
# CESSA

## Compositional Evolution of Secure Services using Aspects

## ANR project no. 09-SEGI-002-01

# Language definition for security specifications

**Abstract.** We introduce CSPL, the CESSA Security Policy Language. CSPL has been designed to enable specifying security policies in distributed systems in a concise, readable and extensible way. CSPL is centered on *policy domains*, *i.e.* sets of locations in which coherent policies apply, and is designed to be enforced by reference monitors controlling the flow of information between services. CSPL allows the definition of both *abstract* and *concrete* policies, expressing respectively high-level properties required and concrete implementation details. These latter are designed to be ultimately woven in the service implementation.

| | |
|---|---|
| Deliverable No. | D2.2 |
| Task No. | 2 |
| Type | Deliverable |
| Dissemination | Public |
| Status | Final |
| Version | 1.0 |
| Date | 27 May 2011 |
| Authors | Matteo Dell'Amico, Sabir Idrees, Yves Roudier (EURECOM); Anderson Santana de Oliveira, Gabriel Serme (SAP); Gaëtan Harel (IS2T) |

# Contents

# Chapter 1

# Introduction

We introduce CSPL, the security policy language for CESSA. This language has the goal of being a convenient way to specify security policies, providing the administrators the following characteristics:

- Expressing the security policy in a single place: the code implementing the services shouldn't be dependent on which security-related measures are in place, such as for example filtering, sanitization, or encryption rules.
- Allowing to express policies both with a level of detail that can vary from very high-level requirements (*e.g.*, ensuring confidentiality or integrity of a particular piece of data) to *concrete* policies, defining in detail what will be the security mechanisms that will be applied. Concrete policies represent aspects that can be woven into the system, and they can be implemented, for example through reference monitors intercepting messaging between services.
- A design goal for the language is to keep specifications clear and understandable, minimizing the need for code duplication and helping maintainability, even when the security policy is determined by several different entities.

In addition, CSPL is agnostic about the characteristics of the distributed system it is implemented on. While at the moment our attention is centered on the Service Oriented Architectures that are at the core of the CESSA project research directions, the language allows for extensions towards other types of distributed systems, making it potentially usable in systems that are hybrid or with different architectures.

Several languages for security policies have been proposed in the literature. However, there are various features that distinguish our proposal from what has been done in the past. First, in several cases the security policies expressed are limited to authorization and access control [13, 7, 12, 5]. Unlike these approaches, CSPL aims to express more than just rules allowing or forbidding access to users, also enabling the possibility of introducing additional security features by *modifying* the communication protocols, for example by introducing cryptography, additional security metadata, or by rerouting messages. Second, our approach is meant to express

security policies at different levels of abstraction, from the abstract properties that the administrator wants to ensure to the concrete implementation details needed to specify the mechanisms that are adopted. We plan on providing support to help administrators reason about the relationship between the implementation details of concrete policies and the abstract properties required.

CSPL is based on the central concept of *policy domains*. A policy domain is a set of locations in which a set of particular security policies apply. Given that security policies can be dependent on several different characteristics, domains can represent very different entities, such as corporations, individuals, down to the level of real or virtual machines, or even applications running on those machines.

Policies are enforced and monitored *at the borders* of policy domains. In this way it is possible to make sure that, for example, particular sets of informations remain confined within a domain or get annotated with additional security metadata (*e.g.*, a Message Authentication Code or a cryptographic signature) when they leave a particular domain.

Policy domains are structured hierarchically, in order to allow the coexistence of policies that cover a wide set of locations (such as an organization) and of very specific ones (for example, access to a particular service).

We allow the definition of *domain attributes* that can be used to specify characteristics of a particular domain that are relevant to security. For example, domain attributes can specify the type of entity corresponding to a domain (organization, individual, server, ...), roles of an individual, up to technical details such as the kind of operating system running on a server. All this information is potentially relevant with respect to the security policies, and domain attributes can be used to specify policies that apply to domains with common characteristics, even when they are in different parts of the domain hierarchy.

We augment the information in the system with *information tags* that representing security metadata. Reference monitors manage information tags and perform several tasks adopting them. For example, based on the tags it has, information can be filtered, transformed (*e.g.*, through encryption, stripping of confidential information, sanitization against injection attacks) and/or rerouted.

The rest of this document is structured as follows: Chapter 2 describes CSPL and its syntax; Chapter 3 shows how the security requirements are expressed in CSPL in our use cases of loan negotiation and home automation; Chapter 4 reviews the related work; finally, Chapter 5 concludes.

# Chapter 2

# The CSPL Security Policy Language

In CSPL, we express security policies in a distributed system by differentiating within *policy domains*. We consider that each policy domain will implement a *reference monitor* that enforces security mechanisms in its domain in line with what has been introduced in the CESSA deliverable D1.3. Indeed, the same security mechanisms can be implemented in other ways, for example by distributing reference monitors or by modifying the code of services in execution. As long as the final behavior of the system will be the same, we consider this to be an implementation detail, and therefore outside the scope of this document.

Many of the mechanisms required will be implemented by intercepting, and taking action, on communications crossing trust domain boundaries. We can illustrate the work of reference monitor with the metaphor of customs control: security is enforced by controls about what gets in and out of a domain. Examples of the kind of actions that a reference monitor can apply are:

- filtering: reference monitors can implement access control to resources outside of their original policy domain[1] by filtering unauthorized messages;
- cryptography: information can be encrypted, decripted or signed when leaving or entering policy domains;
- managing security metadata: in our system, information is augmented with metadata that we label as *information tags*; in Section 2.2, we will focus on them.

In other cases, reference monitors can enforce security policies by triggering actions that will take place in their policy domain: for example, a data retention policy can specify an obligation to erase data after a given amount of time; the reference monitor in this case will need to behave actively, triggering an action to delete the data.

A CSPL policy is divided in two parts: a policy domain hierarchy, and a set of rules. In this Chapter, we provide more details on policy domain hierarchies in Section 2.1; we will then describe information tags in Section 2.2 and describe the form of rules in Section 2.3.

[1] It is important to remember that policy domains can be made as small as required; for example, to enforce access control to a service from any other location, a policy domain can exist enclosing only the original service.

## 2.1 Policy Domains

A policy domain is a location or set of locations in which security policies are applied. It corresponds to a logical set of location that has particular properties that can be relevant to security (for example, a security domain can correspond to an organization, an individual, a place, a real machine, a virtual machine, and even a single application).

Policy domains are organized hierarchically: any domain can contain any number of subdomains, up to an arbitrary level of nesting. For a domain included within a parent domain, policies both in the enclosing and the inner domains will apply. This allows us to naturally define rules both at large (*e.g.*, organizations) and small (*e.g.*, services) scales. We allow reference monitors to communicate exclusively with services in their own policy domain and with the reference monitors of their neighbors in the domain hierarchy (*i.e.*, parent domains and subdomains): this constrains the flow of information and helps us enforce our required policies.

It is important to point out that rules are *not monotonic*. For example, consider a subdomain $B$ of domain $A$. If we consider requirements on data confidentiality, a piece of information can be allowed to be readable only within $B$ and not in the rest of $A$, but also the opposite can apply: if $B$ for some reason is considered "less trusted" than the rest of $A$ (consider for example a mobile device that can fall more easily in the hands of an attacker), then restrictive rules can be applied whenever data is sent to $B$.

We allow data fields to be attached to policy domains, for example to contain encryption keys used by a principal.

The hierarchy of policy domains also allows drafting a security policy cooperatively: the language supports the definition of rules that only apply within one, or more, policy domains. In our framework, administrators from involved entities should agree on "top-level" security policies applying to all domains, but they can be free to define additional security policies applying only to a domain of their competence.

### 2.1.1 Domain Attributes

We also allow specifying an arbitrary number of additional *attributes* for each policy domains, when defining their hierarchy. Domain attributes are free-form text labels, allowing to attach additional information to the containment relationships implied by the hierarchical domain policy structure.

Domain attributes allow to specify policies that apply to several policy domains without the need to list them explicitly, allowing them to "cross" the hierarchical policy domain structure. For example, all policy domains corresponding to mobile devices can be labeled as "mobile"; afterwards, it will be easy to define policies that apply to all mobile devices in the policy domain hierarchy, or some sub-hierarchy of it by writing policies that apply only to domains with a "mobile" attribute.

Domain attributes can be used for several purposes, such as describing what a policy domain corresponds to (*e.g.*, an organization, an individual or a device) or some technical architecture details (*e.g.*, the kind of operating system running on a machine).

A particularly interesting case is when domain attributes identify roles: when a policy domain represents an individual, one attribute can represent the role of that person (*e.g.*, programmer, clerk or manager); rules can then differentiate policies based on individuals' roles. Furthermore, since there is no restriction on the number of attributes that can be attached to a domain, a single individual can have more than one single role if this is needed.

Using policy domain attributes helps maintainability: when a new policy domain is created, it will be sufficient to label it with the appropriate attributes and the relevant security policies will be applied to it as well.

## 2.2 Information Tags

Along with policy domains, information tags are another key part of CSPL. Information tags represent security metadata which is attached to information and processed by reference monitors. Information tags allow to specify meta-data about the content of information. This is used to define rules that apply only to particular categories of data.

In addition, information tags can be used to group different pieces of information. It is possible to define CSPL rules that apply to information that has particular tags; when combined with domain attributes this allows us to naturally define policies that apply to large sets of information and span different policy domains.

A very important application of tags are information flow-based rules. We can identify information based on their provenance, and propagate those tags as information gets propagated. Depending on the particular policy, it will be possible to propagate tags on derived information and define rules for applying tags to derived information or defining services that are able to strip tags away. Among the possible applications of information-flow rules, there are taint checking (*e.g.*, tagging data coming from untrusted sources in order to avoid attacks such as code injection) and privacy enforcement (by not allowing information tagged as confidential to escape from designated policy domains).

## 2.3 Rules

After introducing the key concepts of policy domains and information tags, we are now ready to describe how rules are expressed in CSPL. We will show an example rule and informally describe its semantics.

Our example rule is as follows, and expresses that any message within the BBB Bank which is sent by an employee has to be signed.

```
BBB_Bank {
   m:message, m.from==y::employee, m.contents==x
   -> y.key == k, x is signed(k)
}
```

This rule is limited to the `BBB_Bank` policy domain (and its subdomains); in the place of the `BBB_Bank` identifier there could be a domain attribute (such as, for example, `bank`) to specify that the rule applies to all banks. Policy domains can be easily distinguished from domain attributes because they begin with uppercase letters (reflecting the grammatical distinction between proper and common nouns in English).

Rules have two sides, separated by the $\rightarrow$ construct. What appears on the left side is a *pattern*; reference monitors will have to ensure that what is specified on the right side applies whenever the left-side pattern is matched.[2]

Both on the left side and on the right side, a list of comma-separated clauses is specified; the semantics in both cases is an implicit conjunction: all clauses on the left side should be verified for the pattern to match and all clauses on the right should be enforced by the reference monitor.

All free variables are implicitly quantified universally on the left side (`m`, `y`, and `x` in the example), and quantified existentially when they appear only on the right side (`k` here). This means that when `m`, `y`, and `x` satisfy the conditions appearing on the left, the appropriate reference monitor should enforce that for a value of `k` the clauses on the right hand apply. Note that variables that appear free on both sides (such as `y` in the example) are quantified universally. This semantic choice maps well to our "pattern $\rightarrow$ enforcement" scheme: for *any* variable assignment that matches the pattern on the left side, there should *exist* a set of variables that satisfies the clauses on the right.

Several other language constructs appear in this rule: the `m:message` clause is verified when the piece of data `m` has the `message` information tag; likewise, the `y::employee` clause is verified when the policy domain `y` has the `employee` attribute. In addition, we can access data *fields* of information and on policy domains: in this case, the rule is accessing the fields `from` and `contents` from `m`, and the data field `key` attached to policy domains. The last construct appearing in the example rule is the spefication of a security property that has to apply, recognizable by the usage of the keyword `is`. In this case, `x is signed(k)` is describing that the `signed` property has to apply to `x` with a parameter `k` identifying, in this case, the key that should be used for signature. In this case, `signed` is an *abstract* (*i.e.*, non-concrete) policy, since it does not provide all details neeeded to put the required measure in place (for example, choice of the needed signature algorithm).

In synthesis, a plain English reading of the example rule is the following: *"The following rule applies only to the policy domain of bank BBB. For each message m sent by an employee y, y must have a key, and use it to sign the contents of the message."*

It is possible that rules will require actions that are impossible to satisfy or in conflict with other rules. We plan to investigate how to detect conflicting rules, both statically (*i.e.*, when drafting the security policy) and at runtime, and on determining ways to manage them. In this current proposal, we will limit ourselves to a simple priority rule, giving priority to more specific rules (*i.e.*, those defined for smaller policy domains) and, to discriminate between rules defined at the same hierarchic level, we will give priority to the one defined first.

[2]Despite the syntactical similarity with Prolog or Datalog rules, CSPL rules are *not* inference rules: the right side is used to specify conditions that have to be enforced by reference monitors rather than conclusions that can be proven.

## 2.4 Syntax

In this section we present the abstract syntax of the policy language we propose.

$$
\begin{array}{rcl}
SPEC & ::= & \text{“}domains\text{”}\ DOMHIERARCHY\ \text{“}rules\text{”}\ RULES \\
DOMHIERARCHY & ::= & \text{“}\{\text{”}\ DOMAIN*\ \text{“}\}\text{”} \\
DOMAIN & ::= & DOMAINDECL \\
 & | & DOMAINDECL\ DOMHIERARCHY \\
DOMAINDECL & ::= & DOMAINNAME \\
 & | & DOMAINNAME\ \text{“}(\text{”}\ DOMAINATTR*\ \text{“})\text{”} \\
DOMAINNAME & ::= & [A..Z][A..Za..z_]* \\
DOMAINATTR & ::= & [a..z_]+ \\
RULES & ::= & RULE+ \\
RULE & ::= & FORMULA\ \text{“}\rightarrow\text{”}\ FORMULA \\
 & | & DOMAINNAME\ \{\ RULES\ \} \\
 & | & DOMAINATTR\ \{\ RULES\ \} \\
FORMULA & ::= & TERM+ \\
 & | & TERM\ OP\ TERM \\
 & | & UNOP\ TERM \\
TERM & ::= & VAR \mid VARPROP \mid TAGGEDVAR \mid DOMAINTERM \\
OP & ::= & \in \mid \notin \mid \neq \mid \leq \mid \ldots \\
UNOP & ::= & \neg \mid \ldots \\
VARPROP & ::= & VAR\ \text{“}is\text{”}\ PROPERTY(ARGS) \\
TAGGEDVAR & ::= & VAR\ \text{“}:\text{”}\ TAG+ \\
DOMAINTERM & ::= & DOMAINNAME \\
 & | & VAR\ \text{“}::\text{”}\ DOMAINATTR \\
ARGS & ::= & TAGGEDVAR* \\
 & | & VARPROP* \\
 & | & DOMAINNAME* \\
 & | & VAR* \\
PROPERTY & ::= & \text{“}authenticated\text{”} \\
 & | & \text{“}confidential\text{”} \\
 & | & \text{“}signed\text{”} \\
 & | & \text{“}encrypted\text{”} \\
 & | & \ldots \\
TAG & ::= & [a..z_]+ \\
VAR & ::= & VAR\ \text{“}.\text{”}\ FIELD \\
 & | & VAR\ \text{“}==\text{”}\ TERM \\
 & | & [a..z]+ \mid \text{“}\_\text{”}
\end{array}
$$

A policy specification contains a hierarchical domain description followed by a non-empty set of rules. For instance, a part of the domain policy hierarchy related to a user John would be given as follows:

```
John (individual, top-level) {
    JohnsDevice (device, embedded) {
        MobileBanking (app)
    }
    JohnsSSCD (device, embedded)
}
```

Where "individual", "top-level", "device", "embedded" and "app" are DOMAINATTRs, while "John", "JohnsDevice", etc. are DOMAINNAMEs. They start with a capital letter, reflecting the intuition they are proper nouns.

In large systems with many policy domains (for example, those with many users or many devices) the hierarchy of policy domains can become rather large and be difficult to handle. We consider that this problem can be solved by using a macro language such as GNU m4[3] to automate the generation of the policy domain hierarchy.

A rule is formed by a formula on the left and right-hand sides, separated by $\rightarrow$. A formula is composed of one or more terms separated by commas. We omit the commas in the abstract syntax for simplicity. As we have seen with our example rule in the previous Section, rules can be given a scope, either by attaching them to a domain element or a domain attribute.

As the symbol name suggests, tagged variables are variables annotated with at least one tag, whereas DOMAINTERM stands for the access to some attribute (or value) linked to a given variable inside a domain.

For instance, $m.from == x$ indicates that one wants to capture the sender of the message $m$ and reference to it with the variable $x$ in the policy rule above.

We also included some fundamental properties or tags in the language with special semantics. In this class, we can mention the *message* tag for exchanges between entities, or the *signed*, $confidential$, and *encrypted* properties. The arguments in variable properties can assume several forms, as indicated in the grammar. The exact meaning will depend on the kind of property being activated.

Remark that attributes can have structure, as in object-oriented programming. In the example $x.bank.department$, this can be read as the department attribute of the bank object associated to the variable $x$.

The language also makes use of logical and comparison operators, represented by the grammar symbol OP for binary operators or UNOP for unary operators. The exact set of operators used is not relevant for now, and we will introduce them in the language when necessary.

The language semantics must define means for its users to define properties and tags, extending the language to cover specific situations.

[3] http://www.gnu.org/software/m4/m4.html

# Chapter 3

# Application and Use case Mapping

## 3.1 Use Case 1: Loan Negotiation

Banking services offer a paradigmatic scenario in the e-business application area as they highlight how the coexistence of flexibility on one hand and security and trust on the other often becomes contradictory. If banking applications are expected to be flexible to fulfill the needs of all involved parties, such as bankers, partners, customers and investors, they are also required to strictly meet a variety of security and trust requirements. Such a variety is in fact very large. For example, particular requirements concerning separation of duties, secure logging of events aimed at auditing, non-repudiable actions, digital signatures, etc, need to be considered and satisfied in order to comply with the current regulations.

One of the core processes of a banking scenario is the Loan Origination Business Process (LOBP), which formalizes a bank's evaluation of a customer's request for a loan, described in [8]. To achieve its business goals, the business analysts have designed all the sequences of tasks executed by business resources in order to perform the loan process. We assume the bank is using role-based access control. The LOBP aims to evaluate and possibly grant a customer request for a loan amount. This can be operated by the flow of tasks outlined hereafter. After receiving and identifying the customer, the bank carries out a careful evaluation of the customer's rating through internal mechanisms and by asking assurance to external agencies called credit bureaus. Subsequently, a bundled product is selected and, if both the customer and the bank agree, a contract is signed.

In Figure 3.1, three BPMN lanes capture the three main bank roles that are supposed to be active in the execution of the LOBP: the pre-processors who receive and identify the customer's request and update/create the customer data in the banking system (Input Customer Data and Customer Identification tasks); the post-processor who analyze the credit worthiness of the customer internally and externally via interaction with a credit bureau (Check Internal Rating and Check External Credit Worthiness tasks), identify the most appropriate bundle product (Select Bundle Product task), submit a loan proposal to the customer, and, provided that the contract is signed, they are the ones opening the customer bank account in the bank system (Open Account in System task); the managers who are responsible for the local agency of the bank and intervene

to provide their approval to the loan (Sign Form task).

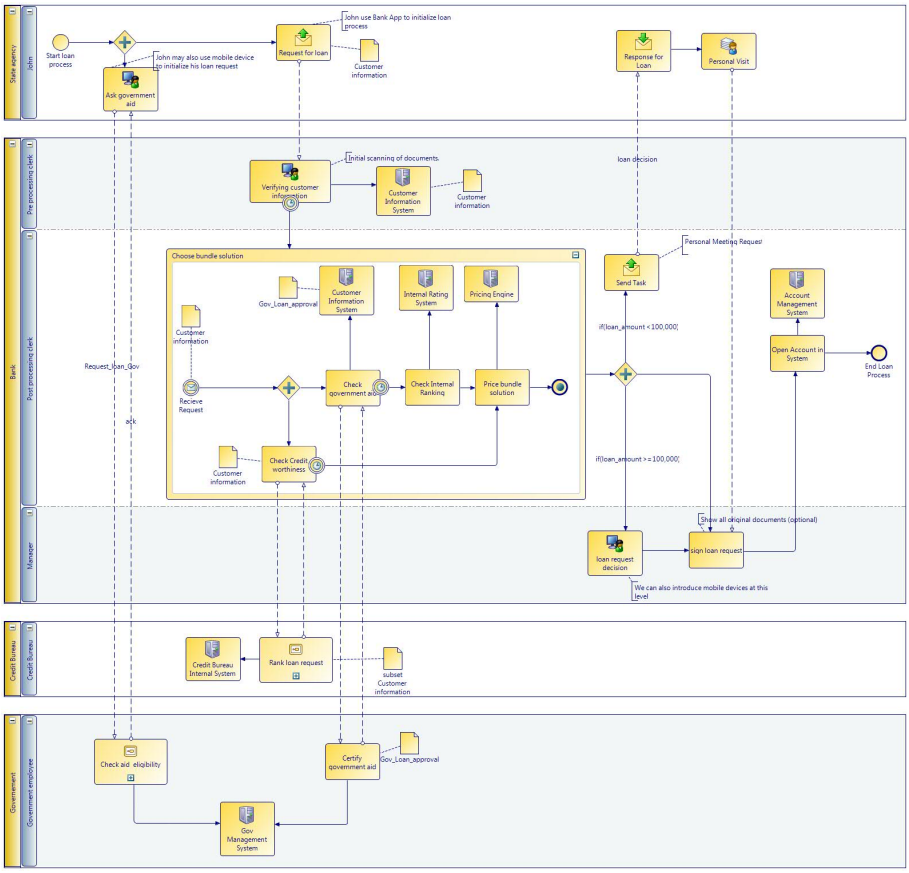



Figure 3.1: Loan Origination Process Flow

### 3.1.1 Policy Domain Hierarchy

In this scenario we can identify the following domain hierarchies:

1. John (individual, top-level)
   (a) John's mobile device (device, embedded)
      i. Mobile Banking application (app)
   (b) John's SSCD secure signature device (device, embedded)

```
John (individual, top-level) {
    JohnsDevice (device, embedded) {
        MobileBanking (app)
    }
```

```
    JohnsSSCD (device, embedded)
}
```

2. Real Estate Agency (organization, top-level)
    (a) Embedded Device (device, embedded)
        i. Application (app)

```
Real Estate Agency (organization, top-level) {
    Embedded Device (device, embedded) {
        Application (app)
    }
}
```

3. BBB Bank (organization, bank, top-level)
    (a) Department X (organization)
        i. Peter (individual, employee, clerk)
        ii. Gabriel (individual, employee, clerk)
        iii. Ted – Manager (individual, employee)
            A. Ted's SSCD secure signature device (device, embedded)
    (b) Data Center (location)
        i. Internal Rating System (service)
        ii. Pricing Engine (service)

```
BBB Bank (organization, bank, top-level) {
    Department X (organization) {
       Peter (individual, employee, clerk)
       Gabriel (individual, employee, clerk)
       Ted (Manager, individual, employee) {
        TedSSCD (device, embedded)
   }
    }
Data center(location){
     Internal rating system(service)
 Pricing engine (service)
}
}
```

4. Credit Bureau (organization, top-level)

(a) Courts & Deeds Offices (organization)

(b) Credit Accounts Dept. (organization)

```
Credit Bureau (organization, top-level) {
       Courts and deeds office (organization)
       Credits account dept (organization)
}
```

5. Government (organization, top-level)

   (a) Employee (organization)

   (b) Government management system (system)

```
Government (organization, top-level) {
       Employee (individual, organization)
       Management system (system)
}
```

### 3.1.2 Security Policies

#### 3.1.2.1 Generic Policies

1. No data is readable by anybody apart from the entities enumerated in our policy domains.

   - x $\rightarrow$ x is confidential(top-level)

   In this policy, "top-level" is a tag corresponding to all the policy domains tagged as "top-level" in the domain enumeration above. It is just a shorthand notation for the enumeration of all top-level policy domains, i.e. "x $\rightarrow$ x is confidential(John, Estate Agency, Bank, Credit Bureau, Government)"

2. Each message sent between different top-level entities is signed by the sending entity to ensure authenticity

   - m:message, m.from == x::top-level, m.to==y::top-level, x $\neq$ y $\rightarrow$ m is signed(x)

#### 3.1.2.2 Client Personal Information

This section lists the security and trust requirements for the banking services scenario. Therefore, the sub-processes forming the loan origination process that were discussed above will have to be scrutinized exactly in terms of security and trust. It must be remarked how privacy often steps in as an important aspect of security and trust.

The Personally Identifiable Information (PII) is particularly important from the trust perspective. PII is "any piece of information which can potentially be used to uniquely identify, contact, or locate a single person" and therefore not everyone should be trusted to handle it. For example, in the banking scenario the primary PII of a customer typically is his passport or identification number. It is important to distinguish the PII from other private information such as gender or political preferences. It is understood that the current social context imposes any banking decision to only be taken upon the PII. This abstract requirement is expressed below by a few requirements concerning privacy.

1. Usage of clients' personal information (at bank, credit bureau, government) is authorized by him

   - x:client-personal-info $\in$ m:message, m.to == y $\rightarrow$ a:authorization, a.subject == y, a.object == x, a is signed(x.client)

2. Any form of customer's authorization shall be provided by the customer's signature

   - x:client-personal-info, a:authorization, a.subject == x $\rightarrow$ a is signed($x$.client)

3. Integrity of personal information is ensured by using signed documents.

   - m:message, x:client-personal-info $\in$ m.contents $\rightarrow$ x is signed(x.origin)

4. Personal client information can only be read by trusted entities:

   - m:message, x:client-personal-info $\in$ m.contents $\rightarrow$ m is confidential(bank, credit bureau, government)

#### 3.1.2.3 Applications in embedded devices

1. Application integrity is verified

   - m:message, m.from == x::app, x.parent == y, y::embedded, y::device $\rightarrow$m is integrity_verified(x)

#### 3.1.2.4 Bank

1. Data in John's customer information file is confidential with employees in Department X

   - x:customer_info $\rightarrow$ x is confidential(x.customer.bank_department, x.bank.internal_rating_system)

2. Clients' process file is confidential within the bank the BBB bank, and its parts are signed by bank employees

   - x:process_file $\rightarrow$ x is confidential(bank)
   - e:employee, e.in == b:bank, x:process_file $\in$ m.contents $\rightarrow$ x is signed(e)

3. Separation of duties between pre-processing and post-processing clerks

   Separation of duties can be seen as a design principle for the protection of information in computer systems and a mechanism to control fraud and ensure the consistency of the data objects. Separation of duties concerns with dividing the responsibility about sensitive information so that no individual acting alone can compromise the security of a business process. A simple way to enforce a separation control is to prevent a single principal to own all the necessary authorizations for each required step of a process. A more relaxed alternative would be to prohibit him to perform all the steps on his own. This is sometimes referred to as a dual control or four-eyes principle, since two or more people are needed for the execution of a critical process. The problems that no-fraud requirements cause are solved by separation of duties by a careful user-to-role assignment strategy of one of the following five kinds:

   (a) A principal must not be a member of any two exclusive roles (static separation of duties). We must define tags to indicate that roles are exclusive.

   (b) If a principal is a member of any two exclusive roles, then he must not activate them at the same time (dynamic separation of duties). In the case we support this case, in addition we must include tags for role activation.

   (c) If a principal is a member of any two exclusive roles and activates them at the same time, then he must not act upon the same object through both (object-based separation of duties).

   (d) If a principal is a member of exclusive roles, then the set of authorizations acquired over these roles must not cover an entire workflow (operational separation of duties).

   (e) If a principal is a member of exclusive roles, and the set of authorizations acquired over these roles cover an entire workflow, then the principal must not use all authorizations on the same object (history-based separation of duties). In this case we need tags to record the history of role activation and object access.

   - x:accept_loan , p1:pre-processing $\in$ x, p2:post-processing $\in$ x $\rightarrow$ x is signed(p1, p2), p1 $\neq$ p2

4. A fair non-repudiation of the signature of the contract shall be ensured to both parties.

   - x:loan_contract(BBB,y)→ z:post-processing-clerk, x is signed(z,y)

5. If the loan exceeds 1 million euros, the manager shall sign the form instead of the post-processing clerk.

   - x:loan_contract(BBB,y), x.amount $>$ 1 million euros → z:manager, x is signed(z,y)

6. The customer's PII shall not be sent to the pricing engine of the bank's internal computing system.

   - m:message, m.to == bank.internal_rating_system → x:customer_personal_info $\notin$ m

7. Appropriate cryptographic techniques shall be used to store the contract so that it cannot be modified without the customer's authorization.

   - x:loan_contract → x is securely_stored()

#### 3.1.2.5 Credit Bureau

- The Directive 95/46/EC is the reference text, at European level, on the protection of personal data. It sets up a regulatory framework which seeks to strike a balance between a high level of protection for the privacy of individuals and the free movement of personal data within the European Union (EU).

  The Directive aims to protect the rights and freedoms of persons with respect to the processing of personal data by laying down guidelines determining when this processing is lawful. The guidelines relate to:

  - The quality of the data: personal data must be processed fairly and lawfully, and collected for specified, explicit and legitimate purposes. They must also be accurate and, where necessary, kept up to date;
  - The legitimacy of data processing: personal data may be processed only if the data subject has unambiguously given his/her consent.

  In the context of the loan origination scenario, the rating requests received by the Credit Bureau need to be sanitized, unless the bank customer has explicitly consented to provide his personal data to third parties. The first case is already exemplified by the rule concerning the bank internal system. The second case is also covered by the rules on the client personal information protection from Section 3.1.2.2 related to the customer agreement on personal data distribution.

  Another concerned raised by the directive 95/46/EC is the notification of processing to a supervisory authority: the holder of personal data must notify the national supervisory

authority before carrying out any processing operation. In our scenario, we assume that the national government domain is responsible also carrying out such task, which is expressed by the following rule:

- x:client-personal-info $\in$ m:message, m.from ==x, g::government $\rightarrow$ m2:message, m2:notification, m2.from == x , m2.to == g

The directive establishes that prior checks to determine specific risks to the rights and freedoms of data subjects are to be carried out by the supervisory authority following receipt of the notification.

Additionally the 95/46/EC directive recommends that measures need to be taken to ensure that processing operations are publicized and the supervisory authorities must keep a register of the processing operations notified. This can be expressed in the following way in CSPL:

- m:message,m:notification, m.to==g, g::government $\rightarrow$ m is securely_stored()

- Code of conduct: the outcomes of the internal and external ratings shall in no circumstance be published.
  - x:rating $\rightarrow$ x is confidential(bank, credit bureau)

  The credit bureaus shall not seek or maintain any private information except the customer's PII. The 95/46/EC directive also establishes that it is forbidden to process personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade-union membership, and the processing of data concerning health or sex life. This provision comes with certain qualifications concerning, for example, cases where processing is necessary to protect the vital interests of the data subject or for the purposes of preventive medicine and medical diagnosis; However we consider this is out of the scope of CESSA since our objectives are not to control semantically the quality of the data being collected by one party in a given business process.

- Data Retention policies: The credit bureaus shall retain the customer's PII for no more than a fixed period of time, which is three years for non-fraudulent data.
  - x:client-personal-info $\rightarrow$ x is expiredby(x.creationdate + 3 years)

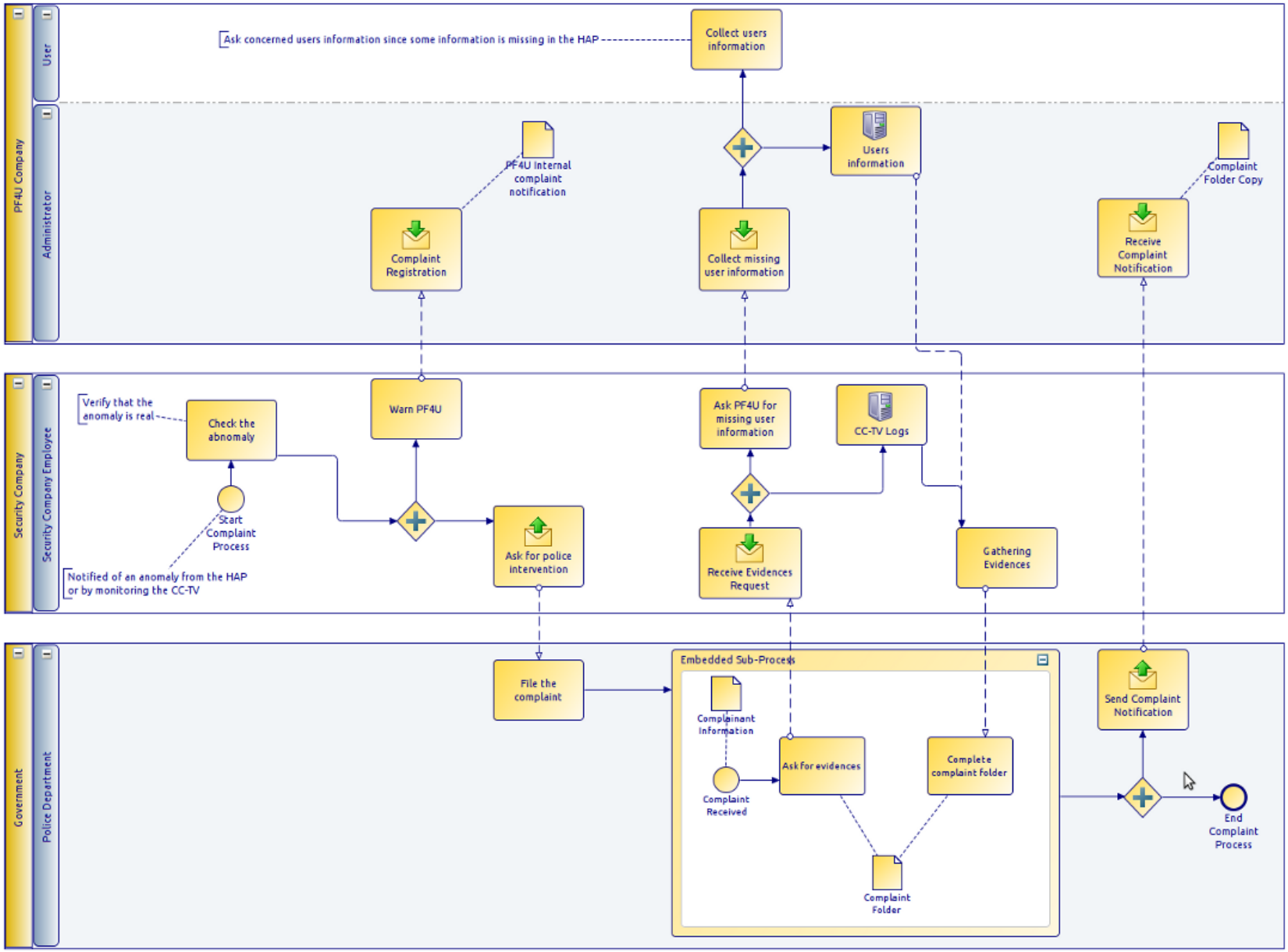
#### 3.1.2.6 Credit Accounts

- Credit accounts are confidential between their owner & Credit Accounts Dept.
  - m:message, x:credit-account-info $\in$ m.contents $\rightarrow$ m is confidential(x.client, credits-account-dept)

## 3.2 Use Case 2: Home Automation

The main scenario focuses on a burglary inside the PF4U company and the evidences that have been collected by the CC-TV system and motion sensors and how it would be provided to the government (the police department). The complaint and evidences can be automatically sent to the police department by the security company with a new secure test process the PF4U company is engaged in.
This section will introduce the security domains and policies regarding the above scenario.

Figure 3.2: Home Automation Burglary Work-flow



### 3.2.1 Policy Domain Hierarchy

The employee tag only refer to PF4U employees. The same notion of top-level flag is kept from the previous use case for global domain policies.

1. PF4U (organization, PF4U, top-level)
   (a) PF4U Employee (individual, employee)
   (b) PF4U Administrator (individual, employee, administrator)
   (c) PF4U HAP (system)
      i. PF4U Central Platform (platform, device)
      ii. PF4U Embedded Devices
      iii. PF4U CC-TV
      iv. PF4U Data-center (location)

```
PF4U (organization, PF4U, top-level) {
    PF4U_Employee (individual, employee)
    PF4U_Administrator (individual, employee, administrator)
    PF4U_HAP (system){
        PF4U_Central_Platform (platform, device)
        PF4U_Embedded_Devices (device)
        PF4U_CCTV (device, CCTV)
        PF4U_Datacenter (location)
    }
}
```

2. Security Company (organization, security, top-level)
   (a) Security Company Administrator (individual, security-employee)
   (b) Security Company Data-center (location)

```
Security_Company (organization, security, top-level) {
    Security_Company_ Administrator (individual, security-employee)
    Security_Company_Datacenter (location)
}
```

3. Police Department (organization, police, top-level)
   (a) Policeman (individual, police)
   (b) Police Department Data-center (location, police)

```
Police_Department (organization, police, top-level) {
    Policeman (individual, police)
    Police_Department_Datacenter (location, police)
}
```

### 3.2.2 Security Policies

#### 3.2.2.1 Generic Policies

The generic policies applies to every domain and serve as a fallback when other policies conflict which each other. The generic policies are quite the same as the ones defined in the loan negotiation use case.

1. No data is readable by anybody outside the entities enumerated in our policy domains.

   - x $\rightarrow$ x is confidential(top-level)

   "top-level" is a tag corresponding to all the policy domains tagged as "top-level" in the domain enumeration above. It is just shorthand notation for the enumeration of all top-level policy domains, i.e. "x $\rightarrow$ x is confidential(PF4U, Security_Company, Police_Department)"

2. Each message sent between different top-level entities is signed by the sending entity to ensure authenticity

   - m:message, m.from == x:top-level, m.to==y:top-level, x $\neq$ y $\rightarrow$ m is signed(x)

3. No-one must be able to modify the contents of a message unnoticed – to guarantee message integrity. The message has to be encrypted with the corresponding key in the possession of the sender.

   - m:message, m.from == y, m.contents == x, y.key == k $\rightarrow$ x is encrypted(k)

#### 3.2.2.2 PF4U Devices

1. PF4U Devices should be hardware verified, integrity of the devices are checked periodically (a software piece of code check the hardware integrity such as availability of different parts.

   - d:device, d.from == PF4U, s == d.software $\rightarrow$ d is integrity_verified(s)

#### 3.2.2.3 Devices Information

1. General Devices Information are managed by the HAP central platform, they are signed and are communicated only to the HAP central platform (signed rule is taken care by generic policies).

   - m:message, x:device_information $\in$ m.contents $\rightarrow$ m is confidential(platform, device)

2. CC-TV Information information are encrypted passed through HAP central platform and stored inside security company data-center, it is not available to PF4U employees. Moreover data are not kept more than 6 months. (signature and encryption are provided by the generic policies).

- x:cctv-information → x is securely_stored(security-data-center)
- x:cctv-information → x is accessible(security-employee, administrator)
- x:cctv-information → x is deleted(x.date + 6 months)

#### 3.2.2.4 PF4U Employees Information

PF4U employees information policies are similar to the client information of the loan negotiation scenario. Except that some employee information are only available to the police department and PF4U.

1. PF4U Employees Information is used by PF4U, the Security Company and Police Department under the authorization of employees
   - x:employee-info ∈ m:message, m.to==y, a:authorization, a.subject==y, a.object == x → a is_signed(x.employee)
2. Any form of employee's authorization shall be provided by the employee's signature
   - x:employee-info, a:authorization, a.subject == x → a is signed($x$.employee)
3. Integrity of employee personal information is ensured by using signed documents.
   - m:message, x:employee-info ∈ m.contents → x is signed(x.origin)
4. Personal employee information can only be read by trusted entities:
   - m:message, x:employee-info ∈ m.contents → m is confidential(PF4U, police_department)

#### 3.2.2.5 Security Company Policies

1. The Security Company Employees can consult CC-TV logs and notify an anomaly through secure channels (signature and encryption are guaranteed by generic policies). Unverified anomalies are kept secret between PF4U and the security company to avoid public statements.
   - m:message, x:cctv-anomaly ∈ m.contents → m is_confidential(PF4U, security_company)
   - x:cctv-information → x is_accessible(security-employee)

#### 3.2.2.6 Police Department Policies

1. Policeman has access to the complaint folder information request the evidences to the PF4U
   - m:message, x:employee_info_request ∈ m.contents, y:evidences_requests ∈ m.contents → m is_confidential(PF4U, Police_Department)

2. Complaint folder information are strictly confidential between the police department and the PF4U company

    - x:complaint_folder $\rightarrow$ x is_confidential(PF4U, Police_Department)

## 3.3 Examples from Deliverable D1.3

This section discusses an example in Section 2.1 from Deliverable D1.3, which is about the confidentiality of data sent between different entities, and some variations on that example. In particular, we investigate different ways to implement the policy stating that *"The customer's subset data at the bank will be shared only between the bank and the European government"* (cf. page 6 of D1.3).

### 3.3.1 Policy Domain Hierarchy

We consider a simple hierarchy containing the three organizations (Bank, National government, European government). We can for instance reuse the same BBB_Bank and Government definitions from Use Case 1, and extend them with that of the European government as follows:

```
EU_Government (organization, top-level) {}
```

### 3.3.2 Security Policies

We consider several ways to implement the policy described above first in terms of a message only policy, then in terms of a usage control policy local to companies.

#### 3.3.2.1 Confidentiality of the Message

A first option is to define that no message will leave the Bank's policy domain if it cannot be rendered confidential and protected from any entity outside the European Government and the Bank.

```
Root {
   m:message, m.from==BBB_Bank, m.to==EU_Government,
   m.contents=x
    -> x is confidential(BBB_Bank, EU_Government)
}
```

This policy might for instance be enforced through the use of diverse concrete security mechanisms, either by forcing all messages to go through a secure channel (VPN) established between the bank and the EU Government for sending messages (although this is a bit more than originally requested), or by encrypting the payload of every message of the bank destined to the EU government. This policy however does not describe what the government might do thereafter with the data received from the bank, and it might for instance send them altogether to the national government (on purpose or by mistake, for instance in the writing of a business process).

#### 3.3.2.2 Confidentiality of the Data

Instead of the approach described above, we might combine a corporate-level policy, defining the semantics of data through extending the tags they carry, together with another policy describing how such data should be handled elsewhere in the system. Compared with the previous approach, where we used default data tags attached to the message, we instead want to attach tags directly to the data carried by messages then later processed within services. In that intent, we have to define whether the dataflow will remain tainted (meaning that the tag will be sticky and will mark other data resulting from processing an initial one), and whether the tag also percolates to nested data structures if any.

That first part of the policy will be as follows:

```
BBB_Bank {
   d:message, d.content==x
   -> x is sticky_nested_tagged(origin,BBB_Bank)
}
```

This local policy would come together with a root-level policy that would allow defining rules for all organizations based on data tainting (i.e., the tags attached to data). This root policy would roughly define how the three organizations define their collaboration and would typically be defined in the course of a legally binding contract.

The root policy might for instance state that any data originating from the bank would be confidential and distributed only to the european government :

```
Root {
   d.origin==BBB_Bank
   -> d is confidential(BBB_Bank, EU_Government)
}
```

This policy would ensure that any data originally sent from the bank to the EU government and its derivatives would be confidential. An enforcement would be required if they go across the EU_Government's boundary again and if the policy is correctly implemented. This thus ensures access control over transitive data flows and even some basic form of usage control. It would of course be possible to express the latter formula both at the bank and at the EU Government policy domains, but in that case, those formulas would be the same exact duplicate. In contrast, the formula above can be inherited by both domains by being expressed at the root.

# Chapter 4

# Related Work

To the best of our knowledge, there have been no previous attempts of expressing security properties (i.e., integrity, confidentiality, authentication, etc.) into a unified security policy language. However, by looking at our work as a security policy language we discover an amount of relevant related work. Some of this work tries to solve the problem using specific mechanisms, but most define unified frameworks in which different policies can be expressed [1, 3, 6, 4, 7, 13, 14, 2, 10, 15, 11]. What is lacking is a common language that will provide a unified approach to support the concept of specifying all (possible) security properties in one policy language.

Damianou *et al.* [7] describe the Ponder language for specifying security policies that map onto various access control policies which is similar to the one followed in SPL [13]. The primary point of contact between our work and this approach is the idea that it is possible to specify security policies based on the concept of domains. However, as mentioned above, Ponder is only designed for specifying and enforcing various kind of access control policies. Thus, it can not be used to specify and enforce other security policies (i.e., confidentiality). Another main difference from CSPL is that ponder does not support policy combination and conflict management issues.

Bauer *et al.* [3] define the Polymer language for composing complex security policies for Java applications. Polymer enforces security policies at run-time by monitoring and modifying the behavior of untrusted Java applications. Thus, Polymer is specification of policy enforcement rather then language for specifying system wide security policies.

Anderson [2] developed a Web Service Policy Language (WSPL), which is suitable for specifying a wide range of policies, including authorization, quality-of-service, quality-of protection, reliable messaging, privacy, and application-specific service options. WSPL is a strict subset of the XACML standard [9]. In essence, a WSPL policy is a sequence of one or more rules, where each rule represents an acceptable alternative. A rule contains a number of predicates, which correspond to policy assertions in WS-Policy. WSPL defines a standard language for use in specifying predicates that constrain domain-specified vocabulary items. Each predicate places a constraint on the value of an Attribute. Possible constraints are: equals, greater than, greater than or equal to, less than, less than or equal to, set equals and subset. Unfortunately, WSPL is not expressive enough to generally specify concrete (i.e., security mechanisms) such as the ones mentioned in chapter 2.

Becker *et al.* [6] introduce SecPAL, an authorization language to handle policy idioms, controlled delegation, recursive predicates and negated policies. In this language, fine-grained delegation control for decentralized systems are expressed using peridicates defined by logical clauses. However, like all other languages discussed above, SecPAL is also designed for access rights and management by keeping decentralized authorization and delegation model in mind.

Another approach to security policy languages is SPL, presented by Ribeiro *et al.* [13]. SPL uses complex constraints like history constraints, obligation constraints, and invariant constraints to specify access control language. FABLE [14], one of the formalisms for a programming language, does allow specifying security policies and ensure that these policies are properly enforced, but only handles access control, information flow, provenance, and security automata. FABLE specifies the semantic of labels in a separate part of the program as an enforcement policy.

Thomas *et al.* [15] describe a language to address security policies crossing different domains. The language makes possible to separate sender and recipient concerns by modelling information flow. Also, they deal with compound or flat data to address requirements in a nested structure. Furthermore, they designed the language around collection of users. Albeit the language is high-level, they sketch the mapping to more concrete policies such as Ponder and XACML. Authors designed the language with military constraints in mind, that clearly emerge domain separation and possibility to have multiple entities managing and defining policies.

Li *et al.* [11] propose yet another approach to provide Access control on web services. The framework uses a ReiT policy language based on rules and ontology. The ReiT policy language is based on the Rei policy language enhanced with temporal ontology. Also, authors try to address structured and non-structured knowledge.

One important difference between CSPL and these approaches on security policy language for heterogenous system is that rather than only enumerating high-level, application-specific, heterogeneous policies, our CSPL encapsulates two view points: *abstract security policies* and *concrete security policies*. Our security policy language is meant to express security policies at different levels of abstraction, from the abstract properties that the administrator wants to ensure to the concrete implementation details needed to specify the mechanisms that are adopted. Together, these features facilitate a coherent evolution from high-level requirements of security policies towards a concrete specification of mechanisms.

# Chapter 5

# Conclusion

In this document we introduced CSPL, the CESSA Security Policy Language, by way of an informal description of its semantics and of several examples. By applying it to our use cases of loan negotiation and home automation, we have shown that CSPL is expressive enough to describe succinctly and tersely our required policies. In particular, domain attributes enable describing role-based policies and information tags allow writing rules based on information flow.

In a future document, we will expand on this work by integrating a formal description of the CSPL semantics; several points are remain open for further work:

- Reasoning on the relationships between abstract and concrete policies. We can think of inference mechanisms to infer whether concrete policies implement the abstract policies required and reports inconsistencies that may appear; an even more ambitious goal would be taking into account engines that automatically derive concrete policies based on abstract ones. With respect to this goal, it will be interesting to explore the trade-off between language expressivity and complexity of automated reasoning.

- Handling policy conflicts. In this document, CSPL only adopts a simple conflict resolution rule. It is possible to think of more elaborate rules, that allow for example defining "hard" policies (which need to be applied in all cases) and "soft" policies (which can be overridden by more specific rules).

- Weaving concrete policies in service implementation. While up to now we defined mechanisms as implemented by a centralized reference monitor in each policy domain, it is possible to have different implementations (such as code weaving in existing service implementation or distribution of reference monitors) that can be more efficient in practice. Investigating the possibilities can be an interesting task.

# Bibliography


[1] M. Al-Morsy and H. Faheem. A new standard security policy language. *IEEE Potentials*, 28(2):19–26, March 2009.

[2] Anne H. Anderson. An introduction to the web services policy language (WSPL). *5th IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004.

[3] L. Bauer, J. Ligatti, and David Walker. A language and system for composing security policies. Technical report, Citeseer, 2004.

[4] M Becker and Alexander Malkis. A practical generic privacy language. *Information Systems Security*, 2011.

[5] Moritz Y. Becker, Cédric Fournet, and Andrew D. Gordon. Secpal: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[6] M.Y. Becker, C. Fournet, and A.D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. *Journal of Computer Security*, 18(4):619–665, 2010.

[7] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. *Policies for Distributed Systems and Networks*, pages 18–38, 2001.

[8] R. Douence, H. Grall, I. Mejía, et al. Survey and requirements analysis. Deliverable D1.1, The CESSA project, June 2010. `http://cessa.gforge.inria.fr/lib/exe/fetch.php?media=publications:d1-1.pdf` .

[9] T. Moses (ed.). extensible access control markup language (xacml) version 2.0. Technical report, OASIS Standard, 2005.

[10] Valerio Genovese, Dov M. Gabbay, Guido Boella, and Leendert van der Torre. Fsl – fibred security language. In Guido Boella, Pablo Noriega, Gabriella Pigozzi, and Harko Verhagen, editors, *Normative Multi-Agent Systems*, number 09121 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany.

[11] Jian-Xin Li, Bin Li, Liang Li, and Tong-Sheng Che. A policy language for adaptive web services security framework. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on*, 1:261–266, 2007.

[12] Carlos Ribeiro and Paulo Ferreira. A policy-oriented language for expressing security specifications. *International Journal of Network Security*, 5(3):299–316, 2007.

[13] Carlos Ribeiro, A. Zuquete, Paulo Ferreira, and Paulo Guedes. SPL: An access control language for security policies with complex constraints. In *Proceedings of the Network and Distributed System Security Symposium*. Citeseer, 2001.

[14] N Swamy and BJ Corcoran. FABLE : A Language for Enforcing User-defined Security Policies. *2008 IEEE Symposium on Security*, pages 1–48, 2008.

[15] R. Thomas and S. Tsang. Cdl: A language for specifying high-level cross-domain security policies. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1 –7, nov. 2008.