting to study possibilities implied from application of authorization system, especially with combination of inheritance as introduced in Chapter 8.

9.2.3 Web-based Implementation

This is last, but not least domain that has not been solved. In the moment, there is no implementation that would demonstrate properties of the proposed authorization system and its applicability. It is a problem that was not in my power to solve. The problem is that to demonstrate security model of workflow system, we have to implement the workflow system first.

I suppose that ideal environment is Internet. Existing technologies solve problems with communication, some basic problems with authentication and so on. The interface is accessible from any web-browser and system implemented in Java is portable. The problem would be an ideal theme for some MSc-thesis; as implementation of simple workflow system as implementation of workflow manager and other parts of the authorization system.

9.2.4 Cryptography

We have not assumed use of cryptography at all, although its necessity is clear when implementing any large secure system. The main reason that determines usage of cryptography is assumption that heterogeneous system is implemented on non-secure media - e.g. Internet. We have to design secure mechanisms for logging into system and also mechanisms for secure communication among secure s-nodes. I do not suppose that there is necessary to search some original solutions of arising problems but combination of needed mechanisms for the particular area may be very interesting.

Index

access control
  classification, 16
  discretionary, 12
  local, 38
  mandatory, 13
  role-based, 15
  s-node, 4
  workflow, 5
action, 20
  completed, 71
  perfect, 20, 71
  secure, 61
  silent, 25
action-sequence
  secure, 75
agent, 19, 21, 73
  constant, 20
  expression, 20
  inactive, 21
  secure
    finite, 91
    serial, 91
  variable, 20
attack
  trojan horse, 13
authorization
  abstraction, 65, 66
  application, 51
  cooperative, 12
  counted enabled, 65
  dynamic, 40
  e-complete, 59
  hierarchical decentralized, 11
  information, 65
  initial, 52
  operator, 64
  ownership, 11
  specialization, 65, 66
  static, 40
authorization system
  active, 40
  passive, 40
authorization template
  abstract, 51
  specific, 51
authorization unit, 48
  execution, 54
  validity, 55
availability, 2

bisimilar
  inheritance, 98
  securely, 84
  strongly, 31
bisimilarity, 33
bisimulation, 32
  inheritance, 98
  secure
    weak, 84
  strong, 30
  weak, 32

categorization, 50, 60
category, 60
class
  categorization, 60
  classification, 60
  security, 60
classification, 60

equational laws, 24
state, 61, 73
co-name, 20, 70
Coalition
Workflow Management, 8
combination
classification class, 68
combinators
dynamic, 24
static, 24
confidentiality, 1
congruent
securely (observation-), 89

data
protection, 48
derivation tree, 22
partial, 22
total, 22
derivative
immediate, 22
secure, 75
secure immediate, 75

equal
securely, 89
equivalence
observation, 33
strong secure, 78
experiment, 88

flow control
workflows, 5
flow graph, 27
flow unit, 48
form
standard, 92
standard concurrent, 26
function
classification, 61
relabelling, 20

information system
autonomous, 4
inheritance
life-cycle, 99
projection, 99
protocol, 99
total, 99
integrity, 1

label, 20, 70
law
equational, 24
laws
monoid, 24
relabelling, 29
restriction, 29
tau, 25
layer
conversion, 39

manager
atomic task, 40
workflow, 40

name, 20, 70

port, 20, 71
secure, 71
privilege
counted, 65
enabled, 48
initial, 48
process
a-complete, 62
e-complete, 62
w-complete, 62
protection data, 55
Protocol
Simple Workflow Access, 8

reference monitor, 13
relation
flow, 68
resource

category, 42

s-node, 4, 37, 45
security class, 60
security controls, 3
security level, 14
security mechanism
external, 12
internal, 12
security policy, 11
security requirements, 2
sort, 23
syntactic, 23
state
secure, 72
substitution
inheritance, 103
system
labelled transition, 74
secure, 66
strongly distributed, 45
workflow management, 1
system node, 45

task
atomic, 39
instance, 46
template, 46
task step, 48
time interval, 49
time mark, 49

workflow, 1, 39

Bibliography

[1] Ifip wg 11.3 database security. http://www.cs.rpi.edu/ifip/.

[2] Department of defense trusted computer system evaluation criteria. Tech. Rep. DoD5200.28-STD, Department of Defense, 1985.

[3] *Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions* (Athens, GA, 1996).

[4] Aalst, W.: A class of petri net for modeling and analyzing business processes. Computer science report, Eindhoven University of Technology, Eindoven, 1995.

[5] Aalst, W.: Three good reasons for using a petri-net-based workflow management system. In Proc. of the International Working Conference on Information and Process Integration in Enterprises, 1996, pp. 179–201.

[6] Aalst, W.: How to handle dynamic change and capture management information? an approach based on generic workflow models. Tech. Rep. UGA-CS-TR-99-01, University of Georgia, Department of Computer Science, Athens, USA, 1999.

[7] Aalst, W.: Inheritance of workflow processes: Four problems, one solution? In Proceedings of the Colloquium on Petri Net Technologies for Modelling Communication Based Systems, 1999, pp. 1–28.

[8] Aalst, W., Basten, T.: Inheritance of workflows - an approach to tackling problems related to change. Tech. Rep. 99/06, Eindhoven University of Technology, Eindhoven, 1999. to appear in Theoretical Computer Science.

[9] Ammann, P., Jajodia, S., Ray, I.: Ensuring atomicity of multilevel transactions. In Proceedings of the IEEE Symposium on Security and Privacy, 1996.

[10] Atluri, V., Huang, W.-K.: An extended petri net model for supporting workflows in a multilevel secure environment. In Proc. of the 10th IFIP WG11.3 Working Conference on Database Security, 1996, pp. 240–258.

[11] Atluri, V., Huang, W.-K.: Enforcing mandatory and discretionary security in workflow management systems. *Journal of Computer Security*, 5(4):303–339, 1997.

[12] Baraani-Dastjerdi, A., Peprzyk, J., Safavi-Naini, R.: Security in databases: A survey study. Tech. Rep. TR-96-2, Department of Computer Science, The University of Wollongong, 1996.

[13] Baraani-Dastjerdi, A., Peprzyk, J., Safavi-Naini, R.: A security model for multi-level object-oriented databases based on views. Tech. Rep. TR-96-3, Department of Computer Science, The University of Wollongong, 1996.

[14] Barkley, J.: Comparing simple role based access control models and access control lists. Tech. rep., NIST, Gaithersburg MD 20899, 1997.

[15] Basten, T., Aalst, W.: Inheritance of behaviour. Tech. Rep. 99/17, Eindhoven University of Technology, Eindhoven, 1999. to appear in the Journal of Logic and Algebraic Programming.

[16] Bell, D., LaPadula, L.: Secure computer systems: Unified exposition and multics interpretation. Tech. Rep. MTR-2997, The Mitre Corporation, Bedford, MA, 1976.

[17] Bergstra, J., Klop, J.: Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[18] Bertino, E., Ferrari, E., Atluri, V.: A flexible model supporting the specification and enforcement of role-based authorizations in workflow management systems. In Proc. of the 2nd ACM Workshop on Role-based Access Control, 1997.

[19] Bertino, E., Jajodia, S., Samarati, P.: Supporting multiple access control policies in database systems. In Proceedings of the IEEE Symposium on Security and Privacy, 1996, pp. 94–107.

[20] Bertino, E., Samarati, P., Jajodia, S.: High assurance discretionary access control in object bases. In Proc. First ACM Conference on Computer and Communication Security, 1993, pp. 140–150.

[21] Boebert, W., Ferguson, C.: A partial solution to the discretionary trojan horse problem. In Proc. of the 9th National Computer Security Conference, 1985, pp. 141–144.

[22] Burkhard, H.: Observations on the real-world implementation of role-based access control. In 20th National Information Systems Security Conference, 1997.

[23] Castalo, S., Fugini, M., Martella, G., Samarati, P.: *Database Security.* Addison-Wesley & ACM Press, 1995.

[24] Clark, D., Wilson, D.: A comparison of commercial and military computer security policies. In Proc. IEEE Symposium on Security and Privacy, 1987, pp. 184–194.

[25] Cvrcek, D.: Access control in database management system. In DATASEM'98 sborník přednášek, 1998, pp. 153–163.

[26] Cvrcek, D.: Nové přístupy k databázové bezpečnosti. In Sborník z letní školy Informační systémy a jejich aplikace, 1998, pp. 18–35.

[27] Cvrcek, D.: Problems of modeling access control in object oriented databases. In ASIS 98 Proceedings, 1998, pp. 21–26.

[28] Cvrcek, D.: Active authorization as high-level control. In Proc. of the IFIP WG 11.3 Workshop on Database Security, 2000, pp. –.

[29] Cvrcek, D.: Mandatory access control in workflow systems. In Knowledge-based Software Engineering - Proc. of the JCKBSE Conference, 2000, pp. 247–254.

[30] Demurjian, S., Ting, T., Price, M., Hu, M.-Y.: Extensible and reusable role-based object-oriented security. In Database Security, X: Status and Prospects, Chapman Hall, 1997.

[31] Denning, D.: *Secure Information Flow In Information Systems.* Phd thesis, University of Purdue, 1975.

[32] Denning, D.: A lattice model of secure information flow. *Communications of the ACM*, 236–243, 1976.

[33] Denning, D.: *Cryptography and Data Security.* Addison-Wesley, 1982.

[34] Denning, D., Schlorer, J.: Inference controls for statistical databases. *IEEE Computer*, 16(2), 1983.

[35] FENCOTT, C.: *Formal Methods for Concurrency.* International Thomson Computer Press, 1996.

[36] FERNANDEZ, E., SUMMERS, R., WOOD, C.: *Database Security and Integrity.* Addison-Wesley, 1981.

[37] FERRAIOLO, D., CUGINI, J., KUHN, R.: Role-based access control: Features and motivations. In Proceedings of the 11th Annual Computer Security Applications Conference (CSAC '95), 1995.

[38] HOLLINGSWORTH, D.: Workflow reference model. Tech. Rep. TC00-003, Workflow Management Coalition, Winchester, UK, 1995.

[39] HUANG, W.-K.: *Incorporating Security into Workflow Management Systems.* PhD thesis, Rutgers University, CIMIC, 1998.

[40] ITSEC: Information technology security evaluation criteria, provisional harmonised criteria, 1991.

[41] KAMATH, M., RAMAMRITHAN, K.: Correctness issues in workflow management. *Distributed Systems Engineering (DSE) Journal : Special Issue on Workflow Management Systems*, 3(4), 1996.

[42] KARGER, P.: Limiting the damage potential of discretionary trojan horses. In Proc. IEEE Symposium on Security and Privacy, 1987, pp. 32–37.

[43] KUHN, D.: Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In Proceedings of the 2nd ACM Workshop on Role-Based Access Control (RBAC-97), ACM Press, 1997, pp. 23–30.

[44] LÖWE, M., WIKARSKI, D., HAN, Y.: Higher-order object nets and their application to workflow modeling. Forschungsbericht 95-34, FB Informatik, TU Berlin, 1995.

[45] MILNER, R.: *Communication and Concurrency.* Prentice Hall, 1989.

[46] MURATA, T.: Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[47] OLIVIER, M., SOLMS, S. V.: A taxonomy for secure object-oriented databases. *ACM Transactions on Database Systems*, 3–46, 1994.

[48] PETRI, C.: *Kommunikation mit Automaten.* Phd. thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.

[49] Qian, X., Lunt, T.: A mac policy framework for multilevel relational databases. *IEEE Transactions on Knowledge and Data Engineering*, 8(1):1–14, 1996.

[50] Rabitti, F., Bertino, E., Kim, W., Woelk, D.: A model of authorization for next-generation database systems. *ACM Transactions on Database Systems*, 1(16):88–131, 1991.

[51] Ramamritham, K., Chrysanthis, P.: A taxonomy of correctness criteria in database applications. *International Journal on Very Large Data Bases*, 4(1):181–293, 1996.

[52] Reisig, W.: *Petri Nets: An Introduction*, vol. 4 of *EATCS monographs on Theoretical Computer Science*. Springer, Berlin, Germany, 1985.

[53] Sandhu, R.: Terminology, criteria and system architectures for data integrity. In Proc. of the Invitational Workshop on Data Integrity, NIST Special Publication 500-168, 1989, pp. A.4 1–14.

[54] Sandhu, R.: Evaluation by parts of trusted database management systems. In Proc. 4th RADC Workshop on Multilevel Database Security, 1991, pp. 122–126.

[55] Sandhu, R.: On five definitions of data integrity. In Proc. of the IFIP WG 11.3 Workshop on Database Security, 1993.

[56] Sandhu, R.: Access control: The neglected frontier. In Proc. First Australian Conference on Information Security and Privacy, 1996.

[57] Sandhu, R.: Role hierarchies and constraints for lattice-based access controls. In Proc. Fourth European Symposium on Research in Computer Security, 1996.

[58] Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control: A multi-dimensional view. In Proc. of 10th Annual Computer Security Applications Conf., 1994, pp. 54–62.

[59] Sandhu, R., Feinstein, H.: A three tier architecture for role-based access control. In Proc. of the 17th NIST-NCSC National Computer Security Conference, 1994, pp. 138–149.

[60] Sandhu, R., Jajodia, S.: Honest databases that can keep secrets. In Proc. of the 14th NIST-NCSC National Computer Security Conference, 1991, pp. 267–282.

[61] Sandhu, R., Suri, G.: Implementation considerations of the typed access matrix model in a distributed environment. In Proc. of the 15th NIST-NCSC National Computer Security Conference, 1992, pp. 221–235.

[62] Sandhu, R. S.: Separation of duties in computerized information systems. In Database Security, IV: Status and Prospects, 1991, pp. 179–189.

[63] Sandhu, R. S., et al.: Role-based access control models. *IEEE Computer*, 38–47, 1996.

[64] Sandhu, R. S., Thomas, R.: Task-based authorization controls (TBAC): A family of models for active and enterprise-oriented authorization management. In Proc. of the IFIP WG 11.3 Workshop on Database Security, 1997.

[65] Thomas, R., Sandhu, R.: Implementing the message filter object-oriented security model without trusted subjects. In Proc. of the IFIP WG 11.3 Workshop on Database Security, 1992.

[66] Thomas, R., Sandhu, R.: Discretionary access control in object-oriented databases: Issues and research directions. In Proc. of the 16th NIST-NCSC National Computer Security Conference, 1993, pp. 63–74.

[67] Thomas, R., Sandhu, R.: Task-based authorization: A paradigm for flexible and adaptable access control in distributed applications. In Proc. of the 16th NIST-NCSC National Computer Security Conference, 1993, pp. 409–415.

[68] Thomas, R., Sandhu, R.: Towards the unified framework and theory for reasoning about security and correctness of transactions in multilevel databases. In Proc. of the IFIP WG 11.3 Workshop on Database Security, 1993.

[69] Thomas, R., Sandhu, R.: Conceptual foundations for a model of task-based authorizations. In Proc. of the 7th IEEE Computer Security Foundations Workshop, 1994, pp. 66–79.

[70] van der Aalst, W.: A class of petri net for modeling and analyzing business processes. Computer science report 95/26, Eindhoven University of Technology, Eindhoven, 1995.

[71] Walter, K., Ogden, W., Rounds, W., Bradshaw, F., Ames, S., Sumaway, D.: Primitive models for computer security. Tech. rep., Case Western Reserve University, Cleveland, USA, 1974.