BRICS
Basic Research in Computer Science

BRICS RS-03-39 J. Abendroth: Applying π-Calculus to Practice: An Example of a Unified Security Mechanism

Applying π-Calculus to Practice: An Example of a Unified Security Mechanism

Jörg Abendroth

BRICS Report Series RS-03-39
ISSN 0909-0878 November 2003

Applying π-Calculus to Practice: An Example of a Unified Security Mechanism

Joerg Abendroth*
Distributed Systems Group, Trinity College Dublin

October 2003

Abstract

The π-calculus has been developed to reason about behavioural equivalence. Different notions of equivalence are defined in terms of process interactions, as well as the context of processes. There are various extensions of the π-calculus, such as the SPI calculus, which has primitives to facilitate security protocol design.
Another area of computer security is access control research, which includes problems of access control models, policies and access control mechanism. The design of a unified framework for access control requires that all policies are supported and different access control models are instantiated correctly.
In this paper we will utilise the π calculus to reason about access control policies and mechanism. An equivalence of different policy implementations, as well as access control mechanism will be shown. Finally some experiences regarding the use of π-calculus are presented.

1 Introduction

The π-calculus has been developed to reason about behavioural equivalence. One notion of equivalence may take into account the process interactions, such as the channels a process is capable to send or receive information.
The work of Abadi and Gordon proposed the SPI-calculus [2], an extension of the π-calculus with cryptographical primitives. Using the spi-calculus security protocols can be shown to fail in the presence of a certain attacker. Hereby, the important application of the SPI-calculus is that the correctness of security protocols can be expressed as statements of behavioural equivalence.
Research in access control is concerned with access control models, policies

*This work is conducted while the author visited BRICS University of Aarhus, Danmark supported by EU IHP 'Marie Curie Fellowship' HPMT-CT-2000-00093.

and access control mechanism. Development of a specialised access control model allows efficient access decisions in novel computer usage scenarios.
Research in access control policies aims at a comprehensive policy description language. Both, access control models and policies build on the research of access control mechanism, which combines results of authentication protocols with results of other areas to provide mechanism to evaluate the access control decision function correctly. A simple approach is to design an access control mechanism for each security model or policy uniquely. First approaches to provide a unifying mechanism can be seen in the Kerberos project [6] or the certificate based systems like the SPKI[1] [34]. A recent approach combines ideas of active capabilities [33] with the ones of proxy-based authorisation [23, 15]. While a prototype implementation [4] has shown positive experiences, the final framework design should be guided by formal methods such as a specification in the π-calculus.
In this paper we present the design of a unified access control mechanism able to instantiate different kind of policies and can be summarised as follows:

- Different access control policies can be compared and shown to be equivalent
- The behaviour of the unified framework does not influence the access control decision
- The basic and extended version of the unified framework can be shown to be behavioural equivalent to a simple conventional access control mechanism.

The remainder of this paper is organised as follows, after discussing related works in section 2. We will describe the goals of a unified access control mechanism in section 3. The final part of the introduction is the π-calculus formal foundation (section 4), which includes syntax, behavioural equivalences, reduction rules and proof system.
The rest of the paper can be seen as two applications of the π-calculus. Section 5 describes the ACL[2] access control model using an informal policy. The policy will be represented as an access matrix and the process expressions to describe the model derived. After presenting a second version of the process expression description we will validate in section 5.5 the process expressions. Finally it will be shown that both process expressions demonstrate the same behaviour by showing a bisimulation in section 5.6.
The second example is the process expressions of different versions of the AS-Cap unified access control mechanism. Section 6 explains how the process expressions are derived. Section 6.3.1 shows behavioural equivalence of the

[1]Simple Public Key Infrastructure
[2]Access Control List

proxy based setup to a hybrid approach. Section 6.3.2 does the same for the external security server based version. After discussion the lessons learned (section 6.4), section 7 gives conclusion and advise for future work.

2 Related Work

Most papers about the π-calculus focus on variants of the language and proof techniques. The SPI calculus [2] can be seen as an example. It originates in research on authentication protocols [21] with the desire to formalise proof of their correctness.
Other papers discus the use of the π-calculus driven by the need to apply the π-calculus to practice. One paper discusses the representation and simulation of biochemical processes [1]. In this paper Regev, Silverman and Shapiro demonstrate how to standardise the encoding of protein networks to be able to share and manipulate the body of existing knowledge. The syntax is the main feature used for describing the biomolecular processes. Further properties of the calculus such as the different equivalences were not studied.
Padget and Bradford employ the π-calculus to model the spanish fish marked, a well known example of multi agent environment. Here [26], different process expressions and their derivation are presented. Unfortunately a final comment suggests that there are still components missing, due to the complexity of the scenario. It is left open whether e.g. potential cheating loopholes could be detected during a simulation, but their application provides valuable insight for our work.
Esterline and Rorie [5] investigates how far the π-calculus could be used to model NASA's LOGOS[3] multiagent system. 11 different components were identified and a specification scenario presented. After identifying the different communication paths the process expressions of each component is given, they derive a development methodology in which the equational congruence testing is used to prove refinements up to the actual implementation being behavioural equivalent to the original specification.
Some other research in the domain of π-calculus is worth mentioning, such as Pierce's PICT programming language [11], which allows to write programs in a π-calculus like syntax. The use of PICT allows automatically generated applications from specifications used to investigate certain behavioural properties. A stepwise refinement as proposed by Esterline and Rorie would not be necessary, thus saving the proof work.
Victor's Mobility Workbench [7, 35](MWB) automates the search for bisimulations for given process expressions. The MWB can be seen as a prototype

[3]Lights Out Ground Operations System

for verification systems in the direction of Smolka et all, [28] work on the use of formal methods to find configuration vulnerabilities in a simplified Unix system. Although it seems they considered the π-calculus in early stages they decided to develop XMC [27], their own tool-set.
In the domain of access control research our work has been influenced by Jajodia's "unified framework for multiple access control policies" [20], which derives an authorisation specification language and flexible authorisation manager to derive access control. Evaluation of correct access control model implementation is left to correct a formulation of the model in the authorisation language.
Olivier's research [24] is lead by the desire to provide unified access control framework and is concentrated mostly on the operation system level. A formal verification technique is not described in the available literature. Here we propose to use the π-calculus formal methods with the aim of showing behavioural equivalence to classical access control mechanism implementations.

3 Specification of a Unified Access Control Mechanism

Research in access control has proposed various access control models [19, 25, 10, 32, 13, 30]. Often each of these models is implemented using a special enforcement mechanism. Part of a successful research includes the technology transfer toward real life use in industry. However in cases where specialised access control mechanisms are required, this can cumbersome. Today most applications are provided with a quasi standard enforcement mechanism providing only a very limited single access control model.
We aim to design an access control enforcement mechanism, which is able to support various access control models. This will allow to select the most suitable access control model at employ time, including the most recent ones. Access control model developer may take advantage of the unified framework by providing their model to a wider user base compared to the use of a customised enforcement mechanism.
Finally, a unified enforcement mechanism allows to detach application development from the security part following the ideas of hidden-software capabilities [15] and component-based programming [29].

3.1 Requirements

A unified access control mechanism will be used by different parties, each of them has certain expectations from the mechanism.

Client Interface

The client does not want to be occupied with more tasks then required by a conventional access control mechanism of the same model. This includes, in cases where different external security servers are employed, that the client is not required to manually set up channels and connections to any of those servers.

- Unified mechanism and customised mechanism interface has to be indistinguishable
- No extra information or channel setup is required

Server Interface

In a customised implementation the server will implement the access control model (decision function) directly, while in a unified framework the implementation will be provided by means of the unified access control mechanism. The server needs to provide a unique interface towards the access decision function called policy in the following. There are requirements that the server has to have a way to check the trustworthiness of the policy, while the correctness may be checked similar to a customised mechanism implementation.

- Provide a channel to interact for each part of the access decision function
- Be able to check the trustworthiness of each part of the access decision function

Policy Interface

The policy in our mechanism design represents the access control model. Later in this paper we will show, that all policies can be written as π-calculus process expressions and therefore employed in the unified access control mechanism. A policy may have different input channels, as well as internal channels to transfer state information. A policy may only have two output channels, which represent grant and deny as the access control decision result.

- Access control mechanism may not influence the policy behaviour
- Internal channels of policy parts distributed onto different external security server shall be secret to the outside.

It shall be possible to split a policy into different separate processes, which interact using internal channels. Each of these processes may be controlled by an independent power, thus practically partially outsourcing [3] the access control decision function onto different entities.
Another desired property of a unified access control mechanism is the possibility of dynamically changing the access control policy, but because the formal specification is considered with only one access request case (ie. snapshot moment) this notion of dynamic policy change is not captured.

General Requirements

As a general requirement onto a unified access control mechanism it can be seen that the performance overhead should be virtually non-existent. The security level and whole system behaviour should be the same as a comparative conventional customised access control mechanism. It is a fact that a more complex implementation is like to yield more bugs and implementation errors, part of them may be observable using techniques of software verification (ie. chapter four in [18]) together with the process expressions presented in this paper and refinement technique described in [5]:

- System behaviour is equivalent to a comparative conventional customised access control mechanism

4 Formal Foundation

The task is to prove that a certain access control mechanism design does not influence the upper lying access control security model. It can be done in different ways. The use of classical Hoare triples [17] would allow to verify that certain properties of programs hold. The number of properties and type would cause the full proof to be complex and very specific. Another approach combined logical reasoning with calculus for communication systems (CCS) to capture the security properties, while benefit from the ease of expressing the communication between different system entities. The idea of having an object (ie. policy) in one scope (ASCap proxy) and only after some communications being accessible by other objects (such as server) would, however, be particularly hard to encode in the CCS.
This brought us to the π-calculus, which was developed to reason about the behaviour of concurrent communication and mobile systems. The notion of restriction and scope, turned out to be ideal to express our dynamic system.

4.1 The π-Calculus

In the following we will give a definition of reduction rules and axioms required in the later of this paper. We will follow the notation of Sangiorgi and Walker [31] and benefit from further insights of Milners introduction [22]. All theorems were taken out of Sangiorgi and Walker's book [31].

4.2 Syntax

Let us write $L(0, \pi, +, =, |, \nu)$ for the language of terms given by

$$P ::= 0 \quad | \quad \pi.P \quad | \quad P + P' \quad | \quad \varphi P P' \quad | \quad P|P' \quad | \quad \nu z P$$

where the prefixes are given by

$$\pi ::= \overline{x}y \quad | \quad x(z) \quad | \quad \tau$$

and the conditions by

$$\varphi ::= [x = y] \quad | \quad \neg\varphi \quad | \quad \varphi \wedge \varphi'.$$

Here $\overline{x}y$ means sending y on the channel x; x(z) refers to receiving a value on channel x, which is then bound to z and τ is the internal transition. Finally to ease notation we abbreviate $(\nu z)\overline{x}z$ by $\overline{x}(z)$.

4.3 Labeled Transition Relations

Each system described by a *process expression* can transit into other process expression by the transition relations given below. We will use the late transition rules, which are distinguished by the time when the placeholder z is instantiated. The instantiates occurres late, when the *communication* is inferred, rather than when the *input* by the receiver is inferred.

Definition 4.1 Late transition relations *The late transition relations,* $\{\overset{\alpha}{\longmapsto} |\alpha \in \pi\}$*, are defined by the rules in Table 1.*

We shall define an additional transition rules for φPQ (mismatch operator).

$$MISM1: \quad \frac{P \overset{\alpha}{\longmapsto} P' \quad [|\varphi|] = true}{\varphi PQ \overset{\alpha}{\longmapsto} P'}$$

$$MISM2: \quad \frac{Q \overset{\alpha}{\longmapsto} Q' \quad [|\varphi|] = false}{\varphi PQ \overset{\alpha}{\longmapsto} Q'}$$

The Late Transition Rules

$$\text{L-Out}\ \frac{}{\overline{x}y.P\overset{\overline{x}y}{\longmapsto}P} \qquad \text{L-Inp}\ \frac{}{x(z).P\overset{x(z)}{\longmapsto}P}$$

$$\text{L-Tau}\ \frac{}{\tau.P\overset{\tau}{\longmapsto}P} \qquad \text{L-Mat}\ \frac{\pi.P\overset{\alpha}{\longmapsto}P'}{[x{=}x]\pi.P\overset{\alpha}{\longmapsto}P'}$$

$$\text{L-Sum-L}\ \frac{P\overset{\alpha}{\longmapsto}P'}{P{+}Q\overset{\alpha}{\longmapsto}P'}$$

$$\text{L-Par-L}\ \frac{P\overset{\alpha}{\longmapsto}P'}{P|Q\overset{\alpha}{\longmapsto}P'|Q}\ bn(\alpha)\cap fn(Q)=0$$

$$\text{L-Comm-L}\ \frac{P\overset{\overline{x}y}{\longmapsto}P' \qquad Q\overset{x(z)}{\longmapsto}Q'}{P|Q\overset{\tau}{\longmapsto}P'|Q'\{y/z\}}$$

$$\text{L-Close-L}\ \frac{P\overset{\overline{x}(z)}{\longmapsto}P' \qquad Q\overset{x(z)}{\longmapsto}Q'}{P|Q\overset{\tau}{\longmapsto}\nu z(P'|Q')}$$

$$\text{L-Res}\ \frac{P\overset{\alpha}{\longmapsto}P'}{\nu zP\overset{\alpha}{\longmapsto}\nu zP'} z\notin n(\alpha) \qquad \text{L-Open}\ \frac{P\overset{\overline{x}z}{\longmapsto}P'}{\nu zP\overset{\overline{x}(z)}{\longmapsto}P'}\ z\neq x$$

$$\text{L-Rep-Act}\ \frac{P\overset{\alpha}{\longmapsto}P'}{!P\overset{\alpha}{\longmapsto}P'|!P}$$

$$\text{L-Rep-Comm}\ \frac{P\overset{\overline{x}y}{\longmapsto}P' \qquad P\overset{x(z)}{\longmapsto}P''}{!P\overset{\tau}{\longmapsto}(P'|P''\{y/z\})|!P}$$

$$\text{L-Rep-Close}\ \frac{P\overset{\overline{x}(z)}{\longmapsto}P' \qquad P\overset{x(z)}{\longmapsto}P''}{!P\overset{\tau}{\longmapsto}(\nu z(P'|P''))|!P}$$

Table 1: The late transition rules

4.4 Behavioural Equivalence

The basic equivalence is the structural congruence, which is defined as:

Definition 4.2 Structural Congruence Structural congruence, *written $\equiv$, is the process congruence over* P *determined by the following equations:*

1. *Change of bound names (alpha-conversion)*
2. *Reordering of terms in a summation*
3. $P|0 \equiv P$,$P|Q \equiv Q|P$,$P|(Q|R) \equiv (P|Q)|R$
4. $\nu a(P|Q) \equiv P|\nu aQ$ *if* $a \notin fn(P)$
5. $\nu a0 \equiv 0$, $\nu abP \equiv \nu baP$

In the literature other forms of behavioural equivalence have been described. We are going to restriction us to the form of weak late congruence, which has a stronger notion than weak late bisimilarity.

Definition 4.3 Weak late bisimilarity *Weak late bisimilarity is the largest symmetric relation, $\approx_l$, such that whenever $P \approx_l Q$,*

1. *$P \overset{x(z)}{\longmapsto} P'$ implies there is Q' such that $Q \overset{x(z)}{\longmapsto} Q'$ and $P'\{y/z\} \approx_l Q'\{y/z\}$ for every y*

2. *if α is not an input action then $P \overset{\alpha}{\longmapsto} P'$ implies $Q \overset{\alpha}{\longmapsto}\approx_l P'$.*

Definition 4.4 Weak late congruence *P and Q are weak late congruent, $P \approx_l^c Q$, if $P\sigma \approx_l Q\sigma$ for every substitution σ.*

4.5 Proof System

We also record and name the rules of equational reasoning for this language:

$$\begin{array}{ll} REFL & P = P \\ SYMM & P = Q \quad implies \quad Q = P \\ TRANS & P = Q \quad and \quad Q = R \quad implies \quad P = R \\ SUM & P = Q \quad implies \quad P + R = Q + R. \end{array}$$

To reason about bisimilarity of summations we add axioms saying that + is associative,commutative, idempotent, and 0 has an identity:

$$\begin{array}{lrcl} S1 & (P+Q)+R & = & P+(Q+R) \\ S2 & P+Q & = & Q+P \\ S3 & P+0 & = & P \\ S4 & P+P & = & P. \end{array}$$

Now consider late bisimilarity on $L(0, \pi, +, =)$. For the input prefix we need to separate out the sound part,

$$PRE1 \quad P = Q \quad implies \quad \pi.P = \pi.Q \quad if \pi\ is\ not\ of\ the\ form\ x(z),$$

and introduce the rule:

$$PRE2 \quad P\{y/z\} = Q\{y/z\} \quad for\ all \quad y \in fn(P, Q, z) \quad implies \quad x(z).P = x(z).Q.$$

For conditions we have to formulate the simple congruence rule

$$C4 \quad P \ = \ Q \quad implies \quad \varphi P = \varphi Q.$$

Then we have an axiom that allows us to replace a condition by an equivalent one

$$C5 \quad \varphi P \ = \ \varphi' P \quad if \varphi \Leftrightarrow \varphi'$$

and four axioms for conditional forms:

$$\begin{array}{lrcl} C6 & \neg[x=x]P & = & \neg[x=x]Q \\ C7 & \varphi PP & = & P \\ C8 & \varphi PQ & = & \neg\varphi QP \\ C9 & \varphi(\varphi' P) & = & (\varphi \wedge \varphi')P \end{array}$$

Then we have a kind of distribution axiom involving conditions and summations:

$$C10 \quad \varphi(P+P')(Q+Q') \;=\; \varphi PQ + \varphi P'Q'$$

And finally, we have two axioms concerning conditions and prefixing, namely

$$\begin{array}{lrcl} C11 & \varphi(\pi.P) & = & \varphi\pi.\varphi P \\ C12 & [x=y](\pi.P) & = & [x=y](\pi\{y/x\}).P \end{array}$$

We need some axioms for expanding compositions and for manipulating restrictions. Consider expansion first. The axiom will be called **E**:
If $P = \sum_i \varphi_i\pi_i.P_i$ and $P' = \sum_j \varphi'_j\pi'_j.P'_j$, then

$$\begin{array}{rcl} P|P' & = & \sum_i \varphi_i\pi_i.(P_i|P') + \sum_j \varphi'_j\pi'_j.(P|P'_j) + \\ & & \sum_{\pi_i \text{opp} \pi'_j} (\varphi_i \wedge \varphi'_j \wedge [x_i = x'_j])\tau.R_{ij} \end{array}$$

where $\pi_i$opp[4]π'_j if

(1) π_i is $\overline{x_i}y$ and π'_j is $x'_j(z)$, when R_{ij} is $P_i|P'_j\{y/z\}$,or

(2) π_i is $\overline{x_i}(z)$ and π'_j is $x'_j(z)$, when R_{ij} is $\nu z(P_i|P'_j)$,

or vice versa. Now we can introduce the rules for restrictions:

$$RES \quad P = Q \quad implies \quad \nu zP = \nu zQ$$

$$\begin{array}{lrcl} RES1 & \nu z\nu wP & = & \nu w\nu zP \\ RES2 & \nu z(P+Q) & = & \nu zP + \nu zQ \\ RES3 & \nu z\pi.P & = & \pi.\nu zP \quad if\, z \notin n(\pi) \\ RES4 & \nu z\pi.P & = & 0 \quad \text{if } \pi \text{ is } z(w) \text{ or } \overline{z}y \text{ or } \overline{z}(w) \\ RES5 & \nu z[x=y]P & = & [x=y]\nu zP \quad \text{if } x,y \neq z \\ RES6 & \nu z[z=y]P & = & 0 \quad \text{if } y \neq z. \end{array}$$

Then to obtain axiomatisations for *weak* late congruence on $L(0, \pi, +, \varphi)$, it suffices to add the following axioms

$$\begin{array}{lrcl} TAU1 & \pi.\tau.P & = & \pi.P \\ TAU2 & \tau.P + P & = & \tau.P \\ TAU3 & \pi.(P+\tau.Q) & = & \pi.(P+\tau.Q) + \pi.Q. \end{array}$$

Finally, an axiom for replication:

$$REP \quad !(P|Q) = (!P)|(!Q)$$

Let LD be the collection of axioms and rules: REFL, SYMM, TRANS, SUM, S1-S4, C4-C12, E, RES, RES1-RES6, TAU1, TAU2, TAU3 and REP.

[4]read opposes

Theorem 4.1 *LD is an axiomatisation for weak late congruence of the terms in $L(0, \pi, +, \varphi, |, \nu)$.*

The proofs for the theorems and rule applications can be found in Sangiorgi and Walker [31].

5 Access Control Policies

In this part we will use the π-calculus to express the behaviour of the simple ACL[5] access control model. To enhance the practical relevance we will start with an informal description of the rule, draw some conclusions to understand the model and derive the process expressions in the last step.

5.1 Informal Policy

In a company user might be given numerical user ids as usernames and be divided into three groups with different permissions. The informal policy, formulated by the security administrator, reads as follows:
Odd numbered users have only r right, even number user rw rights, if the user id is dividable by four the user gets rwx rights. Our system only uses the accounts with id 1 to 5.

5.2 Access Matrix

From the informal policy above an access matrix as described by Bell [8] can be built. Each row summarises the rights a user has, while each column shows all permissions to the corresponding object. Intersecting cells hold the specific access of the user to the object. Below an access matrix of the example policy is shown:

User	Object 1	...	Object n
u1	r		
u2	rw		
u3	r		
u4	rwx		
u5	r		

5.3 Process Expression Version 1

To derive the process expression for the given policy, one may first determine the row (userid) and then the access rights. This would result in the process

[5] Access Control List

expression below:

$$
\begin{aligned}
P_{acl1} &\overset{def}{=} l(u).[u=1]U_1([u=2]U_2([u=3]U_3([u=4]U_4([u=5]U_5D))))\\
U_1 = U_3 = U_5 &\overset{def}{=} l(p).[p=r]GD\\
U_2 &\overset{def}{=} l(p).[p=r]G([p=w]GD)\\
U_4 &\overset{def}{=} l(p).[p=r]G([p=w]G([p=x]GD))\\
G &\overset{def}{=} \bar{l}grant\\
D &\overset{def}{=} \bar{l}deny + l().D
\end{aligned}
$$

This process receives information on the channel l, which can be understood as a login channel. After some transitions the result (grant or deny) is transmitted on the l channel. P_{acl1} is the start state of the policy process, U_1 etc. are the states according to the userid, G can be understood as a Grant state, and D as deny. In D the process may receive additional information on $l()$, which will be ignored, this behaviour is required as an invalid user id might result in a default denial regardless of the right requested.

5.4 Process Expression Version 2

A different implementation might derive the process expression directly from the informal description, thus resulting in the following process:

$$
\begin{aligned}
P_{acl2} &\overset{def}{=} (\nu k)(S_{accountcheck}|S_{rightcheck})\\
S_{accountcheck} &\overset{def}{=} l(u).[u=1]\overline{k}o([u=2]\overline{k}en4(\\
&\quad [u=3]\overline{k}o([u=4]\overline{k}e4([u=5]\overline{k}oD))))\\
S_{rightcheck} &\overset{def}{=} k(g).[g=o]U_o(\\
&\quad [g=en4]U_{en4}([g=e4]U_{e4}D))\\
U_o &\overset{def}{=} l(p).[p=r]GD\\
U_{en4} &\overset{def}{=} l(p).[p=r]G([p=w]GD)\\
U_{e4} &\overset{def}{=} l(p).[p=r]G([p=w]G([p=x]GD))\\
G &\overset{def}{=} \bar{l}grant\\
D &\overset{def}{=} \bar{l}deny + l().D
\end{aligned}
$$

Note that the indices refer to the following meanings:o= odd (ie.in respect to an imaginary userid), en4=even, but not dividable by 4, e4=even and dividable by 4. $S_{accountcheck}$ can be understood as translating a userid into a group (g)(or role like in RBAC [14]), while $S_{rightcheck}$ decides the permissions.

5.5 Validating the Process Expressions

For validation rule-test expressions can be written, which compliment a correct policy implementation and send out a passed to the environment upon completion. Some rule-test expressions are given below:

$$\begin{aligned} R_1 &\overset{def}{=} \bar{l}1.\bar{l}r.l(z).[z = grant]\overline{passed} \\ R_2 &\overset{def}{=} \bar{l}1.\bar{l}w.l(z).[z = deny]\overline{passed} \\ R_3 &\overset{def}{=} \bar{l}6.\bar{l}r.l(z).[z = deny]\overline{passed} \end{aligned}$$

One test would be to compose a test system with one of the rule-test expression and the policy process expression, and verify that such a system should be reducible to a single output $\overline{passed}$.
Note, that it can be shown that all possible combinations (rule-test,policy expression) will reduce correctly, but for space reasons we will print out only the following two systems:

1. $T_1 = P_{acl1}|R_1 \rightarrow^* \overline{passed}$
2. $T_2 = P_{acl2}|R_3 \rightarrow^* \overline{passed}$

5.5.1 $T_1 = P_{acl1}|R_1 \rightarrow^* \overline{passed}$

In the following we give the reductions used.

$$\begin{aligned} l(u).[u = 1]U_1([u = 2]U_2(\quad &| \quad \bar{l}1.\bar{l}r.l(z).[z = grant]\overline{passed} \\ [u = 3]U_3([u = 4]U_4([u = 5]U_5D)))) & \end{aligned}$$

$\boxed{\text{REACT (Communication on } l).}$

$$\begin{aligned} [1 = 1]U_1([1 = 2]U_2(\quad &| \quad \bar{l}r.l(z).[z = grant]\overline{passed} \\ [1 = 3]U_3([1 = 4]U_4([1 = 5]U_5D)))) & \end{aligned}$$

$\boxed{\text{MISM1 and expand } U_1.}$

$$l(p).[p = r]GD \quad | \quad \overline{l}r.l(z).[z = grant]\overline{passed}$$

$\boxed{\text{REACT (Communication on } l).}$

$$[r = r]GD \quad | \quad l(z).[z = grant]\overline{passed}$$

$\boxed{\text{MISM1 and expand } G.}$

$$\overline{l}grant \quad | \quad l(z).[z = grant]\overline{passed}$$

$\boxed{\text{REACT (Communication on } l).}$

$$0 \quad | \quad [grant = grant]\overline{passed}$$

$\boxed{\text{Ignoring empty process and MISM1.}}$

$$\overline{passed} \quad \blacksquare$$

5.5.2 $T_2 = P_{acl2}|R_3 \to^* \overline{passed}$

Reduction rules from **??** are given for each step.

$$(\nu k)(S_{accountcheck}|S_{rightcheck}) \quad | \quad \overline{l}6.\overline{l}r.l(z).[z = deny]\overline{passed}$$

$\boxed{\text{Expanding } S_{accountcheck} \text{ and } S_{rightcheck}.}$

$$\begin{aligned}(\nu k)(l(u).[u = 1]\overline{k}o([u = 2]\overline{k}en4(\quad &| \quad \overline{l}6.\overline{l}r.l(z).[z = deny]\overline{passed}\\ [u = 3]\overline{k}o([u = 4]\overline{k}e4([u = 5]\overline{k}oD))))| &\\ k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k &\end{aligned}$$

$\boxed{\text{REACT (Communication on } l).}$

$$\begin{aligned}(\nu k)([6 = 1]\overline{k}o([6 = 2]\overline{k}en4(\quad &| \quad \overline{l}r.l(z).[z = deny]\overline{passed}\\ [6 = 3]\overline{k}o([6 = 4]\overline{k}e4([6 = 5]\overline{k}oD))))| &\\ k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k &\end{aligned}$$

$\boxed{\text{MISM2.}}$

$$\begin{aligned}(\nu k)([6 = 2]\overline{k}en4(\quad &| \quad \overline{l}r.l(z).[z = deny]\overline{passed}\\ [6 = 3]\overline{k}o([6 = 4]\overline{k}e4([6 = 5]\overline{k}oD)))| &\\ k(g).[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k &\end{aligned}$$

$\boxed{\text{4x MISM2 and expand } D.}$

$$(\nu k)(\bar{l}deny + l.D|k(g). \quad | \quad \bar{l}r.l(z).[z = deny]\overline{passed}$$
$$[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

$\boxed{\text{REACT (Communication on } l\text{) and expand } D.}$

$$(\nu k)(\bar{l}deny + l.D|k(g). \quad | \quad l(z).[z = deny]\overline{passed}$$
$$[g = o]U_o([g = en4]U_{en4}([g = e4]U_{e4}D)))_k$$

$\boxed{\text{REACT (Communication on } l\text{).}}$

$$(\nu k)(0|k(g).[g = o]U_o([g = en4] \quad | \quad [deny = deny]\overline{passed}$$
$$U_{en4}([g = e4]U_{e4}D)))_k$$

$\boxed{\text{MISM1.}}$

$$(\nu k)(0|k(g).[g = o]U_o([g = en4] \quad | \quad \overline{passed}$$
$$U_{en4}([g = e4]U_{e4}D)))_k$$

$\boxed{\text{Because } k \text{ is the only reduction possible, but } k \text{ is not known outside of the scope, the process and restriction can be ignored. This argument goes along RES4 of the proof system.}}$

$$\overline{passed} \quad \blacksquare$$

5.6 Showing Behavioural Equivalence

A validation as done in the previous section may show similar behaviour in the context of certain test expressions. For achieving confidence that both expressions correctly depict the informal policy it is necessary to show that both processes are weakly late bisimilar.
First we have to notice that version 2 consists of two concurrent process communicating by the bound channel k. $S_{rightcheck}$ has no other means to be reduced then by initially receiving on k. $S_{accountcheck}$ final action is sending on k. If we are able to convince ourself, that $\tau_{[u=1]}$ in $S_{accountcheck}$ (see section 5.4) strictly leads to $\tau_o \overset{def}{=} \overline{k}o|k(g).[g = o]$ (see figure 2), we can show that $P_{acl1} \approx P_{acl2}$ using the bisimulation S.

$$\begin{aligned} S \;=\; & ((P_{acl1}, P_{acl2}), (P'_{acl1}, P'_{acl2}), (U_1, U_o), (U_1, U''_o), (U_3, U_o), (U_3, U''_o) \\ & , (U_5, U_o), (U_5, U''_o), (U_2, U_{en4}), (U_2, U''_{en4}), (U_4, U_{e4}), (U_4, U''_{e4}), (U'_1, U'_o), \\ & (U'_3, U'_o), (U'_5, U'_o), (U'_2, U'_{en4}), (U'_4, U'_{e4}), (D, D), (G, G)). \end{aligned}$$

Using the tree diagrams given below, it is possible to verify the correctness of the bisimulation. Please note that leaves, which result into the same state, such as D or U_o are cut for a better overview.

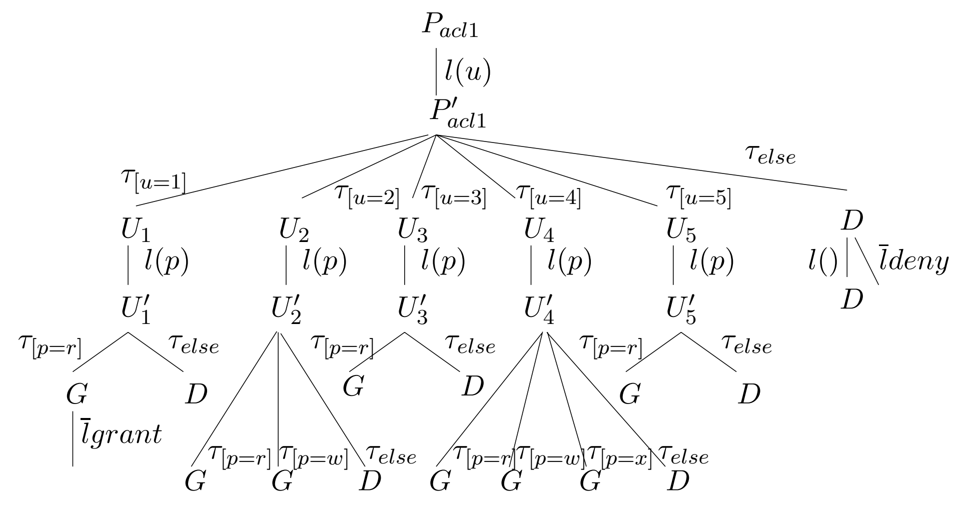

Figure 1: Tree of policy version 1

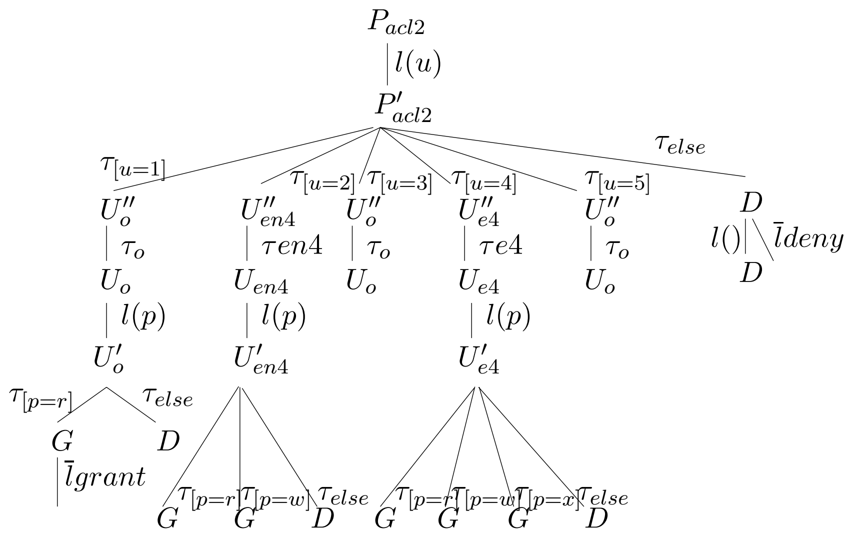

Figure 2: Tree of policy version 2

5.7 Conclusion and Lessons of First Part

In this part we have shown how to derive process expressions from the an informal access control policy definition. To convince ourself of the correct behaviour we wrote test expressions and verified that each system is reducible to the proposed outcome. This validates each system, but may not capture hidden differences in behaviour. Therefore we showed that the two systems are weakly late bisimilar.
The results suggest two things: it is possible to express access models in the π-calculus by using the mismatch construct, and one can reason about different models and their behavioural equivalence. In the presented systems we found the bisimulation "by hand", but a automated tool, such as the mobility workbench [7] can be helpful for bigger systems.
Access control models can be classified into three categories: The access models, the information flow models and the new area of trust based models. The ACL model presented belongs to the first category. Hennessy and Riely's work [16] suggest that also information flow models can be expressed using the π-calculus.
An open problem is how trust based models can be expressed using the π-calculus. Carbone, Nielsen and Sassone are currently undertaking work [9] on developing a process-calculus with trust values.

6 Process Expressions of the Unified Access Control Mechanism

The π-calculus can also be used to describe the behaviour of a unified access control mechanism. In the first section of this part general thoughts about practical systems will be presented. Then in section 6.2 the actual process expressions will be derived. Section 6.3.1 and 6.3.2 show the equivalence using the proof system presented in 4. In section 6.4 an analysis of the mechanism and proof will conclude this chapter.

6.1 A Unified Access Control Mechanism

In section 3 the specification of a unified access control mechanism was given. The specification does not require specific implementation details, which will be discussed in this section.
An access control mechanism may have three different general forms

1. Hybrid

2. Proxy Based

3. External Security Server Based

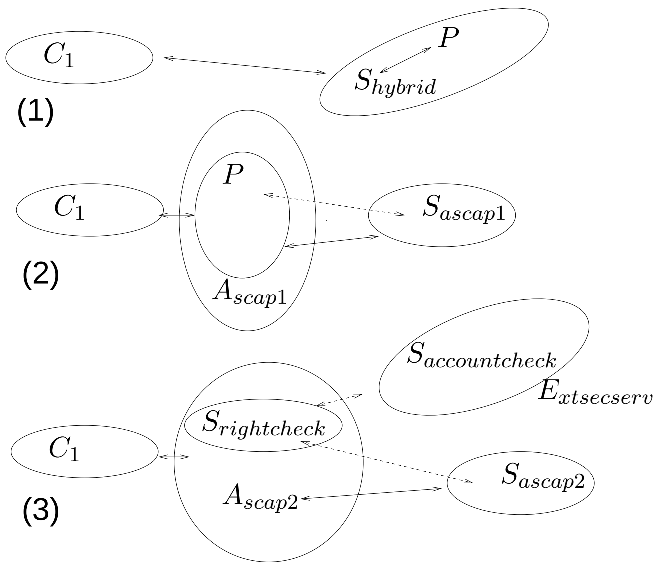

Figure 3: The different access control mechanism.

Figure 3 shows an abstract view of the three different systems. C_1 represents the Client on the left side, and the server is situated on the right. A_{scap} is the access control mechanism proxy in the middle area of the figure. Note that in (3) in Figure 3 $(S_{rightcheck}|S_{accountcheck} = P_{acl2}) \approx P_{acl}$ using the bisimulation of Section 5.6.

Hybrid

A hybrid access control mechanism can be seen as the basic form. The full policy and authentication implementation is situated in the server, the client access the server directly. Security in this system can be shown by verifying the server implementation. This can be identified as the current practice, which requires a customised implementation of the access control mechanism in each server.

Proxy Based

In a proxy based system the server and client are separated by a security mechanism proxy, which handles authentication and policy object selection. The server does not have the policy, but gets a references (or the whole executable signed code object) from the proxy. Security depends not only on the implementation, but integrity of the policy and trust into its source is required. In return the advantage of flexibility and dynamic policy adaption is gained.

External Security Server Based

The proxy based system can be extended by allowing the policy to be split into different parts, which will be hosted on different entities (named external security servers). Such a system is required to behave equivalently to a hybrid system with policy parts residing in the same server.

In the following section we will derive process expressions for each of these systems, and we will see the hybrid system as a measurement for the correct system behaviour. Therefore our proof will show behavioural equivalence of the two extended versions to the simple hybrid version.

6.2 Deriving the π-Calculus Expressions

In the following we will derive process expressions for a hybrid, seen to be conventional system, the client process expression and the two unified access control mechanism.

6.2.1 Hybrid System

$$S_{hybrid} \overset{def}{=} (\nu l)(\overline{c}l|P)$$

The server has the policy implementation P (e.g. P_{acl1}), as well as the l channel to access it in its private scope. Upon an access request the server extends the scope of the l channel to the client. The policy may have a format like that described in section 5 taking the userid and requested right as input and sending a grant or deny on the same channel as output. A system including a full authentication, use of cryptography like described by Abadi et all [2] and a compute process on the server side is also deceivable, but will be neglected for our behaviour study.

6.2.2 Client Process

$$C_1 \overset{def}{=} c(n).\overline{n}1.\overline{n}r.n(x).[x = grant]\overline{passed}$$

The client process expression is alike a test expression in the first part. The difference is that first a new channel name is received, which will be used as login channel.

6.2.3 Proxy Based Setup

This is the first of two unified framework setups.

ASCap Proxy in Proxy Based

The proxy based setup consists of a server and an ASCap Proxy (see [4] for an introduction to the roots of ASCap). The ASCap proxy consists of three components.

- Secure Server Connection Setup
- Policy-Server Interface
- Client-Server Interface

A secure connection is setup to an available server $(\nu m)(\overline{c_{sa}}m.\overline{m}c_{sp}.\overline{m}c_{sl})$. The channel c_{sa} is used to provide the two communication channels (sl=Server login-information; sp=Server Policy). The restriction of m can be implemented i.e. by a fresh symmetric key. The policy implementation residing in the AS-Cap proxy needs to be accessible by the server $((\nu l)(\overline{c_{sp}}(l)|P))$. In practice the scope extrusion can be implemented in different ways, either the server receives a reference to the policy implementation (i.e. residing on a secure repository) or a signed full policy object (mobile code) can be transfered. This aspect is also described by Milner in [22].
Finally the login information from the client interface needs to be made accessible to the server $(c_{sl}(b).\overline{c}b)$. The full ASCap proxy process expression reads as follows:

$$A_{scap} \overset{def}{=} (\nu c_{sp}, c_{sl})\,[(\nu m)(\overline{c_{sa}}m.\overline{m}c_{sp}.\overline{m}c_{sl}) \quad | \quad (\nu l)(\overline{c_{sp}}l|P) \quad | \quad (c_{sl}(b).\overline{c}b)]$$

Server in Proxy Based Setup

In the proxy based setup the server process expression needs to adapt to the policy and client connection being provided by the ASCap proxy.

$$S_{ascap1} \overset{def}{=} c_{sa}(d).d(p).d(q).p(a).\overline{q}a$$

The sever receives the fresh channels on c_{sa} and via the new channel p and q. Then in the simplest case the server provides the client (q) with the login channel received from the policy (p). This means a new channel between the client and policy is created by the server.

6.2.4 External Security Server Based

This is the second unified framework. It is an extension of the proxy based setup using external security servers to further out source part of the policy behaviour.

ASCap Proxy in External Security Server Based

The ASCap Proxy in the external security server based setup is extended with the external security server communication part. In an implementation this would require additional connection setup code and signature checking. Core process is $(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}h.h(b).\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')$, which provides a login information channel to the client $\overline{c}h$ receives information on it $h(b)$ and redirects it ($c_{ess}(a).h(b)$ and $\overline{a'}b$) to either the $S_{accountcheck}$ part of the policy (resided in the external security server), or $S_{rightcheck}$ ($c_{sl}(d')$ and $h(b').\overline{d'}b'$). Further on the two policy parts require to interact on a private channel k, which is handled by $c_{ess}(a').c_{sl}(d)$ and $a(x).\overline{d}x$. It becomes obvious, that once a policy has a different interaction pattern (i.e. requires more secret channels, or different login information at different points), this part of the ASCap proxy has to be adapted. Therefore in the following the proof of equivalent behaviour can only be given for the class of policies of the form $S_{rightcheck}$ of form $l'(x).\cdots.\overline{k'}$ and $S_{rightcheck}$ of form $k(y).\cdots.l(z)$. The full ASCap proxy process expression is given below:

$$\begin{aligned}A_{scap2} \overset{def}{=} \; & (\nu c_{sl}, c_{sp})((\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}h.h(b).\\ & \overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')_h| \\ & (\nu l, k)(\overline{c_{sp}}k.\overline{c_{sp}}l|S_{rightcheck})|\overline{c_{sa}}c_{sp}.\overline{c_{sa}}c_{sl})_{c_{sl},c_{sp}}\end{aligned}$$

External Security Server in External Security Server Based

An external security server has a form similar to the policy interface part of an ASCap proxy.

$$E_{xtsecserv} \overset{def}{=} (\nu l, k)(\overline{c_{ess}}k.\overline{c_{ess}}l|S_{accountcheck})$$

Server in External Security Server Based

The particularity, that the policy is split into two parts is also reflected in the server process expression.

$$S_{ascap2} \overset{def}{=} c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a$$

Generally the server will require to handle as many channels as policy parts in the system exists.

6.3 Concurrent Systems

This section gives an introduction about the expressiveness of the π-calculus by presenting different concurrent setups of the ASCap framework. It incorporates different servers and clients, which can be expressed by the replication operator.

$$(\nu c)(C_{lient}|!A_{scap})|!S_{ascap1a}|S_{ascap1b}$$

c_{lient} may be c_1 (see section 6.2.2) or another client, which issues several access requests. A_{scap} is bound statically, i.e. by storing it locally, into the client scope - only they share c. The replication is used to allow the client several access requests each one handled by an ASCap proxy instance. At the same time there are either several independent servers. These servers $(S_{ascap1a}|S_{ascap1b})$ can have different internal behaviour, such as different data sources for a weather forecast. The same server can take

several access requests in parallel $(!S_{ascap1a})$. Like in section 5.5 it can be validated, that once the client commits to one server no interference with the replicas of other server process expressions can take place.

$$c_{c1}|(\nu c_{SA})!(A_{scap}|S_{scap})|(\nu c_{SA1})!(A_{scap}|S_{scap1b})$$

In this case the client can access one of the ASCap proxy, i.e. by downloading them. Each ASCap proxy is associated with one server - they share a secret channel c_{SA}, this scope extrusion could be implemented by public key cryptography. In this example the client restricts the server he will use by the ASCap proxy he downloads at the beginning.

Further scenario can be simulated using the replication, but in the following we will retreat to a single entity of each process.

6.3.1 Showing Behavioural Equivalence Of Hybrid and Version 1

In this section the process expressions will be rewritten using the axiomatisation system of section 4 to show weak late congruence. First the process expressions are copied for ease of reading:

$$A_{scap} \overset{def}{=} (\nu c_{sp}, c_{sl})((\nu m)((\overline{c_{sa}}(m).\overline{m}c_{sp}.\overline{m}c_{sl}) \quad | \quad (\nu l)(\overline{c_{sp}}(l)|P_{acl1}) \quad | \quad (c_{sl}(b).\overline{c}b)$$

$$S_{ascap1} \overset{def}{=} c_{sa}(d).d(p).d(q).p(a).\overline{q}a$$

Then we would like to introduce an axiom to put a policy P out of a scope of a restriction:

$$SA \quad (\nu l)(\overline{c_{sp}}(l)|P) \approx_l^c (\nu l)(\overline{c_{sp}}(l).P) \; if \; P \; of \; form \; l(x).\cdots$$

To proof this axiom it has to be noted, that P can not do a transition before $\overline{c_{sp}}(l)$ extends the scope of l thus allowing other expressions to communicate with P.

In the following the commented sequence of equations are given:

$(\nu c_{sa})(A_{scap} \quad | \quad S_{ascap1})$

$$\boxed{\text{We restrict } c_{sa} \text{ as we only consider the case in which this } A_{scap} \text{ and } S_{ascap1} \text{ use } c_{sa} \text{ to communicate with each other instead of with other processes. This restriction can be implemented using public key cryptography. Note we are using the abbreviated form for the restriction } (\nu m).}$$

$$(\nu c_{sa})((\nu c_{sp}, c_{sl}) \quad (\quad \overbrace{(\overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl}))}^{A} \,|\, \overbrace{(\nu l)(\overline{c_{sp}}(l)|P_{acl1}) \quad | \quad (c_{sl}(b).\overline{c}b))}^{B} \,| \\ \overbrace{c_{sa}(d).d(p).d(q).p(a).\overline{q}a}^{C})$$

$$\boxed{\begin{array}{l}\text{It has to be kept in mind that every } (\nu c_{sp}, c_{sl}) \text{ in the expansion is the same as in the wider scope.} \\ \text{Using E on } (\nu c_{sp}, c_{sl})(B|\overbrace{A)|C}^{1} \text{ with} \\ P = \overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl}) \\ \text{and } P' = c_{sa}(d).d(p).d(q).p(a).\overline{q}a.\end{array}}$$

$$\begin{aligned} 1)P|P' \quad = \quad & (\overline{c_{sa}}(m).(\overline{m}(c_{sp}).\overline{m}(c_{sl})|c_{sa}(d).d(p).d(q).p(a).\overline{q}a) + \\ & c_{sa}(d).(\overline{c_{sa}}(m).\overline{m}(c_{sp}).\overline{m}(c_{sl})|d(p).d(q).p(a).\overline{q}a) + \\ & ([c_{sa} = c_{sa}])\tau(\nu m)(\overbrace{\overline{m}(c_{sp}).\overline{m}(c_{sl})|m(p).m(q).p(a).\overline{q}a)}^{2}) \end{aligned}$$

$$\boxed{\begin{array}{l}\text{After this expansion it can be seen that the system can proceed in three ways. Either the proxy establishes a con-} \\ \text{nection with an arbitrary server (by doing } \overline{c_{sa}}(m)\text{); Or the server receives an connection from a different proxy } (c_{sa}(a)); \\ \text{or both interact with each other. As we restricted } c_{sa} \text{ we can rule out the first two cases by using RES4,RES2 and RES1.} \\ \text{Now we apply E to 2 with } P = \overline{m}(c_{sp}).\overline{m}(c_{sl}) \\ \text{and } P' = m(p).m(q).p(a).\overline{q}a\ .\end{array}}$$

$$\begin{aligned} 2)P|P' \quad = \quad & \overline{m}(c_{sp}).(\overline{m}(c_{sl})|m(p).m(q).p(a).\overline{q}a) + \\ & m(p).(m(q).p(a).\overline{q}a|\overline{m}(c_{sp}).\overline{m}(c_{sl})) + \end{aligned}$$

$$\tau.(\nu c_{sp})\overbrace{(\overline{m}(c_{sl})|m(q).c_{sp}(a).\overline{q}a)}^{3}$$

Although the expansion results in three nondeterministic choices, the upper two can be removed by RES4 and RES2 thus we expand (3) further.
The (νc_{sp}) is the same as in wider scope.
E with $P = \overline{m}(c_{sl})$ and $P' = m(q).c_{sp}(a).\overline{q}a$.

$$\begin{aligned} 3)P|P' \quad = \quad & \overline{m}(c_{sl}).(0|m(q).c_{sp}(a).\overline{q}a) + \\ & m(q).(c_{sp}(a).\overline{q}a|\overline{m}(c_{sl})) + \\ & \tau.(\nu c_{sl})(0|c_{sp}(a).\overline{c_{sl}}a) \end{aligned}$$

The upper two choices can be removed using RES4, RES2 and RES1. We need to recall, that E in 1) used the form $(\nu c_{sp}, c_{sl})(B|A)|C$ with $A|C$ like (1(2(3))). We rewrite the original term now as:

$$\begin{aligned} (\nu c_{sp}, c_{sl}) \quad (\quad & (\nu l)(\overline{c_{sp}}l|P_{acl1}) \quad | \quad (c_{sl}(b).\overline{c}b) \quad | \\ & \tau.(\nu m)\tau.(\nu c_{sp})\tau.(\nu c_{sl})(0|c_{sp}(a).\overline{c_{sl}}a)) \end{aligned}$$

We remember that the scope of the restrictions $(\nu c_{sp}, c_{sl})$are the same as in the wider scope, hence we don't need to write it twice . We apply RES3 and RES4 to remove (νm). TAU1 can be used to remove the τ.

$$(\nu c_{sp}, c_{sl}) \quad (\quad (\nu l)(\overline{c_{sp}}(l)|P_{acl1})|(c_{sl}(b).\overline{c}b) \quad | \quad c_{sp}(a).\overline{c_{sl}}a)$$

Before we can apply the E rule we need to put P_{acl1} out of the scope of (νl). This can be done by applying SA of above. E can now be used with $P = \overline{c_{sp}}(l).P_{acl1}$ and $P' = c_{sp}(a).\overline{c_{sl}}a$.

$$\begin{aligned} 4)P|P' \quad = \quad & (\overline{c_{sp}}(l).(P_{acl1}|c_{sp}(a).\overline{c_{sl}}a + \\ & c_{sp}(a).(\overline{c_{sp}}(l).P_{acl1}|\overline{c_{sl}}a + \\ & \tau.(\nu l)(P_{acl1}|\overline{c_{sl}}(l))) \end{aligned}$$

Upper two are 0 according to RES4,RES2 and RES1.
After applying TAU1 and SA use E rule with $P = \overline{c_{sl}}(l).P_{acl1}$ and $P' = c_{sl}(b).\overline{c}b$.

$$\begin{aligned} 5)P|P' \quad = \quad & \overline{c_{sl}}(l).(P_{acl1}|c_{sl}(b).\overline{c}b) + \\ & c_{sl}(b).(\overline{c_{sl}}(l).P_{acl1}|\overline{c}b) + \\ & \tau.(\nu l)(P_{acl1}|\overline{c}(l))) \end{aligned}$$

Upper two are 0 according to RES4,RES2 and RES1. τ removed by TAU1. Then use (4) and (5) to rewrite the original term and remove $(\nu c_{sp}, c_{sl})$ by RES3 and RES4, we get:

$$(\nu l) \qquad \overline{c}(l)|P_{acl1}$$

Recalling that $S_{hybrid} \overset{def}{=} (\nu l)(\overline{c}l|P_{acl1})$ we can now see that

$$(\nu l)\overline{c}(l)|P_{acl1} \approx_l^c (\nu l)(\overline{c}l|P_{acl1})$$

means

$$A_{scap}|S_{ascap1} \approx_l^c S_{hybrid}.$$

This result can be lifted to

$$!A_{scap}|!S_{ascap1} \approx_l^c !S_{hybrid}$$

by showing that the initial communication on c_{sa} between A_{scap} and S_{ascap1} extends the scope of the fresh m (and later c_{sp}, c_{sl}). m, c_{sp}, c_{sl} being secret prevents instances of A_{scap} and S_{ascap1}, which are in the same stage of communication to interfere with each other.

Informally that means that infinite parallel running instances of the A_{scap} proxy and infinite simultaneous running instances of the S_{ascap1} are weak late congruent with infinite number of S_{hybrid} waiting for communication, provided both use the same policy P as access decision function.

This result holds because in the equational reformulation we were only using rules which do not modify weak late congruence. However we have to keep in mind that after using SA, our system is restricted to policies, which start with an input action on channel l before freely interacting with the environment. We feel that this limitation is of no burden to the practical policy appliance.

6.3.2 Showing Behavioural Equivalence Of Hybrid and Version 2

This section shows behavioural equivalence of the external security server based system described by the process expressions given before:

$$\begin{aligned} A_{scap2} \overset{def}{=} \; & (\nu c_{sl}, c_{sp})((\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.a(x).\overline{d}x. \\ & h(b').\overline{d'}b')_h|(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck}) \quad | \\ & \quad \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} \end{aligned}$$

$$E_{xtsecserv} \overset{def}{=} (\nu l, k)(\overline{c_{ess}}(k).\overline{c_{ess}}(l)|S_{accountcheck})$$

$$S_{ascap2} \overset{def}{=} c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a$$

In the equations below the braces are augmented with indices noting the scope of restrictions ending by this brace. Also partial expressions are omitted, if it can be shown that they can be reduced to the empty process. Section 6.3.1 can be used to see examples of the full terms.

$$\begin{array}{rcl}
(\nu c_{sa})((\nu c_{ess})(A_{scap2}|E_{xtsecserv}) & | & S_{ascap2}) \\
(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})(& | & c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}} \\
(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b). & & \\
\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')_h| & & \\
(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})| & & \\
\overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} & & \\
(\nu l, k)(\overline{c_{ess}}(k).\overline{c_{ess}}(l)|S_{accountcheck}))_{c_{ess}} & &
\end{array}$$

$$\boxed{\text{Renaming } l \text{ \& } k \text{ .}}$$

$$\begin{array}{rcl}
(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})(& | & c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}} \\
(\nu h)(c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b). & & \\
\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b')_h| & & \\
(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})| & & \\
\overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} & & \\
(\nu l', k')(\overline{c_{ess}}(k').\overline{c_{ess}}(l')|S_{accountcheck}))_{c_{ess}} & &
\end{array}$$

$$\boxed{\begin{array}{l}\text{E with } P' = \overline{c_{ess}}(k').\overline{c_{ess}}(l') \text{ and } P = \\ c_{ess}(a).c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.a(x).\overline{d}x.h(b').\overline{d'}b' \text{ .)}\end{array}}$$

$$\begin{array}{r}
P|P' = (c_{ess}(a).\cdots+\overline{c_{ess}}(k').\cdots+ \\
\tau. \\
(\nu k')c_{ess}(a').c_{sl}(d).c_{sl}(d'). \\
\overline{c}(h).h(b).\overline{a'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'|\overline{c_{ess}}(l'))
\end{array}$$

$$\boxed{\begin{array}{l}\text{The first two choices can be reduced to 0 by RES4, RES2 and RES1.} \\ \text{E with } P = c_{ess}(a').c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{a'}b.k'(x).\overline{d}x.h(b').\overline{d'}b' \text{ and} \\ P_j = \overline{c_{ess}}(l').0.\end{array}}$$

$$\begin{array}{r}
P|P' = (c_{ess}(a').\cdots+\overline{c_{ess}}(l').\cdots+ \\
\tau. \\
(\nu l')c_{sl}(d).c_{sl}(d'). \\
\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b'|0)
\end{array}$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1. Hence the whole expression after further simplifications becomes:

$$\begin{array}{rcl}
(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})((\nu h)(& & \\
(\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). & | & c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a)_{c_{sa}} \\
\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h| & & \\
S_{accountcheck})_{l',k'}| & & \\
(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck})| & & \\
\overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl}))_{c_{sl},c_{sp}} & &
\end{array}$$

E with $P = \overline{c_{sa}}(c_{sp}).\overline{c_{sa}}(c_{sl})$ and $P' = c_{sa}(p).c_{sa}(q).p(a).\overline{q}a.p(a).\overline{q}a$.

$$\begin{aligned}
P|P' = \overline{c_{sa}}(c_{sp}).\cdots + c_{sa}(p).\cdots +& \\
\tau.(\nu c_{sp})\overline{c_{sa}}(c_{sl})|& \\
c_{sa}(q).c_{sp}(a).\overline{q}a.c_{sp}(a).\overline{q}a)&
\end{aligned}$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1. E with $P = \overline{c_{sa}}(c_{sl})$ and $P' = c_{sa}(q).c_{sp}(a).\overline{q}a.c_{sp}(a).\overline{q}a$.

$$\begin{aligned}
P|P' = \overline{c_{sa}}(c_{sl}).\cdots + c_{sa}(q).\cdots +& \\
\tau.(\nu c_{sl})0|& \\
c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a)&
\end{aligned}$$

The first two choices can be reduced to 0 by RES4, RES2 and RES1. After further simplifications the full expression becomes.

$$\begin{array}{rcl}
(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp})((\nu h)(& & \\
(\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). & | & \tau.\tau.c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a)_{c_{sl},c_{sp}})_{c_{sa}} \\
\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h| & & \\
S_{accountcheck})_{l',k'}| & & \\
(\nu l, k)(\overline{c_{sp}}(k).\overline{c_{sp}}(l)|S_{rightcheck}) & &
\end{array}$$

E with $P = \overline{c_{sp}}(k).\overline{c_{sp}}(l)$ and $P' = c_{sp}(a).\overline{c_{sl}}a.c_{sp}(a).\overline{c_{sl}}a$.

$$\begin{aligned}
P|P' = \overline{c_{sp}}(k).\cdots + c_{sp}(a).\cdots +& \\
\tau.& \\
(\nu k)(\overline{c_{sp}}(l)|\overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k)&
\end{aligned}$$

The first two choices can be removed using RES4, RES2 and RES1. Hence the full term reads:

$$(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k)((\nu h)($$

$$\begin{array}{rcl} (\nu l', k')(\tau.\tau.c_{sl}(d).c_{sl}(d'). & | & \tau.\tau.\tau\overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k))_{c_{sl},c_{sp},k})_{c_{sa}} \\ \overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b')_h| & & \\ S_{accountcheck})_{l',k'}| & & \\ (\nu l)(\overline{c_{sp}}(l)|S_{rightcheck}) & & \end{array}$$

$$\boxed{\text{E with } P = c_{sl}(d).c_{sl}(d').\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{d}x.h(b').\overline{d'}b' \text{ and } P' = \overline{c_{sl}}(k).c_{sp}(k).\overline{c_{sl}}(k).}$$

$$\begin{array}{r} P|P' = c_{sl}(d).\cdots+\overline{c_{sl}}(k).\cdots+ \\ \tau.(\nu k)c_{sl}(d').\overline{c}(h). \\ h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b'| \\ c_{sp}(k).\overline{c_{sl}}(k)) \end{array}$$

$$\boxed{\begin{array}{l}\text{The first two choices can be removed using RES4, RES2 and RES1.} \\ \text{Hence the full term reads:}\end{array}}$$

$$\begin{array}{rcl} (\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k)((\nu h)(& & \\ (\nu l', k')(\tau.\tau.\tau.c_{sl}(d'). & | & \tau.\tau.\tau c_{sp}(k).\overline{c_{sl}}(k))_{c_{sl},c_{sp},k})_{c_{sa}} \\ \overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b')_h| & & \\ S_{accountcheck})_{l',k'}| & & \\ (\nu l)(\overline{c_{sp}}(l)|S_{rightcheck}) & & \end{array}$$

$$\boxed{\text{E with } P = c_{sp}(k).\overline{c_{sl}}(k) \text{ and } P' = \overline{c_{sp}}(l).}$$

$$\begin{array}{r} P|P' = c_{sp}(k).\cdots+\overline{c_{sp}}(l).\cdots+ \\ \tau.(\nu l)\overline{c_{sl}}(l)|0) \end{array}$$

$$\boxed{\begin{array}{l}\text{The first two choices can be removed using RES4, RES2 and RES1.} \\ \text{Hence the full term reads:}\end{array}}$$

$$\begin{array}{rcl} (\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k, l)((\nu h)(& & \\ (\nu l', k')(\tau.\tau.\tau.c_{sl}(d'). & | & \tau.\tau.\tau.\tau\overline{c_{sl}}(l))_{c_{sl},c_{sp},k,l})_{c_{sa}} \\ \overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b')_h| & & \\ S_{accountcheck})_{l',k'}| & & \\ S_{rightcheck}) & & \end{array}$$

$$\boxed{\text{E with } P = c_{sl}(d').\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{d'}b' \text{ and } P' = \overline{c_{sl}}(l).}$$

$$\begin{array}{r} P|P' = c_{sl}(d').\cdots+\overline{c_{sl}}(l).\cdots+ \\ \tau.(\nu l)\overline{c}(h).h(b). \\ \overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b'|0) \end{array}$$

The first two choices can be removed using RES4, RES2 and RES1. Hence the full term reads:

$$\begin{aligned}&(\nu c_{sa})((\nu c_{ess})((\nu c_{sl}, c_{sp}, k, l)((\nu h)(\\ &\qquad (\nu l', k')(\tau.\tau.\tau.\tau. \quad | \quad \tau.\tau.\tau.\tau.0)_{c_{sl},c_{sp},k,l})_{c_{sa}}\\ &\overline{c}(h).h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b')_h|\\ &S_{accountcheck})_{l',k'}| \quad S_{rightcheck})\end{aligned}$$

Using RES1-RES4 some restrictions can be removed. TAU1 removes the τ.

$$\begin{aligned}&(\nu k, l)((\nu h)((\nu l', k')(\overline{c}(h).\\ &h(b).\overline{l'}b.k'(x).\overline{k}x.h(b').\overline{l}b')_h \quad | \quad S_{accountcheck})_{l',k'}|S_{rightcheck})_{k,l}\end{aligned}$$

$$\begin{aligned}(\nu k, l)((\nu h)((\nu l', k') \quad (\quad &\overbrace{\overline{c}(h).h(b).\overline{l'}b}^{A}\,.\,\overbrace{k'(x).\overline{k}x}^{B}\,.\,\overbrace{h(b').\overline{l}b'}^{C})_h|\\ &S_{accountcheck})_{l',k'}|S_{rightcheck})_{k,l}\end{aligned}$$

Notable in this process expression is, that $S_{accountcheck}$ and $S_{rightcheck}$ have no direct communication channel, ie. they reside in different servers. From section 5.4 we know that $S_{accountcheck}$ requires communication on l' before it does internal communication on k'. The part A receives the login information from the client and sends it to $S_{accountcheck}$. B transfers the internal information from $S_{accountcheck}$ to $S_{rightcheck}$ and C provides $S_{rightcheck}$ with the next information received from the client.
If $S_{accountcheck}$ has the form $l'(x).\cdots.\overline{k'}$ and $S_{rightcheck}$ has the form $k(y).\cdots.l(z)$ then the system can be shown to be $\approx_l^c S_{hybrid}$ using the bisimulation given in section 5.6.

6.4 Conclusion and Lessons of Second Part

In this section the π-calculus specifications of a unified access control framework has been discussed. After giving the process expression of a hybrid system, which stands for today's customised access control mechanism built into each proprietary server, two versions of a unified access control mechanism were presented. The first version (proxy based) introduced dynamic policy change by migrating the location were the policy is stored. The second version extended the setup by external security servers, which allows the policy to be split into parts, to be stored and administrated by different entities. The section concludes with two proofs that both frameworks do not change the behaviour of the full system. This is a substantial result as the gained

flexibility comes at no costs.
The process expression of an exemplified client is also given, which allows the reader to verify that from the client perspective the systems appear indistinguishable. This is important as existing clients, once the ASCap unified access control mechanism is interfaced, can use any kind of mechanism without changing their interface. Hence new access control models could be employed without modifying the client application.

6.4.1 Lessons

Writing the process expressions involves understanding both the π-calculus and the idea of the framework to be described. An interesting question was whether certain properties could not be encoded. Obvious candidates were performance properties, cryptographical algorithms or deterministic choices in the policy expressions.In the following we will discuss different encoded properties.

Performance Properties

Performance properties can be seen as parallel to the communication behaviour and would require further primitives in the π-calculus. It is possible to guess from the amount of label transitions and τ transitions whether a process might have a overhead, but further encoding would be needed.

Security Properties

Security properties, such as cryptographical algorithms are similar to the performance properties - additional primitives in the π-calculus are needed. These had been provided in the SPI-calculus by Abadi [2].

Deterministic Choices

Deterministic choices is not included in the simple π-calculus, but the polyadic form provides the match and mismatch operator. Further variants include choices on priority or types.

Online Modification

In real life applications it is conceivable that a server receives login information of the client in form of an authentication certificate. After checking the correctness of the key, it reformats the information or modifies certain values. This is different to simple information passing. It is possible to encode this in the π-calculus as $x(a).M\{a/b\}.\overline{y}(b)$ or $x(a).\tau.\overline{y}(a)$. However during the proof it became clear, that these encodings can introduce substantial complexity. Recalling that the π-calculus describes the communication behaviour of processes a reformatting of information does not change the information content, thus it has to be decided if it has to be included in the system specification.

Impressiveness of Process Expressions

Above we were recalling which properties can and should be encoded in the process expressions. Now it will be discussed, which analysis can be done using the process expressions. The obvious answerable questions include, if a certain information is accessible by a certain process (ie. can the server access the policy residing inside the ASCap proxy only communicating on the secret channel l - Answer: Using scope extrusion it can be derived). But during the behavioural equivalence proof it became clear that further statements are derivable. The ASCap framework incorporates the channel c_{SA} between the ASCap proxy and server. If this channel is not cryptographically secured (meaning restricted), it is possible that the client commits into communication with an arbitrary server, or even a Man in the Middle Attack becomes possible. If the framework is only implemented the need to throughoutness is less immediate, while by using formal methods (ie. π-calculus) to rewrite the equation the strictness of rules points out weaknesses. In our case for the equivalence proof we wanted to erase two nondeterministic choices, but it became clear that using an unrestricted channel c_{SA} these choices exist. In the implementation this would allow not only the ASCap proxy to connect to the server, but to other malicious servers, too.
For our framework this means that we could prove *weak late congruence* and not open bisimilarity. Formally this was caused by the use of mismatch operator and is expressed that congruent expresses may not be equivalent after renaming of channels or in different contexts. An example would be, that the client process expression renames r to w and vice versa, thus changing the meaning of a channel name.

Specification and Implementation

Like stated in the paper of Esterline and Rorie [5] the π-calculus expressions act as system specifications and it is possible to use them as verification tools for an actual implementation. The process expression would then need to be refined gradually, showing for each step a proof, that behavioural equivalence is given.

Generalisation

It is desirable to proof behavioural equivalence for all policy expressions, but the proof of the external security server based setup has shown, that this is not possible. Recalling that this setup allows one or more external security servers to be incorporated, or in different terms the policy to be split into one or more parts, it is clear, that different ASCap proxies and server implementations must exist. This is reflected also in the process expressions. We have chosen to present a proof restricting the form the policy expression can take and it is conceivable to generalise this approach to derive policy expression classes. Future work in π-calculus may categories expressions, practically categorising system behaviour.
A different approach to generalisation is the use of automated equivalence testing tools, such as the mobility workbench [7]. Automated tools are convenient to use, but may have limitations such as that not all primitives can be used or only one class of equivalence is tested. This resulted in our case to reconsider to manually proof the equivalence.

7 Conclusions

In the previous sections we have shown that behavioural equivalence between different policy implementations, as well as access control mechanism can be done using the π-calculus.

This result can be beneficial in two ways, in the first it shows that a unified access control mechanism can safely simulate all behaviour an equivalent hybrid system can do. The access control community may *ravish* from this result by safely implementing new models, policy languages or management tools using the unified framework. The practitioner eventually will be provided with a unified access control mechanism implementing all possible access control models. In marketing terms: the time to market from the research developments will be drastically reduced. Secondly this result shall be interesting to the theoretical computer science community to allow insights of the thoughts and needs practitioners may have. Especially the section of access control policy and mechanism process expression derivation should show that existing π-calculus derivates are sufficient. However the mismatch encoding might be a bit clumsy in larger frameworks and research toward a programming language like syntax might help non-theoretical employment of the existing calculus. Further on the theoretical calculus community evolves around terms of congruence, equivalence or bisimulation. Employing a calculus in the security

setting clearness of meaning and relevance of the different terms may change. To clarify this further examples of the differences of the different terms including their translation into the practical world may be useful. It may be well possible that techniques to emphasise the difference of process expressions, such as showing contexts in which the similarity fails will lead to development of automated security checking tools. Some examples of this can be seen in Abadi's work on formalising authentication protocols [21]. Or the automated protocol checker of Crazzolara and Milicia [12].
Finally it can be said that although formal methods require a certain insider knowledge they are already today ready to employ to practical world problems.

Acknowledgement

The author would like to express his appreciation to Mogens Nielsen for his guidance and invaluable advice during this research project and finally to invite me to be guest at BRICS (Basic Research in Computer Science).
Also I would like to thank Hans Huettel, Pawel Sobocinski and the whole of the BRICS PhD students for their input, encouragement and support.

References

[1] W. S. A. Regev and E. Shapiro. Representation and simulation of biochemical processes using the pi-calculus process algebra. In *Proceedings of the Pacific Symposium of Biocomputing 2001 (PSB2001)*, pages 6:459–470, 2001.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[3] J. Abendroth and C. D. Jensen. Partial outsourcing: A new paradigm for access control. In *Proceedings of the 8th ACM Symposium on Access Control Models and Technologies(SACMAT 03), June, Como Italy*, pages 134–141, 2003.

[4] J. Abendroth and C. D. Jensen. A unified security mechanism for networked applications. In *Proceedings of 18th Symposium on Applied Computing (SAC2003)*, pages 351–357. ACM, March 2003.

[5] T. R. Albert C. Esterline. Using the π-calculus to model multiagent systems. In *Lecture Notes in Computer Science Volume 1871*, pages 164–179, 2001.

[6] J. G. S. B. Clifford Neuman and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Winter 1988 USENIX Conference*, pages 191–201, Dallas, TX, 1988.

[7] B. Victor and F. Moller. The mobility workbench: A tool for the pi-calculus. In David L. Dill, editor, *Proceedings of the sixth International Conference on*

Computer-Aided Verification CAV, volume 818, pages 428–440, Standford, California, USA, 1994. Springer-Verlag.

[8] D. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Report MTR 2547 v2, MITRE, November 1973.

[9] M. Carbone and M. Nielsen. A formal model for trust in dynamic networks. In *Proceedings of the Software Engineering and Formal Methods, SEFM'03, IEEE Computer Society Press*, September 2003.

[10] G. Coulouris and J. Dollimore. Security requirements for cooperative work: a model and its system implications. In *ACM European SIGOPS Workshop*. ACM, 1994.

[11] B. C.Pierce. Programming in the pi–calculus - an experiment in programming language design. Technical Report Tutorial notes on the Pict language., University of Edinburgh, 1995.

[12] F. Crazzolara and G. Milicia. Developing security protocols in χ-Spaces. In *Proceedings of the 7th Nordic Workshop on Secure IT Systems (NordSec)*, November 2002.

[13] e. a. Eve Cohen. Models for coalition-based access control (cbac). In *7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 97–106, 2002.

[14] Ferraiolo and Kuhn. Role based access control. In *Proceedings of 15th National Computer Security Conference*, 1992.

[15] D. Hagimont, J. Mossiere, X. R. de Pina, and F. Saunier. Hidden software capabilities. In *International Conference on Distributed Computing Systems*, pages 282–289, 1996.

[16] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(5):566–591, 2002.

[17] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[18] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science*. Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, 2002.

[19] R. S. S. Jaehong Park. Towards usage control models: beyond traditional access control. In *7th ACM Symposium on Access Control Models and Technologies (SACMAT 2002)*, pages 57–64, 2002.

[20] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD International Conference on Management of Data*, pages 474–485. ACM Press, 1997.

[21] Lampson, Abadi, Burrows, and Wobber. Authentication in distributed systems: Theory and practice, from ACM transactions on computer systems, november, 1992. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press, 1996*. ACM, 1996.

[22] R. Milner. *Communicating and Mobile Systems: The Pi-Calculus*. Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, 1999.

[23] B. C. Neumann. Proxy-based authorisation and accounting for distributed systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 283–291, Pittsburgh, Penn, U.S.A., May 1993.

[24] M. S. Olivier. Towards a configurable security architecture. *Data and Knowledge Engineering*, 38(2):121–145, 2001.

[25] A. Ott and S. Fischer-Hübner. Rule set based access control as proposed in the 'generalized framework for access control' in linux. In *Karlstadt Univeristy Studies, 2001:28, ISBN 91-89422-63-5*, 2001.

[26] J. A. Padget and R. J. Bradford. A pi-calculus model of a spanish fish market - preliminary report. In *AMET*, pages 166–188, 1998.

[27] C. R. Ramakrishnan, I. V. Ramakrishnan, S. A. Smolka, Y. Dong, X. Du, A. Roychoudhury, and V. N. Venkatakrishnan. XMC: A logic-programming-based verification toolset. In *Computer Aided Verification*, pages 576–580, 2000.

[28] C. R. Ramakrishnan and R. Sekar. Model-based analysis of configuration vulnerabilities. *Journal of Computer Security (JCS)*, 10(1 / 2):189–209, 2002.

[29] R.E.Johnson. Frameworks= components + patterns. *CACM*, 40Nr.10:39–42, 1997.

[30] M. Röscheisen and T. Winograd. A communication agreement framework for access/action control. In *Proceedings of the IEEE Symposium in Security and Privacy*, 1996.

[31] D. Sangiorgi and D. Walker. *The Pi-Calculus, A theory of Mobile Processes*. Cambridge University Press, The Edinburgh Building, Cambridge, DB2 2RU, UK, 2001.

[32] R. K. Thomas and R. S. Sandhu. Towards a task-based paradigm for flexible and adaptable access control in distributed applications. In *ACM SIGSAC New Security Paradigms Workshop*, pages 138–142, 1993.

[33] W. L. Tin Qian. Active capability: An application specific security and protection model. Technical report, University of Illinois at Urbana-Champaign, 1996.

[34] Various. Open source pki book, http://opensourcepkibook.sourceforge.net, 1.12.2002.

[35] B. Victor. *A Verification Tool for the Polyadic π-Calculus*. Licentiate thesis, Department of Computer Systems, Uppsala University, Sweden, May 1994. Available as report DoCS 94/50.