TripCom

Triple Space Communication

FP6 – 027324

Deliverable

D5.2

Definition of security and trust support model for the reference architecture

Alessandro Ghioni
Davide Cerri
Francesco Corcoglioniti
Jacek Kopecký
Gerson Joskowicz
Lyndon Nixon
Dario Cerizza
Noelia Pérez Crespo

October 19, 2007

Executive Summary

This deliverable defines the first version of the security and trust model for TripCom, following the state of the art and requirement analysis of deliverable D5.1. The core of the model is the Triple Space policy, which governs access to triplespaces by defining access control rules and trust and attribute mapping rules. We also define how security processing relates to TripCom kernel architecture (modelling the Security Manager functionalities and the "information flow" related to security processing inside the kernel), and a model for reputation information.

Document Information

IST Project Number	FP6 – 027324	**Acronym**	TripCom
Full Title	Triple Space Communication		
Project URL	http://www.tripcom.org/		
Document URL			
EU Project Officer	Werner Janusch		

Deliverable	**Number**	5.2	**Title**	Definition of security and trust support model for the reference architecture
Work Package	**Number**	5	**Title**	Security and Trust

Date of Delivery	**Contractual**	M18	**Actual**	31-Oct-07
Status	version 1.0		final ☒	
Nature	prototype ☐ report ☒ dissemination ☐			
Dissemination Level	public ☒ consortium ☐			

Authors (Partner)	Davide Cerri (CEFRIEL), Alessandro Ghioni (CEFRIEL), Dario Cerizza (CEFRIEL), Francesco Corcoglioniti (CEFRIEL), Jacek Kopecký (LFUI), Lyndon Nixon (FUB), Noelia Pérez Crespo (TID), Gerson Joskowicz (TUW)			
Resp. Author	Alessandro Ghioni		**E-mail**	ghioni@cefriel.it
	Partner	CEFRIEL	**Phone**	+39 02 23954 212

Abstract (for dissemination)	This deliverable defines the first version of the security and trust model for TripCom, and in particular the Triple Space policy.
Keywords	Triple Space computing, security, trust, policy, access control, reputation

Version Log			
Issue Date	**Rev No.**	**Author**	**Change**
23 April 2007	1	Davide Cerri	First version, general view
25 May 2007	2	Alessandro Ghioni	New structure
31 August 2007	3	all	Draft version
19 September 2007	4	all	Final draft for QA
11 October 2007	5	all	Revised version after QA comments
19 October 2007	6	Davide Cerri	Final version

Project Consortium Information

Acronym	Partner	Contact
Leopold Franzens University Innsbruck http://www.deri.at	LFUI	Prof. Dr. Dieter Fensel Digital Enterprise Research Institute (DERI) Innsbruck, Austria E-mail: dieter.fensel@deri.org
National University of Ireland, Galway http://www.deri.ie	NUIG	Dr. Laurentiu Vasiliu Digital Enterprise Research Institute (DERI) Galway, Ireland Email: laurentiu.vasiliu@deri.org
University of Stuttgart http://www.iaas.uni-stuttgart.de/	USTUTT	Prof.Dr. Frank Leymann Inst. für Architektur von Anwendungssystemen (IAAS) Stuttgart, Germany E-mail: frank.leymann@informatik.uni-stuttgart.de
Vienna university of Technology http://www.complang.tuwien.ac.at/	TUW	Prof.Dr. eva Kühn Institut für Computersprachen Vienna, Austria E-mail: eva@complang.tuwien.ac.at
Free University Berlin http://www.ag-nbi.de/	FUB	Prof. Dr.-Ing. Robert Tolksdorf AG Netzbasierte Informationssysteme Berlin, Germany E-mail : tolk@inf.fu-berlin.de
Ontotext Lab, Sirma Group Corp. http://www.ontotext.com/	ONTO	Atanas Kiryakov, Vassil Momtchev, Ontotext Lab, Sirma Group Corp. Sofia, Bulgaria E-mail: vassil.momtchev@ontotext.com
Profium OY http://www.profium.com/	Profium	Dr. Janne Saarela Profium OY Espoo, Finnland E-mail: janne.saarela@profium.com
CEFRIEL SCRL. http://www.cefriel.it/	CEFRIEL	Davide Cerri CEFRIEL SCRL. Milano, Italy E-mail: cerri@cefriel.it
Telefonica I+D http://www.tid.es/	TID	Noelia Pérez Crespo Telefonica I+D Madrid, España E-mail: npc@tid.es

Table of Contents

List of Abbreviations

AC	Access Control
AP	Attribute Provider
DBMS	Data Base Management System
OASIS	Organization for the Advancement of Structured Information Standards
PDP	Policy Decision Point
SAML	Security Assertion Markup Language
SM	Security Manager
SM-AC	Security Manager – Access Control
SM-Authn	Security Manager – Authentication
SM-TAM	Security Manager – Trust and Attribute Mapping
SSL	Secure Socket Layer
SSO	Single Sign-On
TAM	Trust and Attribute Mapping
TLS	Transport Layer Security
TS	Triple Space
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

1 Introduction

1.1 Scope and motivation

This deliverable defines the first version of the security model for TripCom, following the state of the art and requirement analysis of deliverable D5.1 [4]. The core of the model is the Triple Space policy, which governs access to triplespaces by defining access control rules and trust and attribute mapping rules. We also define how security processing relates to the TripCom kernel architecture, modelling the Security Manager functionalities and the "information flow" related to security processing inside the kernel.

Work on the design of TripCom distributed infrastructure is currently in progress in other workpackages, therefore in this version of the security model we refer to the single-kernel situation, and we just sketch some ideas about the evolution of the security model towards a distributed (multi-kernel) infrastructure. The design of the security model for the distributed infrastructure will be addressed in subsequent tasks (T5.4).

The requirement analysis [4] revealed a strong need for access control functionalities that had not been previously anticipated; on the other hand the use cases do not exhibit a particular need for elaborate trust and reputation functionalities. It is worth noting that TripCom deals with a general purpose infrastructure that cannot take decisions in spite of services, as these are up to single, domain dependent applications. Therefore, TripCom security infrastructure will provide fundamental general purpose security features. From a platform perspective, the way to vouch for trust and security towards its users is first of all by providing suitable policy mechanisms. These are indeed needed by clients in order to grant or deny access to the spaces they use for mutual interaction. For this reason, we slightly refocused WP5 objectives in order to better address access control needs, designing a security policy model. Regarding reputation, we will provide a model of reputation information that can be exploited by services as a foundation to build a trust management system, but TripCom infrastructure cannot take trust decisions that pertain to client-to-client interactions.

Finally, it is important to point out that we do not aim at defining a new general purpose security system or security model. The goal of this work is to investigate how we can apply security to the Triple Space infrastructure, how we can adapt security solutions to triplespaces, and how we can apply the space-based technology of the kernel to a security system.

1.2 Document structure

This document is structured as follows. Chapter 2 provides a general view of the model, introducing the main concepts that will be then detailed in chapters 3 (client authentication, trust and attribute mapping) and 4 (access control). Chapter 5 deals with logging and auditing issues, while chapter 6 defines the scope of trust and reputation in TripCom infrastructure. Finally, chapter 7 presents initial ideas towards a distributed architecture, and chapter 8 discusses how use cases relate to security functionalities.

2 General view

In order to define the security model for the TripCom architecture, we will start illustrating the basic principles, actors and scenarios involved in authentication, authorization and accounting. This chapter will give a general view of how security will work in TripCom, introducing concepts that will be detailed in next chapters. Sections 2.1 and 2.2 will introduce the Triple Space policy and the different authorities that are involved, section 2.3 will discuss the role of infrastructure security and related assumptions, while section 2.4 will present Security Manager functionalities. Finally, section 2.5 will discuss the "information flow" related to security processing.

2.1 Triple Space policy

Access to triplespaces is governed by *policies*: each (sub)space has its own policy, which governs access to data contained in it[1]. We define a Triple Space policy ontology, an extension of the Triple Space ontology defined in WP2 [5]. The domain and scope of the ontology extension are similar to those of the TS ontology. Different parts of the policy ontology, dealing with trust and attribute mapping (TAM policy) and with access control (AC policy), are defined in different chapters of this document; the whole ontology is listed in Appendix A. Concepts from the TS ontology from WP2 are referred to with the prefix `ts:`; properties are noted as `DomainClass propertyName <cardinality> RangeClass`.

Triplespace policies are stored in the persistent RDF storage (they are not volatile data), but they cannot be accessed as any other data exposed through the TS API. We therefore refer to a "security space" concept, which represents a space for policy data of a ts:Space.

SecuritySpace

The security space is outside the normal ts:Space hierarchy (as seen by clients), i.e. it is not accessible using the TS-API operations. The security space contains trust filtering rules, mappings from authentication attributes to roles, and an access control policy set.

```
ts:Space hasSecurityData <1> SecuritySpace
SecuritySpace hasTrustFilteringRule <0..*> TrustFilteringRule
SecuritySpace definesRole <0..*> Role
SecuritySpace hasRoleMapping <0..*> RoleMapping
SecuritySpace hasPolicySet <1> PolicySet
```

2.2 Authorities

A key concept for security is represented by the *authority*, namely the entity that defines rules and policies for a given portion of the Triple Space. The requirements revealed only one kind of authority, the one which could define the security for the

[1]The space hierarchy is used in policy evaluation, so a subspace policy is not completely independent of its parent's one. Policy combination in the hierarchy of subspaces will be detailed in Chapter 4.

system without separating between a data layer and an infrastructural layer, but, as we will illustrate, it is useful to split this concept in two, thus resulting in the distinction between:

- *infrastructure authorities*, which operate at infrastructural level: they run kernels, manage components and their security properties, and regulate user rights related to putting a (sub)space on a particular kernel;
- *data authorities*, which operate at data level: they manage security policies related to (sub)spaces and (as a consequence) to triples.

To clarify this distinction with a well-known example, we can use the scenario of information publishing on the Web. In this example, we consider two actors: a user who wants to publish some data on the Web and a Web hosting provider, which owns some machines connected to the Internet. Once a trust relationship between the two actors has been established (e.g. by signing a contract), the hosting provider allows the user to use some resources on its machines, in this case by allowing publishing of web pages. The hosting provider is responsible for things like the security of the machines and of the communication between the web server and the DBMS. Once acquired the right to use the resources, the user is free to manage his data as he wants, since he is the data owner. Using an application server, he can deploy a web application and define its users and its security policy. Using the above distinction, we can clearly see the web hosting provider as an infrastructure authority, and the user as a data authority.

In the TripCom scenario, the infrastructure is constituted by a set of kernels, while data is represented by the (sub)spaces and the contained triples. Therefore, we can define a *kernel authority* on the infrastructure level, and a *space authority* on the data level. A user has to establish a trust relationship with a kernel authority in order to have the permission to create a space on a kernel. Once the space has been created, the user becomes a space authority as he can define the users who can access data contained in his own space and the security policy that has to be enforced on the space itself.

Dealing with multiple triplespaces and multiple kernels is different from dealing with a triplespace (and its subspaces) on a single kernel. A scenario composed by many kernels arises issues about how a single request is distributed among the kernels as well as which operations require trust between different kernel authorities. This document focuses on the single kernel scenario, since TripCom distributed architecture has not been designed yet. Before starting this description, it is necessary to define the "who, where and what" of security by defining the actors, the data objects on which the operations will be performed, and the operations themselves.

The requirements state a minimum of two actors in security, namely the *client* and the *Security Manager* component. The Security Manager main functionalities are authentication and authorization, and it implements the *policy decision point* (PDP).

Data objects on which operations are performed consist of spaces and triples. Spaces are logical containers arranged in a tree-like hierarchy, while the actual data is composed by triples. From a security point of view, the main difference between a triple and a space consists in the latter being associated to its own policy, while the former is managed according to the policy of the space(s) it belongs to. Therefore, operating on a triple requires only evaluating and enforcing a pre-existing policy, while

creating a space may imply the creation of a *security object* (i.e., the policy). Another important difference is about ownership: each space has an *owner*, i.e. its data authority, who can set and modify the policy on that space. Operations on triples and spaces are defined by the TS API, and allow users to manage creation and destruction of spaces as well as querying, adding and deleting triples in spaces.

2.3 Infrastructure security

In this document we mainly refer to security for the data layer, i.e. on how data authorities can define their security policies, and on security mechanisms that evaluate and enforce these policies in the infrastructure. TripCom reference architecture [8] specifies a kernel made of several components, that coordinate themselves using an internal space-based middleware. The security of the communications between these components must be taken into account, but these issues can be solved with standard techniques like X.509 certificates and TLS/SSL channels.

We can identify the following requirements/assumptions for infrastructure (i.e. intra-kernel) security:

- all legitimate components are trusted, i.e. not malicious;
- the coordination middleware authenticates components before they can "join" the kernel, and lets only legitimate (authorized) components join;
- communications channels between components are secure (authentication, integrity, confidentiality);
- there is no way from the outside of the kernel to directly access the coordination middleware (security perimeter).

Ensuring all these is responsibility of the kernel authority, who will accordingly configure the kernel components (in particular the internal coordination middleware) and possibly the network infrastructure (e.g. firewall, VPN, etc).

In the following, when talking about authentication and authorization, we will refer to authentication and authorization of *clients*, and not of kernel components.

2.4 Security Manager functionalities and internal organization

As we said before, the Security Manager (SM) is the kernel component in charge of checking clients' identity and give them appropriate permissions. More precisely, we can identify three different functionalities in the Security Manager:

- *Authentication* (SM-Authn): performs an authenticity check on client's credentials and other supplied information;
- *Trust and Attribute Mapping* (SM-TAM): filters client's supplied information (identity attributes) based on trust relationships with asserting parties, and maps these attributes to roles for access control, according to the TAM policy;
- *Access Control* (SM-AC): takes access control decision, according to the AC policy.

These three functionalities can be implemented as three different subcomponents. The following subsections will detail their role.

2.4.1 Client authentication, trust and attribute mapping

Authentication is the first stage of the security process that ends in the decision about whether to allow or not the execution of a given TS API operation.

Besides the client and the Security Manager, we introduce a third actor in this process: the *Attribute Provider* (AP). The AP is the entity that holds and certifies the client's identity attributes, which will be used by the Security Manager in order to give the right roles (and consequently permissions) to the client. An attribute can be either a label, representing some state or class of clients, or a name – value pair, which represents and specify some characteristics of the client. The AP can be seen as an external service with respect to the rest of TripCom architecture (it can be a SAML [10] asserting party, while the kernel is the relying party in SAML terminology). The basic principle of the Attribute Providers is that there is a set of actors who know the clients and can vouch for their identity attributes, issuing appropriate *assertions*.

Being the AP an external service, trust between it and the Security Manager is not implicit, but it has to be modelled in an appropriate policy. This policy allows the Security Manager to validate all the assertions it receives by stating, for example, that only certain Attribute Providers are trusted.

Since the AP is an external service, attributes certified by the AP may not correspond exactly to roles used in the AC policy, therefore a mapping between assertion attributes and AC roles is performed.

Summing up, the whole process, which conceptually takes place before AC policy evaluation, has three phases:

1. authentication (performed by SM-Authn): the client's credentials and assertions are validated (i.e. authenticity check);

2. AP trust (performed by SM-TAM): client's assertions are "filtered", discarding untrusted statements;

3. attribute mapping (performed by SM-TAM): attributes found in assertions are mapped to roles found in AC policy.

The second and third phases must be governed by a policy. The assertions are used in conjunction with an operation request, which has as target a space, to take decision about access to that space. Therefore, this policy has to be linked (and thus to refer) to the space and not to the kernel that is handling the request, because it is a matter of data, and not of infrastructure. Being linked to the specific space, this policy must be defined by the data authority of the subspace itself. This policy is called the *TAM policy*, and is a part of the Triple Space policy (the other part being the *AC policy*).

Client authentication, trust and attribute mapping will be discussed in more detail in chapter 3.

2.4.2 Authorization

Authorization is the second stage in deciding whether or not to allow the execution of a given operation. Authorization decides whether the client, whose identity attributes

are established in the first stage (authentication), has the appropriate access rights. As stated in section 2.1, every (sub)space is associated with a security policy.

According to the requirements of a role-based access control, authorization policies are written in terms of client roles, and spaces are also associated with role assignments — mappings from the user identity attributes to the roles. The AC policy is related to the space, and not to the kernel, because it deals with access to data. It must be defined by the data authority (space owner). The policy regulates the operations defined in the TS API on the space it is linked to, and also on the hierarchy of its subspaces (depending on the policy combining rules). Policy management functionalities should be provided by the Management API.

Apart from the reading and writing operations of the TS API, the security policy of a space also has to control the creation and deletion of subspaces of the space.

In the general view, the authorization subcomponent receives information about the operation requested, the space in which the operation should be executed, and the authenticated attributes. Based on the policies of the space and its ancestor spaces, the user authentication attributes are mapped to a set of roles (potentially different in the scope of each policy), and then the access control decision is taken based on the roles and the rights assigned to those roles in the policies.

2.5 Information flow

2.5.1 The Security System as a space based application

The security component providing authentication and authorization functions is itself an application that leverages the capabilities of space based computing. The components of the security sub-system interact via the XVSM based backbone [3, 12], which is based on (non-semantic) space technology.

Thus, the understanding of the information flow within the security sub-system is based on the very same principles used in any space based system; its sub-components interact by coordinating their actions through *in's*, *out's* and *rd's* on structured spaces. The use of multiple subspaces allows us to isolate inner parts of the system — i.e. parts that trust the information they process — from outer parts of the system — i.e. parts that need to perform security checks before processing.

XVSM provides a couple of useful features:

- XVSM supports ordered access to a subspace, (FIFO and LIFO ordered access) on top of a "selector". This feature allows the implementation of various queuing mechanisms like regular queues, topic queues, etc.
- XVSM supports a blocking wait operation for a set of tuples that satisfy a "selector". This feature is essential to implement joins or recombination of workflows.
- XVSM supports a bounded buffer model used to implement back-pressure. This feature allows to control resource allocation and to manage overload conditions.
- XVSM allows for attaching tuples as subspace properties. These tuples are maintained in attached and extendable meta-spaces attached to the subspace. This feature can be used to handle access information as part of the meta-information maintained for a subspace.

With the space based infrastructure in place, we are able to model the information flow. The architecture of the information flow is inspired by the SEDA (Staged Event Driven Architecture) model [11]. The SEDA model proposes to decompose the flow of a business process into a network of individual stages separated by queues.

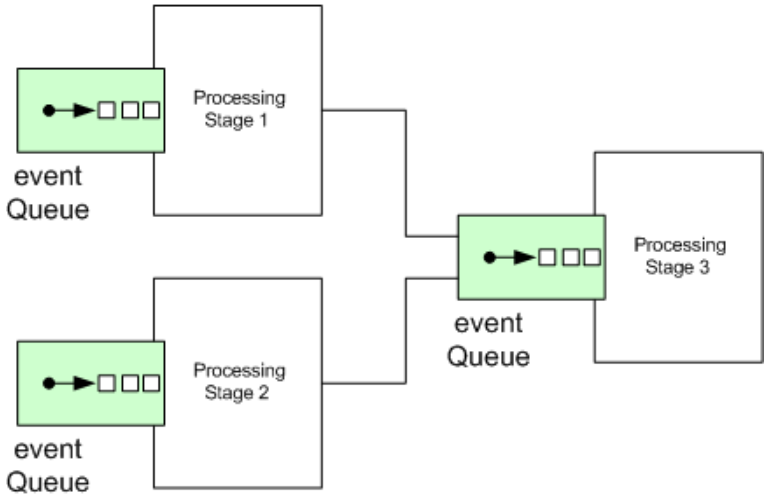

Figure 2.1: Simple SEDA arrangement

The subspaces placed in the XVSM kernel take the role of SEDA's event-queues. As mentioned above, XVSM supports directly LIFO / FIFO coordination, allowing a straight forward implementation. The destructive and blocking *rd* corresponds to a wait for an event to be processed by the processing stage.

The processes must follow some simple principles that have been stipulated for this type of architecture:

- no data sharing: data sharing is limited to data maintained in (sub)spaces, i.e. queues;
- stateless task processing: each processing stage must be stateless; states are only maintained in (sub)spaces;
- no fate sharing: prevent a processing stage to fail due to the failure of another processing stage. Such failures may occur if multiple processing stages share common resources.

2.5.2 Applying XVSM to the authentication information flow

Figure 2.2 sketches the information flow for the TS Authentication process.

A client issues a TS-API request. The request is transported to the TS kernel — we omit the details of the transport since from the point of view of the Security Manager the system is transport agnostic. The request consists of the API as described in [8] wrapped by an envelope which holds the necessary security information.

The TS-API component has the responsibility to capture the request and to insert the request into a special *non-authenticated* subspace. The subspace is embodied by an XVSM-container. Each stage may produce a response either in the form of data like in the case of a *rd* or *out* or in the form of a status for *rd* like "rejection", "security violation", etc.

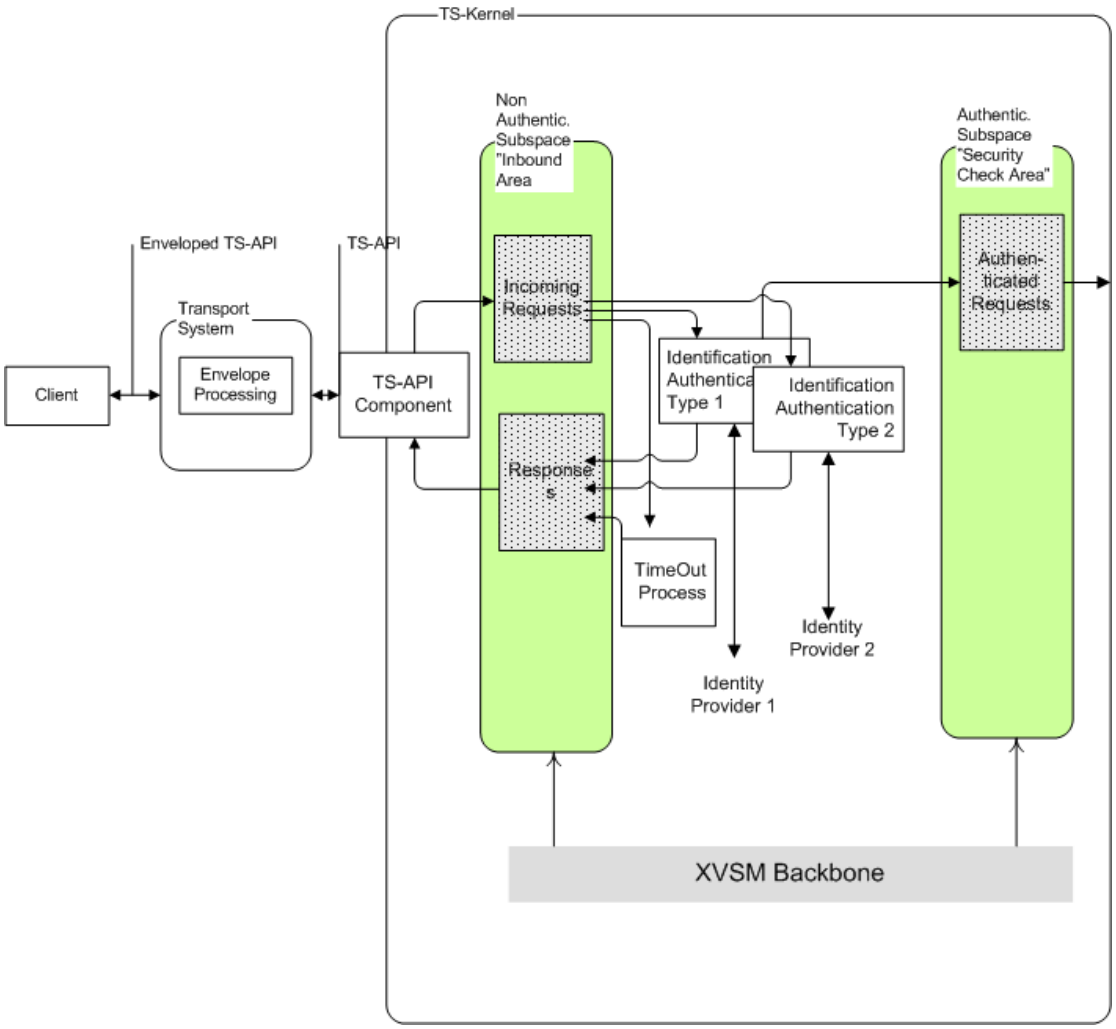

Figure 2.2: Sketch of authentication

Each stage contains processes that work on requests stored in specific subspaces. The system must ensure that only specific, privileged processes are allowed to access data that have not been validated. The subspaces that serve the Security Manager[2] are therefore grouped in "areas". An "area" groups all subspaces that allow a certain group of processes to access them. Thus, from a conceptual architecture point of view, areas defined on subspaces correspond to security zones in message based infrastructures. The *Inbound Area* would correspond to an external zone, the *Security Check Area* would correspond to a DMZ (Demilitarized zone).

Requests entering the *non-authenticated* subspace are processed by a SM-Authn process. In analogy to the SEDA partitioning pattern, the SM-Authn process can be implemented by different types of authenticators. The splitting mechanism is supported by simple template matching e.g. each identification/authentication processing stage might issue a *rd* request with a template match to the sub-space holding the incoming requests:

- The request could be processed by one of a set of specific authenticators that specialize in specific authentication types. This is indicated in figure 2.2 by

[2]These spaces are in the coordination middleware (XVSM) and are only for internal use of the kernel; they are not triplespaces accessible by clients via TS API.

"Type 1" and "Type 2". This allows for multiple authentication schemes to be applied, which is similar in concept to the pluggable authentication architecture used in mainstream systems like UNIX or Windows.

- A request holding an authenticated cookie containing the proof of a previous authentication that is still valid, also be directly processed by a processing stage that validates these authenticated cookies only. This mechanism provides the quick validation of already established application sessions.

- Requests that are not processable or not processed after a given time are collected by a "Timeout process" that handles the required "timeout" and error response.

The proposed architecture requires the following specific features:

- Subspaces must be defined that correspond to the security level. We need subspaces that hold non-authenticated and non-validated data, as well as subspaces that hold authenticated but non-authorized data.

- The result of each processing stage (authentication, authorization, etc) is appended to the data and moved to the next stage.

- The access of processes to specific subspaces is controlled by kernel mechanisms. The XVSM extendable meta structures are usable to cover this requirement. Thus, the inbound area (subspace) will only be accessible to SM-Authn processes, the security check area (subspace) will only be accessible to SM-TAM and SM-AC processes.

As stated later in chapter 4, the tuples emitted into the space by the Security Manager would contain an "applicative" part and a "security" part. The "security" part is filled by the Security Manager and expanded as the requests passes through the security related processes.

2.5.3 Applying XVSM to the authorization information flow

Figure 2.3 shows how the information flow applies to the security check stage.

Authenticated requests are moved by the SM-Authn process to the subspaces of the *Security Check Area*. These subspaces are accessed by the SM-TAM and the SM-AC processes. The use of space based principles allows to implement cooperative behaviour between SM-TAM and SM-AC. As described in section 2.4 these processes apply Trust and Attribute Mapping as well as Access Control to the requests. Both processes take from (*rd* operation) and write to (*out* operation) the *authenticated requests* subspace in the *Security Check Area*. As described in section 3.3.2 the authorization process requires the cooperation of SM-TAM and SM-AC. The final authorization decision is performed by SM-AC, while SM-TAM resolves the attributes that are necessary for the decision making process. Thus, the processes may cooperate using simple rules:

- Non-authorized requests are taken from the *authenticated requests* subspace. The SM-AC process inspects these requests and checks if all the necessary preconditions are met to perform the decision.

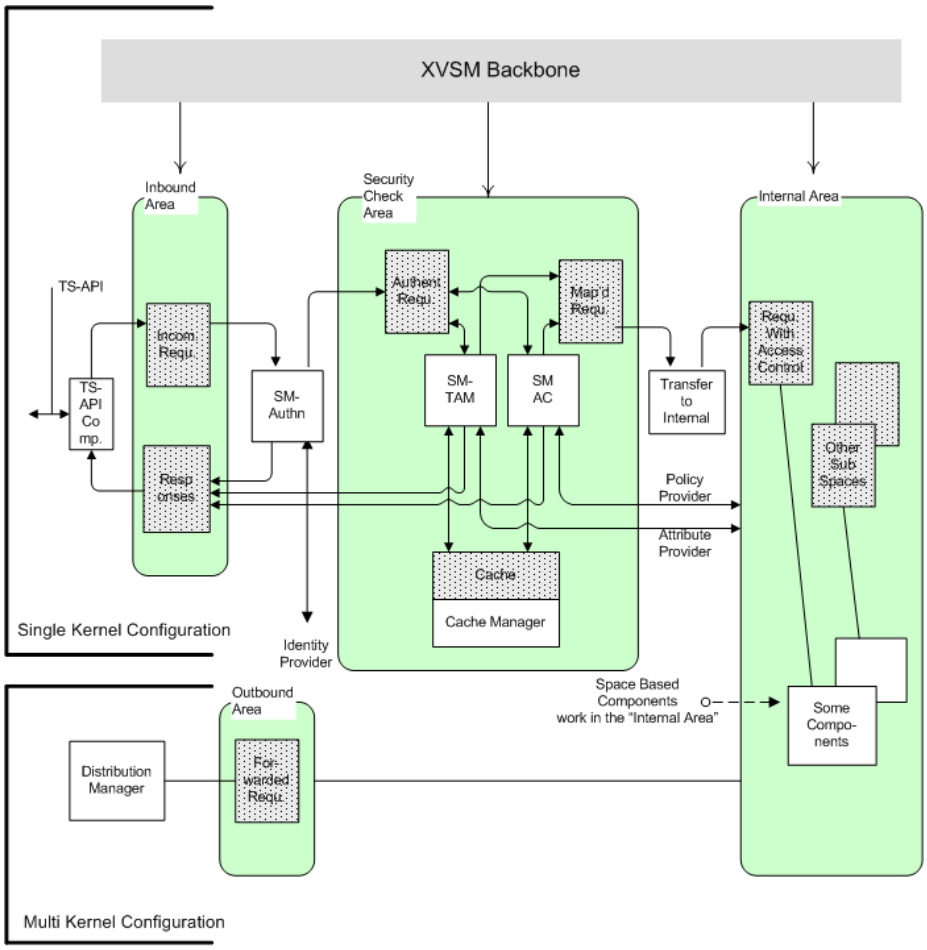

Figure 2.3: Sketch of security flow

- A request that cannot be authorized due to missing information is written back to the *authenticated requests* subspace. The SM-TAM process picks up requests that are flagged for missing attributes. The use of a single subspace allows to "bounce" unresolved requests between SM-AC and SM-TAM until the authorization is decidable.
- A request that is fully authorized is moved to the *Mapped requests* subspace. It will be moved by a special, privileged process to space into the *Internal Area*.
- Either process may reject the authorization and may generate a negative response that is collected in the *Inbound Area*, although the SM-AC process is the decision making process. However, fatal errors in the SM-TAM process should stop the authorization process and generate a negative response.

SM-TAM and SM-AC type processes are required to navigate other spaces in order to retrieve the necessary information to perform the attribute mapping and to set the access control information. This is performed via provider interfaces. The insertion of provider interfaces allows the SM-AC and SM-TAM processes to reach different information sources:

- Processes in the *Internal Area*. The sub-spaces in the *Internal Area* and their security related meta information are obviously the main source of information for the SM-AC and SM-TAM processes.

- Outside information sources that need to be federated into the security system like LDAP based directories.

The SM-AC and SM-TAM processes will require some simple caching mechanisms for performance reasons. Both processes depend on the retrieval of triples for each invocation. Security related information on subspaces as well as security information on request originators changes slowly and are excellent candidates for caching. A subspace based on XVSM together with a cache manager component implement the caching mechanism. The mechanism leverages on three properties of XVSM:

- the possibility to apply multiple coordination types on a subspace;
- the bounded buffer model of XVSM that defines the cache capacity;
- "aspects" that can be bound to interactions with the subspace. An "aspect" is a function that is executed before or after an interaction with the subspace. Conceptually, "aspects" are equivalent to triggers in relational databases. The cache manager is implemented as a set of functions bound to *in* and *out* operations.

The SM-TAM and SM-AC processes are able to place information in the cache. The task of the cache manager is:

- To manage the cache according to classic replacement policies like LRU (Least recently used), MRU (Most recently used) or LFU (Least frequently used) policies. Such policies should be configurable.
- To invalidate cached information after a configurable timeout.
- To pass on requests that the cache cannot satisfy.

Finally the authenticated and authorized requests are transferred to the *Internal Area*. The *Internal Area* holds all subspaces accessible by all non-privileged processes. These processes implement the Triple Space application.

So far this description covers the case of a single kernel. In the next project phases, the architecture will support multi-kernel, distributed configurations. Such configuration requires the introduction of an additional area, the "Outbound Area". The *Outbound Area* is reserved for access by the distribution manager which forwards requests to other kernels. The *Outbound Area* implements therefore the security mechanisms between kernels. See deliverable D6.3 for the details of the distributed architecture.

2.5.4 Rationale for the proposed architecture

The rationale for the proposed architecture is based on the following:

1. The Security Manager leverages on the XVSM capabilities and is based on typical patterns for space based business processes.
2. The Security Manager is composed of independent, individually configurable components that can be replaced — even dynamically at runtime.
3. The resources (number of processing instances) can be adapted to the operational requirements, like current load or current availability of resources.

3 Client authentication, trust and attribute mapping

3.1 Main concepts

We already introduced the main actors involved in the authentication, trust and attribute mapping process, i.e.:

- the *client*, who wants to perform some operation on a triplespace, and therefore must be authenticated by a Triple Space kernel;
- the *Attribute Provider* (AP), i.e. an external service that knows the client and is able to vouch for the client's identity attributes;
- the *Security Manager*, i.e. the kernel component that performs authentication and authorization, logically divided into SM-Authn, SM-TAM, SM-AC (see chapter 2).

Moreover, we introduce three pieces of data related to the process, i.e.:

- an identity *assertion* is issued by an Attribute Provider and states the client's identity and attributes. It is used by the client, along with some secret information (e.g private key), in order to prove its identity and relevant attributes to a kernel. The assertion must have a standardized format, because it must be understandable by any kernel.
- a *security context* is established in the kernel as the result of the authentication & TAM process, in order to keep track of an established "session" and to temporarily store security data related to this session (e.g. the assertion provided by the client, the client's roles, etc.), in order to avoid to repeat the same processing for subsequent operation requests.
- an *access cookie* is returned by the kernel (Security Manager) to the client as a result of a positive authentication attempt. The cookie is used as a "shortcut", in order to avoid the need for the client to explicitly authenticate itself in all subsequent messages (and therefore the need for the server to validate the client's assertions). From the client's point of view, it is an "opaque" piece of information, since it is just a reference to the security context maintained by the Security Manager.

The client authentication, trust and attribute mapping process consists of three phases. In the first phase (authentication), the SM receives the client's assertion(s) and checks if they are valid and if the client can claim their possession. This includes checking the AP signature on the assertion and verifying if the client is the legitimate holder of that assertion (e.g. by checking a proof of possession of a private key). In the second phase the Security Manager takes the client's identity assertions and, basing on the AP from which they come, decides whether to trust them or not[1]. In the third phase, the Security Manager takes the trusted attributes that result from the previous

[1]The Security Manager may also decide to trust only part of the assertion, since the TAM policy can specify that the AP is only trusted for certain attributes and not for others.

phase and maps them into roles that are relevant for the access control policy, according to the TAM policy. The attribute mapping is performed because the access control policy can be application-dependent and therefore refer to its own roles, which are meaningful to that particular application and may not be the same as the attributes found in the assertions provided by the AP (which can be application-independent).

If the client issues multiple operation requests targeted at the same space, all this process does not need to be repeated, which is not true for access control (an AC decision must be taken for each client request). This is because the authentication and TAM process results depend on the assertion and on the target space of the operation, but not on the type of operation (e.g. reading vs. writing). This is the reason why the authentication and TAM process establishes the security context, which can be used for subsequent operations by the same client without repeating the aforementioned process. The cookie which is returned to the client is a reference to the security context, and can be used by the client instead of the assertion (for a limited time).

Attributes contained in assertions are mapped to roles, which are in general not one-to-one with clients. This is useful for access control, because AC policy can be specified in terms of roles rather than single clients. Nevertheless, an "identity" information, i.e. some attribute that is related to each particular client and can identify it, is often needed. Therefore, such an attribute may be required in the assertion and tracked even if it is not mapped to any role and is not used by AC policy.

3.2 Authentication

An operation request, written to the inbound area of the kernel, is always taken first by SM-Authn for authentication purposes. A client may also submit an operation request without any attribute information: in this case the request will be directly passed to access control decision, and then accepted or rejected according to AC policy rules for anonymous clients (the default behaviour being to deny the request).

In order to access private spaces (i.e. spaces where anonymous clients are not allowed), or spaces for which anonymous access is restricted (e.g. anonymous clients can read but cannot write), the client needs to supply an assertion that states its identity and relevant attributes, and to prove that it is the legitimate owner of this assertion. The assertions are issued by Attribute Providers; we plan to use SAML for assertion format[2] (see section 3.5).

From an architectural point of view, it has currently not been defined yet if Triple Space clients will contact a Triple Space kernel using a particular "Triple Space transfer protocol", or if the TS API will be completely "protocol agnostic" and no particular protocol will be specified. This has a significant impact on the authentication mechanism, because in the first case we can design an authentication protocol as part of (or strictly bound to) the Triple Space transfer protocol, otherwise we must follow a "protocol agnostic" approach as well, and just assume there will be some kind of enveloping mechanism for self-contained authentication messages. Therefore, we skip the design of a precise authentication solution at this moment, and just state the role of authentication in the whole security process.

[2]The interaction between the client and the AP is out of scope for TripCom.

3.3 Trust and attribute mapping

3.3.1 TAM policy

The trust and attribute mapping process is governed by the space policy. The Triple Space policy can be considered as composed by two parts, namely the TAM policy and the AC policy: the TAM policy is described in this section, while the AC policy will be described in chapter 4. These two parts are distinct because they regulate two different aspects, but they are defined by the same entity (the data authority, i.e. the space owner) and, jointly taken, they govern access to the same data. Another difference is that the AC policy evaluation result may be a "permit" or a "deny", so we need a way to resolve potential conflicts between policy rules (through rule combining algorithm) and between policies of parent and child spaces in a hierarchy (through policy combining algorithms), if they are in contrast. This is not the case for the TAM policy, since TAM policy evaluation can only "add" things, and there is nothing similar to permit/deny (from which conflicts may arise). The link between the TAM policy and the AC policy is given by roles.

The TAM policy is defined at the data level, and each space has its own TAM policy defined by the space authority (just like the AC policy), whose rules are inherited through the space hierarchy, as explained below. This policy governs the TAM process, since it defines:

- which Attribute Providers are trusted, and for which attributes or attribute–value pairs;
- the attribute mappings rules between the identities and attributes found in identity assertions and the roles found in the AC policy.

The simplest form of the attribute trust part of the TAM policy is a set of trusted Attribute Providers, possibly with restrictions on attributes. More complex possibilities, taking into account hierarchical relationships and webs of trust, will be added in the refinement of this model. In any case, this acts like a filter that excludes untrusted statements about attributes. In this version of the TAM policy model, attribute names and values are strings.

Attribute mapping rules allow to say that a given set of attributes maps to a given role. Every role mapping specifies a set of attributes and their value and the set of roles that are assigned to the clients that have the attributes with those specified values. This simple mechanism allows expressing a conjunction of attributes for a role, e.g. that attribute "education" with the value "medical doctor" maps to role "doctor" only if the presence of the attribute "degree-accredited" indicates an accredited institution.

A role is "visible" only in the space that defines it[3] and in all of its subspaces; likewise, attribute mapping rules defined in the TAM policy of a space are inherited in all its subspaces. Note that the same role cannot be defined more than once in a path from any (sub)space to the root. Otherwise, unwanted behaviours may occur with role mapping inheritance, as in the following example. Let us consider a space `/a/1`, which is a subspace of space `a`, that defines a role `r` and specifies in its TAM policy that attribute `A1` is mapped to role `r` and in the corresponding AC policy that role `r` can read. If a space owner defines a role (such as `r`), he expects he is the first one to

[3]We say that a space *define* a role if it is created and stored in the associated security space.

define it and he assumes to have full control over it. But, if a new mapping rule from attribute `A2` to role `r` is added in the TAM policy of parent space `/a`, this will be valid also in space `/a/1`, therefore a client with attribute `A2` will be granted role `r` (and thus the ability to read) also in `/a/1`[4]. In order to avoid this kind of problems, we limit role visibility only to the subspaces of the defining space, and require exclusivity across spaces which are directly or indirectly related as parent-child in the hierarchy (i.e., if a role is already defined in a space, then ancestor and descending spaces cannot define the same role later, and in this case an error must be raised).

The TAM policy is defined using the following concepts:

TrustFilteringRule
A trust filtering rule defines what attributes from a given provider can be accepted. Specific subconcepts of TrustFilteringRule define concrete rules.

```
TrustFilteringRule hasAttributeProvider <1> AttributeProvider
```

PreestablishedTrustRule subclass of TrustFilteringRule
This class represents a fixed trust rule defined by the space authority. Such a rule says that all attributes from a given provider are accepted or that only some attributes with a specific name or name–value pair (specified by the attribute constraints, see later) are allowed from this provider.

```
PreestablishedTrustRule
          hasAttributeConstraint <0..*> AttributeConstraint
```

AttributeProvider
Any attribute provider is an instance of this class.

Role
This class represents a role that can have access rights. There are predefined roles (instances of Role) *admin*, *banned* and *everyone*.

RoleMapping
This class represents a mapping from a set of authentication attributes and/or their values to a set of roles. The attribute constraints are taken in conjunction, i.e. all the constraints of a role mapping must be satisfied for the client to be assigned to the given roles. Disjunction (a logical *or* relationship) can be achieved simply with multiple role mappings. If no attribute constraint is present, the mapping applies to all identities, i.e. everybody gets the indicated roles. This is used to create the special role *everyone*.

```
RoleMapping hasAttributeConstraint <0..*> AttributeConstraint
RoleMapping hasRole <1..*> Role
```

[4]Moreover, this happens also if AC policy of space `/a` has actually no effect, because a FirstApplicable combinig rule (see chapter 4) is specified.

AttributeConstraint

This class represents a single constraint on an attribute value; either it just specifies that an attribute is present (if there is no value to the property hasAttributeValue), or that an attribute is present and it has the specified value, as determined by literal string equality.

```
AttributeConstraint hasAttributeName <1> string
AttributeConstraint hasAttributeValue <0..1> string
```

Unlike AC rules, attribute mapping rules are not ordered, and evaluation is never based on "first match": all rules are evaluated, thus the result will be a set of roles. When designing a triplespace policy, particular care is necessary about the interplay of roles; especially about the ordering of AC policy rules that may deny something to some role and permit it to another role. In some applications, it is desirable to have strict roles, i.e. allowing clients to play only one role at a time. This can be achieved if the AP is part of the application domain, and asks the client to choose a "role" and refuses to certify combination of attributes distinctive of different roles in the same assertion. In this case, if the Security Manager does not accept multiple assertions from the same client in the same "session"[5], the client can play different roles only in different sessions (i.e., in different requests). The possibility to refuse multiple assertions in the same session (i.e., to have multiple assertions in the same security context) may be governed by the TAM policy.

It is likely than in many cases subspaces will not need a different TAM policy with respect to their parent space. In the hierarchy of spaces, the trust filtering rules and role mappings of a space are also valid in its subspaces; therefore the subspaces need not repeat the same role mappings. This improves manageability, but in complex setups the designer must mind the role propagation from parent spaces.

Rationale

Access control is the "core" of the Triple Space policy, while TAM can be seen as an additional feature that can be plugged in if there is not a complete a-priori knowledge about clients[6]. Therefore, our goal is to let the policy designer focus on AC policy alone, and then add TAM policy in a quite straightforward way. The TAM process has been designed in order to be as simple as possible and to not change AC policy behaviour.

There is no closure assumption in trust filtering rules: the fact that an Attribute Provider is not explicitly trusted does not necessarily imply that it is untrusted, since a subspace TAM policy may declare it as trusted (only within the subspace). On the other hand, trust filtering rules can be only positive, i.e. it is not possible to declare a provider as untrusted. Adding negative trust rules would significantly complicate the model, requiring something like the rule/policy combining algorithms of the AC policy and making the whole TAM + AC process difficult to understand (and thus the TAM + AC policy difficult to design). Moreover, negative trust rules may not be

[5]If the client sends multiple assertions, the Security Manager should refuse authentication and send an error.

[6]If the set of clients is a-priori known, and everything lies in a closed and controlled domain, it is possible to give each client an assertion that directly specifies its role and use it for AC decision, without the need for TAM.

very effective, since the set of Attribute Providers is not known a-priori; therefore one could simply start a new Attribute Provider in order to bypass all negative trust rules.

The possibility for every space to have its TAM policy could potentially open a problem, because a subspace can declare a new AP as trusted, and in this way "bypass" the parent space AC policy. Nevertheless, this may be possible also without TAM functionality, as shown by the following example. In a system without TAM (i.e., where clients and roles that can be played by each client are defined in the system) the space authority must design an AC policy. If the AC policy of space `a` does not say anything about someone with role `r`, this does not mean that role `r` will be denied access to any subspace of `a`, because subspaces' policies may contain rules that permit access. If space `a` owner does not want this to happen he must explicitly specify a "catch-all" rule in the AC policy, acting as a default rule which denies access to any role which is not explicitly granted access[7]. If such a default rule is not present, space `a` owner allows subspace policies to grant access to other roles, "extending" the policy, even if the policy combining algorithm is LastApplicable and thus the parent's policy prevails (see chapter 4 for details on AC policy evaluation). A "default deny" rule in AC policy will still be effective when TAM policies are "plugged in", causing subspace policies not to be evaluated if LastApplicable is specified, thus avoiding the problem of a subspace policy adding a new trusted provider which should not be trusted.

3.3.2 TAM process flow

In this section we describe the trust and attribute mapping process flow, i.e. how the previously defined actors interact in order to perform the process.

As described in the authentication section, SM-Authn checks the validity of the assertion(s) and initializes the security context, thus after authentication we have a set of attributes (each one asserted by an AP), an operation request and a target space. The operation request is not relevant for TAM processing, since trust and attribute mapping do not depend on the operation type (they depend only on the AP, the attributes, and the target space). Theoretically, TAM processing comes before AC processing, since AC needs role mapping. Nevertheless, in order to perform TAM processing some information from AC is needed. Subspace hierarchy and policy combining algorithms define the order in which AC policies will be evaluated given the target space of the request (see chapter 4 for details), and in order to avoid unnecessary processing it is reasonable to follow the same order for TAM policy evaluation. Therefore, the request will be taken first by SM-AC, in order to decide which is the first space policy that must be evaluated. If TAM processing for this space has not been performed yet (i.e. mapped roles for this space are not present in the security context of this request), SM-TAM will do it (i.e. filter attributes according to trust rules and map them into AC roles) and update the context. This will be repeated if SM-AC needs to evaluate the AC policy of another space (e.g. a subspace) in order to take the AC decision. If the same clients submits another request using the cookie, the same security context will be used, so TAM processing will not be repeated[8].

[7]Space `a` owner may also explicitly deny access to each role which is not permitted access using appropriate deny rules. Nevertheless, this is quite impractical, and fails if a new role is added in the system at a later stage.

[8]If the assertion is the same and the policy has not been changed in the meantime, TAM policy evaluation on the same space gives the same results (since the input is unchanged).

3.4 Authentication and APIs

The TS-API has been defined in the work of WP3. Rather than add security information such as an operation or parameter for authentication directly in the API (and hence make the API dependant on a chosen security approach), we choose to use the given transport protocol as the means to pass security data such as an identity assertion to a Triple Space kernel. As the Triple Space is protocol independent, it will be a matter for each "grounding" of Triple Space communication to a given transport protocol to define how security information is concretely passed. As a general rule, a call to the Triple Space can be seen as a message with a header and an enveloped body (see Figure 2.2). The body contains the API operation and parameter values. The header will contain the security information. The Triple Space API implementation component in the kernel is able to access both header and body of the communication, and will first process the header (e.g. by passing its contents to the Security Manager) and then the body.

Hence, in the case of a first access to a kernel, with an identity assertion, the authentication is handled first, if a cookie is generated the operation is further handled as if the cookie had already been part of it and the client gets its response, with the cookie in the header. In the case of an out operation, which normally has no response, then only in this first case, is there a response with an empty body and the cookie in the header. Subsequent operations on the kernel pass the received cookie in the protocol header.

3.5 Related technologies: SAML

SAML (Security Assertion Markup Language) [10], developed by the Security Services Technical Committee of OASIS, is an XML-based framework for communicating user authentication, entitlement, and attribute information. SAML allows business entities to make assertions regarding the identity, attributes, and entitlements of a subject to other entities, such as a partner company or another enterprise application.

SAML is often used in Web single sign-on scenarios, in which an Identity Provider authenticates the user and issues an authentication assertion to a service provider, stating it has an "active" authenticated session with that client. This kind of scenario is however primarily targeted at human users interacting with services through a web browser, which is not the case of TripCom. This is the reason why we do not plan to directly reuse the SAML SSO profiles. Nevertheless, SAML assertions can be used to convey attributes, and SAML Assertion Query and Request Protocol can be used between the client and the Attribute Provider in order to retrieve the assertion.

4 Access control

Once authentication is completed, the client can send Triple Space API operations to the Triple Space. However, whether or not an operation sent by an authenticated client is actually carried out is determined by a further security stage: authorization, or "access control". In this chapter we introduce the access control model of Triple Space. Following an explanation of the main concepts, we outline the policy model and authorization flow. Finally, we model the external aspects of access control in the Management API, allowing the administrator users with security access to define and alter the access control policies.

4.1 Main concepts

The Triple Space system can be viewed at two levels: the physical layer of machines and networks, and the virtual layer of spaces and data. The physical layer is made up of kernels, which are organized under different administrative authority. The virtual layer is made up of spaces, which are hosted across kernels. The following two terms, defined in Section 2.2, are important for access control:

- The *kernel authority* defines which clients may manage spaces hosted by a kernel under its authority. The virtual layer is made up of spaces, which contain data. A space can be created at any kernel by a client who has the appropriate rights from the kernel authority;
- The *space authority* is the creator of a space and generally any data placed within their space is considered to be owned by them. Space authorities can define the security policies on the spaces administrated by them.

To control access, an authority may define an AC Policy, as defined below:

- *Access Control Policy* is a set of rules written in an agreed format which defines which client roles are allowed the execution of a particular operation. A rules processor which understands the policy language is able to reason if a client with a particular set of roles is permitted to carry out a particular operation or not. AC policies are associated to kernels or spaces.

Other important concepts such as the client, access cookie and the Security Manager were introduced in Section 3.1.

4.2 Internal authorization flow

Authenticated clients have an access cookie which is passed with every operation they send to a kernel and which associates them with some identity information. The advantage of a cookie in every operation is that it preserves the asynchrony and statelessness of the communication. The security manager's task is to reason, using a set of access control policies, whether the client with its given identity can be authorized to carry out the operation at the kernel.

In Triple Space, we can see this as the security manager being notified of incoming new operation requests for the kernel. Before any further processing of that operation

takes place, it must be determined if the requester has the authorization to perform that operation in the kernel. So operations being added in the system bus trigger notification of the security manager, with a structure such as (operation o, space s, content c, client-attributes ca). [1]

A data operation will be processed by checking if the given operation o is permitted to a client with client attributes ca by the access control policies for space s. A space operation will be processed by checking if the given operation o is permitted to a client with client attributes ca by the access control policies for the present kernel. Access control policies are stored in security containers on the kernels, with space policies being associated to the respective space and the kernel policy being associated to the root space of the kernel (which does not directly contain any client data, but is the root of all spaces managed by this kernel).

The tuple emitted back into the space by the security manager would contain an "applicative" part and a "security" part. The security part, generated by the security manager and appended to the original tuple, would contain the results from the policy decision point. We could imagine this to be a simple, boolean TRUE (authorized) or FALSE (not authorized), while restrictive forms of the TRUE authorization could also be imagined, e.g. operation authorized on the space but not on any of its child spaces. This would need representation by a more expressive datatype, e.g. constant integer values such as is done in Java.

A failed (not authorized) operation would be removed from the system bus, either by being taken by the API Implementation (if the operation was local to that kernel) or being notified back to an originating kernel (if the operation was remote to that kernel). We do not consider an error message to the client, however once all kernels that have been asked respond, either through a response or a failure message, the operation should be ended at the local kernel (i.e. so that a read does not block endlessly when all kernels fail to authorize the operation). An authorized operation should be taken by other kernel components for further processing.

The policy decision point must be able to reason on the basis of acquiring a set of policies which apply to the space or kernel on which the operation shall be applied. The set of policies it acquires should be able to combine a value from the set of operations supported in the system on the space or kernel (as are defined in the Triple Space API) with a value from the set of attributes of clients known by the system (we consider roles as the primary attribute to be tested) and determine either a boolean TRUE or FALSE (authorized or denied).

4.3 Policy model

This section describes the access control part of the policy ontology for Triple Spaces, an extension of the TS ontology defined in WP2. The domain and scope of the ontology are similar to those of the TS ontology. The whole ontology is listed in Appendix A.

This ontology is based on an identified subset of XACML [9], shown in Figure 4.1. The figure shows the XACML model, with greyed-out parts that the ontology does not explicitly capture. In short, every space in a TS kernel contains an implied policy set that aggregates the policy of this space with the policies of its subspaces. Every

[1] actual tuples describing operations may carry more information such as active timeout but we consider the fields given here as the relevant ones for the operation of the security manager

policy is implicitly targeted at the space in which it resides (Resource target). Policies may contain rules that govern the actual access rights; every rule is targeted at some subject(s) and some action(s); rule targets do not use Resource specification. Finally, a policy can specify the rule combining algorithm (Deny-rule-overrides, Permit-rule-overrides, First-applicable-rule), and the policy set can similarly specify the policy combining algorithm.

This section first describes the policy ontology and then recommends a default policy for the root space of a kernel (the default kernel policy).

4.3.1 Classes of the policy ontology

This section shows the classes and the properties that apply to them. The security ontology does not contain inverse properties as in RDF properties can be navigated in both ways.

Concepts from the TS ontology from WP2 are referred to with the prefix `ts:`. Properties are noted as `DomainClass propertyName <cardinality> RangeClass`.

PolicySet

This class represents the collection of all the policies assigned to a ts:Space and its subspaces. The containment relationships between the policy set, the policy of the owning space and the policy sets of the subspaces of the owning space is implied by the subspace hierarchy. This class is used for policy evaluation.

```
PolicySet hasPolicySet <0..*> PolicySet
    // implied by the space hierarchy
PolicySet hasPolicy <1> Policy
    // implied by the space ownership
PolicySet hasPolicyCombiningAlgorithm <1> PolicyCombiningAlgorithm
    // default is First-applicable
PolicySet hasTarget <1> Target
    // this is implied, the target only specifies the ts:Space
```

For the purpose of ordered policy combining algorihtms, the space policy is ordered **after** the subspace policy sets. The order of the subspace policy sets is insignificant as their targets do not overlap so not more than one of them can apply to a single policy decision.

PolicyCombiningAlgorithm

This class represents policy combining algorithms. We adopt three XACML policy combining algorithms: *Deny-overrides*, *Permit-overrides*, *First-applicable*; and we add a new *Last-applicable*.

Last-applicable is added to allow constructing policies where subspace policy is overridden by the policy of the parent space.

Policy

This class defines the policy of the ts:Space that owns the SecuritySpace containing this Policy. The policy has an implied resource target which is a ts:Space.

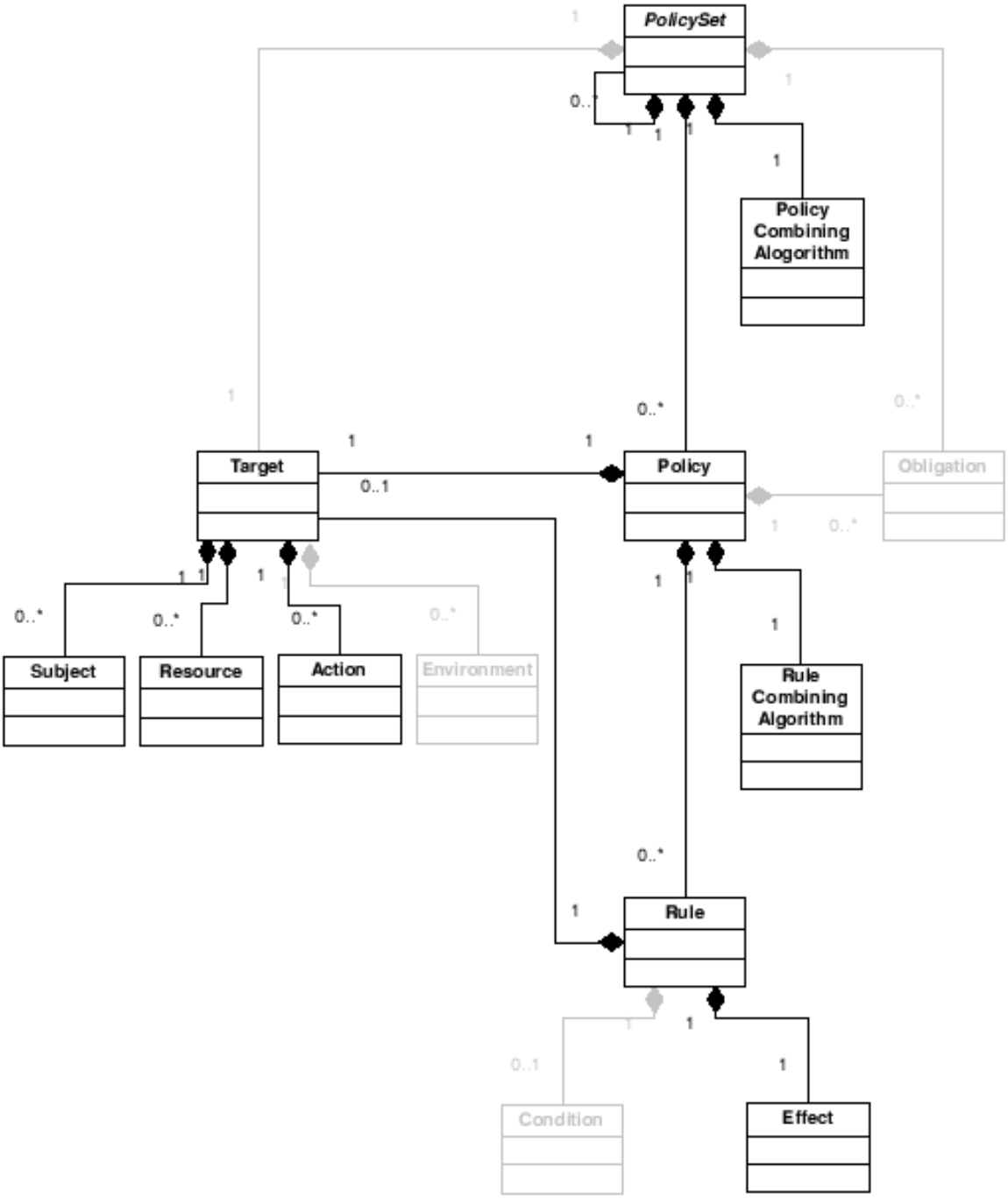

Figure 4.1: XACML subset in the Triple Space policy ontology

```
Policy hasRules <1> RuleSeq
    // Rules are in an ordered rdf:Seq container
Policy hasRuleCombiningAlgorithm <1> RuleCombiningAlgorithm
    // default is First-applicable
Policy hasTarget <1> Target
    // this is implied, the target only specifies the ts:Space
```

Rule

This class represents a single permission granting or denying some access rights to a subject in a space.

```
Rule hasTarget <0..1> Target
Rule hasEffect <1> Effect
```

Effect

This class groups the two possible effects: *Permit* and *Deny*.

RuleCombiningAlgorithm

This class represents rule cimbining algorithms. We adopt three XACML rule combining algorithms: *Deny-rule-overrides*, *Permit-rule-overrides*, *First-applicable-rule*.

Target

A policy target points to exactly one ts:Space and no actions or roles. A rule target represents a set of subjects and a set of actions (at least one of each), and no ts:Space.

```
Target hasSubject <0..*> Role
Target hasAction <0..*> Action
   // hasSubject and hasAction only on rule target
Target hasResource <0..1> ts:Space
   // hasResource only on policy target
```

A target with more than one role applies to any of the roles, not only those clients who play all the specified roles. In other words, if a client plays any of the roles specified in the target, the target applies to the client. Similarly with the actions, the target applies to any of the actions specified; at a single time only one action can be requested, so if it is among the actions of the target, the target applies.

We currently don't support rules with data conditions, this may be added later if needed for finer-grained security settings; e.g. the following scenario: client X can write requests and only read the responses to its requests: X writes messsage A, Y can read it and create message B in response, X can read message B but no other messages.

Action

This class represents actions that can be granted or denied to subjects in triplespaces. We define the following actions: *Read*, *In*, *Out*, *Subscribe*, *Create*.

4.3.2 Recommended root space policy

This section describes the recommended root space policy (kernel policy), explaining the reasons behind the simple rules. This policy can also be built-in, and some parts of it may be fixed in the security manager, so that a Triple Space kernel cannot be put into a state where it would deny all requests.

Role mapping a known identity onto the role "admin" and any banned identities onto the role "banned", to be used by the following rules; plus any identity onto the role "everyone", which may be used by application policies.

The **policy combining algorithm** for the root policy set should be *Last-applicable*, so that the following rules always apply and the admin is always allowed to read everything and to manipulate spaces.

The **rule combining algorithm** of the root policy should be *First-applicable*, so the admin cannot be banned by accident.

First rule: role "admin" is permitted any action.

Second rule: role "banned" is denied any action.

This policy allows the kernel administrator to read all data, to manipulate spaces (create and remove them, change their security settings), and to ban some identities (based on their attribute values) from ever using this particular kernel.

4.4 Authorization flow

This section describes all the steps that the security manager goes through in order to authorize or deny a particular request.

A Triple Space kernel contains any number of spaces, grouped together in a single *root space* for the purposes of security management. The policy of any given space also applies to its subspaces, therefore the relevant policies for any request need to be combined within the subspace hierarchy, as described in Section 4.4.1. Since `read` operations combine the data from all the subspaces of the target space, they need to be controlled by the policies of the subspaces as well, as described in Section 4.4.2.

Section 4.4.3 defines the process that guides the evaluation of a particular single policy. Section 4.4.4 contains an example of a policy setting and evaluations of the policies for various queries, to illustrate the evaluation rules. Finally, Section 4.4.5 sketches some potential optimizations, should policy evaluation prove to be a performance bottleneck.

4.4.1 Combining policies in subspace hierarchy

Since data in a triplespace also logically contains the data of its subspaces, it is necessary that a space can control access to its subspaces as well.

The root policy set of a kernel (or of its root space) logically contains all the policies or all the spaces on this kernel. The target of this policy set, as well as the target of the root space, is the whole root space, i.e. including all the subspaces. Therefore we can say that the authorization decision for any request is the result of evaluating the root space policy set. The Triple Space policy enforcement point is **deny-biased**, i.e. if no rule in any of the applicable policies matches the request, the result of the policy set evaluation would be *NotApplicable*, and the authorization decision will be *Deny*.

The evaluation of a policy set is adopted from XACML (cf. Section 7 in the XACML specification [9]). If the policy set target does not match the request, the result of the policy set evaluation is *NotApplicable*. If the policy set target does match, the result depends on the evaluation of the policies and policy sets contained in this policy set, combined using the specified policy combining algorithm.

The policy combining algorithms are *Deny-overrides*, *Permit-overrides*, *First-applicable*, and *Last-applicable*. Within a space policy set, the subspace policies are ordered **before** the policy of the space itself. The default is *First-applicable*, which means that subspaces **override** the policy of the space. Policy combining algorithms are specified on a per-space basis. The following points define the four policy combining algorithms:

- *First-applicable:* the policies and policy sets are evaluated in order[2]; the first result other than *NotApplicable* is also the result of the evaluation of the policy set.
- *Last-applicable:* the policies and policy sets are evaluated in **reverse** order; the first result other than *NotApplicable* is also the result of the evaluation of the policy set.
- *Permit-overrides:* if any of the contained policies and policy sets evaluates to *Permit*, the policy set evaluates to *Permit*.
- *Deny-overrides:* if any of the contained policies and policy sets evaluates to *Deny*, the policy set evaluates to *Deny*.

The latter two, Permit-overrides and Deny-overrides, are provided here for completeness as they are in XACML, but since the policy set structure reflects the hierarchy, it is improbable that these two policy combining algorithms would be commonly used. Instead, most of the Triple Space deployments are expected to First-applicable and Last-applicable as the policy combination algorithms, depending on whether the subspaces should be able to override the parent spaces (super-spaces) in the policy for this scope.

Figure 4.2 contains a pseudo-code algorithm showing how a full authorization goes through the policy sets. For the evaluation of the individual space policies, see further Section 4.4.3.

4.4.2 Note on access control for `read` operations

Operations that read data from a triplespace generally use all the data from the space itself and from its subspaces. It is possible that some role has read access to a space, but not to some specific subspace. Therefore, when the data from the space and its subspaces is being combined, all the subspaces must evaluate their policies and in case the read is denied, the data from the denying subspace will not be available to the client query.

Note that if a space does not grant read access to a given role, a `read` operation will be **denied regardless** of whether the role has read access to some of the subspaces.

By extension, if a client has read access to space A, does not have read access to its subspace A/1, and again has read access to its subspace A/1/i, a read on A will not

[2]To remind, in the policy set of a space, the space policy is ordered **after** the policy sets of the direct subspaces of the given space.

Figure 4.2: Policy set evaluation pseudo-code algorithm

```
authorization(Space target, Action action, AuthAttr[] attrs)
    Effect effect = policySetDecision(target, action, attrs, rootSpace, emptySet)
    if (effect is NotApplicable) then
        return Deny
    else
        return effect

policySetDecision(Space target, Action action, AuthAttr[] attrs,
                  Space currentSpace, Role[] previousRoles)
    SecuritySpace sspace = currentSpace->securitySpace
    PolicySet ps = sspace->policySet
    if (ps->target->resource doesNotMatch target) then
        return NotApplicable
        // doesNotMatch means the target space of the policy set is not
        // the target space of the operation or any of its superspaces

    RoleMapping[] rm = sspace->roleMappings
    Role[] userRoles = previousRoles + mapRoles(attrs, rm)
    Effect effect = NotApplicable

    // note that sps<-policySet below means reverse navigation from a policy set to the space
    // that owns it; semantic languages and specifically RDF allow such reverse navigation

    if (ps->policyCombiningAlgorithm is First-applicable) then
        for each PolicySet sps in ps->subspacePolicySets
            effect = policySetDecision(target, action, attrs, sps<-policySet, userRoles)
            if (effect is not NotApplicable) then
                return effect
        effect = policyDecision(action, ps->policy, userRoles)
        return effect

    if (ps->policyCombiningAlgorithm is Last-applicable) then
        effect = policyDecision(action, ps->policy, userRoles)
        if (effect is not NotApplicable) then
            return effect
        for each PolicySet sps in reversed ps->subspacePolicySets
            effect = policySetDecision(target, action, attrs, sps<-policySet, userRoles)
            if (effect is not NotApplicable) then
                return effect
        return NotApplicable

    if (ps->policyCombiningAlgorithm is Permit-overrides) then
        Effect effectIfNoPermit = NotApplicable
        for each PolicySet sps in ps->subspacePolicySets
            effect = policySetDecision(target, action, attrs, sps<-policySet, userRoles)
            if (effect is Permit) then
                return Permit
            if (effect is Deny) then
                effectIfNoPermit = Deny
        effect = policyDecision(action, ps->policy, userRoles)
        if (effect is Permit) then
            return Permit
        if (effect is Deny) then
            effectIfNoPermit = Deny
        return effectIfNoPermit

    if (ps->policyCombiningAlgorithm is Deny-overrides) then
        Effect effectIfNoDeny = NotApplicable
        for each PolicySet sps in ps->subspacePolicySets
            effect = policySetDecision(target, action, attrs, sps<-policySet, userRoles)
            if (effect is Deny) then
                return Deny
            if (effect is Permit) then
                effectIfNoDeny = Permit
        effect = policyDecision(action, ps->policy, userRoles)
        if (effect is Deny) then
            return Deny
        if (effect is Permit) then
            effectIfNoDeny = Permit
        return effectIfNoDeny
```

Figure 4.3: Access control for read operations pseudo-code algorithm

```
read(Space target, AuthAttr[] attrs, Query q)
    Effect ac = authorization(target, Read, attrs)
    if (ac is Deny)
        throw exception OperationDenied
    else
        Data data = readRecursive(target, attrs)
        return query(q, data)

readRecursive(Space target, AuthAttr[] attrs)
    Data results = lowLevelRead(target)
    for each Space subspace in target->subspaces
        Effect ac = authorization(subspace, Read, attrs)
        if (ac is not Deny)
            results += readRecursive(subspace, attrs)
    return results

lowLevelRead(Space target)
    // returns the data from the target space, but not its subspaces
```

Figure 4.4: Policy evaluation pseudo-code algorithm

```
policyDecision(Action action, Policy policy, Role[] userRoles)
    if (policy->ruleCombiningAlgorithm is First-applicable-rule) then
        for each Rule rule in policy->rules
            effect = ruleDecision(rule, action, userRoles)
            if (effect is not NotApplicable) then
                return effect
        return NotApplicable

    if (policy->ruleCombiningAlgorithm is Permit-rule-overrides) then
        Effect effectIfNoPermit = NotApplicable
        for each Rule rule in policy->rules
            effect = ruleDecision(rule, action, userRoles)
            if (effect is Permit) then
                return Permit
            if (effect is Deny) then
                effectIfNoPermit = Deny
        return effectIfNoPermit

    if (policy->ruleCombiningAlgorithm is Deny-rule-overrides) then
        Effect effectIfNoDeny = NotApplicable
        for each Rule rule in policy->rules
            effect = ruleDecision(rule, action, userRoles)
            if (effect is Deny) then
                return Deny
            if (effect is Permit) then
                effectIfNoDeny = Permit
        return effectIfNoDeny

ruleDecision(Rule rule, Action action, Role[] userRoles)
    if ((rule->target->subjects is empty
         or intersection(rule->target->subjects, userRoles) is not empty)
        and
        (rule->target->actions is empty
         or rule->target->actions contains action))
    then
        return rule->effect
    else
        return NotApplicable
```

contain any data from A/1, and especially it will not contain any data from A/1/i. Nevertheless, the client may read A/1/i directly (as long as the A/1 policy set has the default First-applicable policy combining algorithm to allow the subspace policy of A/1/i to override it).

Figure 4.3 contains a pseudo-code algorithm that shows a high-level view of how read operations work in terms of authorization.

4.4.3 Policy evaluation

A single policy is made up of rules that are combined according to the rule combining algorithm. The available rule combining algorithms are *Deny-rule-overrides*, *Permit-rule-overrides*, and *First-applicable-rule*. The order of the rules depends on the designer of the Triple Space application.

Each rule has a *target* which consists of two parts: a set of roles to which the rule applies (empty set means any and all possible roles), and a set of actions which the rule controls (again, empty set means any actions). The target of a rule *matches* if and only if the following conditions are fulfilled:

1. either the set of roles in the rule target is empty, or any of the client's roles is mentioned among the rule target roles;

2. either the set of actions in the rule target is empty, or the client request action is mentioned among the rule target actions.

Along with a target, each rule has an *effect*, which is either *Permit* or *Deny*. If the rule target matches, the effect is the result of the evaluation of the rule.

If no rule target in a policy matches the current request, the result of evaluating the policy is *NotApplicable*.

Policies are captured in terms of what is permitted or denied to various *roles*. Authentication, described in previous chapter, assures that every request is accompanied with some information about the identity of the requester which maps into a set of roles. Any client may play multiple roles. A security design of a particular Triple Space application needs to take particular care about the interplay of roles; especially about the ordering of policy rules that may deny something to some role and permit it to another role. It needs to be considered whether the policy should permit or deny the action to a client playing both roles. This is going to be determined by the rule combining algorithm and by the order of the rules.

Figure 4.4 shows a pseudo-code algorithm for evaluating a single policy.

4.4.4 Policy evaluation examples

Figure 4.5 shows a sample setting of spaces "/a" and "/b" in a Triple Space kernel. Space "/a" has subspace "/a/1", space "/b" has subspaces "/b/1" and "/b/2", and subspace "/b/1" has a subsubspace "/b/1/i". In this setting, the example requests and the policy evaluation results are shown in Table 4.1.

Notably, due to the policy combining algorithms used in the example, the order of evaluating policies for access to space /b/1/i is the following: first, the root policy / is checked, and then the space policies in reverse order, i.e. first /b/1/i, then /b/1 and then /b. This shows how the subspace policies in this case are allowed to override the parent space policies, with the exception of the root policy.

Figure 4.5: Example policy evaluation setting

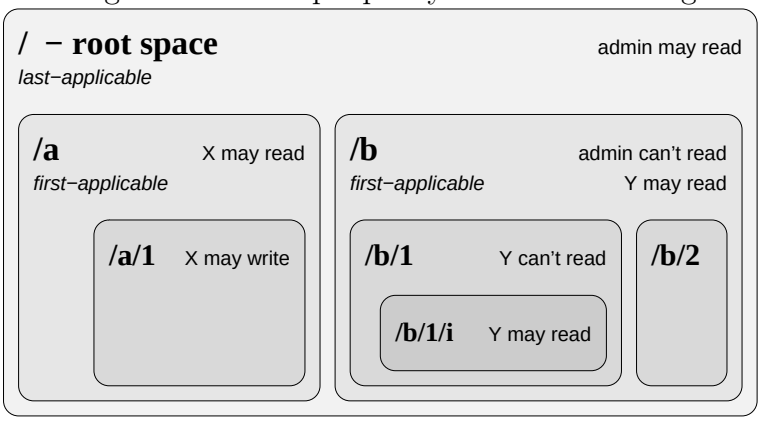

Table 4.1: Example requests and results for policies in Figure 4.5

Request	**Result**
X reads /	Denied because no policy applies to X reading /
X reads /a	Return contents of /a, /a/1
X writes in /a	Denied because no policy applies
X writes in /a/1	Allowed
Y reads /b	Return only contents of /b, /b/2
Y reads /b/1	Denied by /b/1 policy
Y reads /b/1/i	Return contents of /b/1/i
admin reads /b	Return contents of /b and subspaces, allowed by root space policy, even though /b would deny that

4.4.5 Notes on potential optimizations

Abstractly, every request to some space triggers the evaluation of all the parent spaces' policies, and read requests further trigger the full policy evaluation for each subspace. Such full evaluation may become a performance issue in deep space hierarchies or in spaces with complex policies. This section mentions some optimization strategies that may make policy evaluation more efficient.

First, policies can be compiled into an internal representation, instead of accessing the security space and evaluating the RDF triples representing the policies. Of course any change to space policies must trigger a recompilation of the internal representation.

Every evaluation starts with the root space policy set and dives into the appropriate subspace tree. The internal representation of the policy set tree should be optimized for the selection of the appropriate subtrees, without having to evaluate up to all the space designator targets.

The nodes in the space policy tree can cache common policy decisions. Assuming a limited number of roles, the cache memory footprint should be very manageable.

Finally, especially for read operations, the abstract process evaluates the policies, starting from the root, as many times as many there are subspaces of the space read by the operation. This can be optimized away by a close binding between the component that combines data from subspaces and the security manager component; the component that dives into the subspaces will at the same time dive into the policy tree and so all evaluations will only be done once.

4.5 Security policies and APIs

The Management API will provide operations for both the kernel authority — the administrator of a Triple Space instance on a server — and the space authority — the creator/owner of a space created within Triple Space. Hence we will also need to ensure that they can be authenticated and trusted. They will require an access cookie from the authentication stage which associates them with the identity of kernel authority or data authority in order to be authorized to use the Management API operations (in the access control policies, they have the role of "admin").

Kernel and space authorities will provide the security manager with the access policies they wish to enforce on a kernel or space (respectively) that they own. These could be expressed as RDF named graphs and hence added to/removed from the space using the standard Triple Space API operations applied to a special "security" space. However this has the danger of confusing the regular Triple Space interaction with security-related interaction that requires a particularly careful handling. Hence we propose rather a set of management operations named m-out, m-rd, m-in and m-update which emit, read, remove and update a RDF graph of security data respectively. Table 4.2 provides detail to these operations.

Operation	Returns	Description
m-out(Graph g, Space s)	boolean	Adds the access control policies in graph g to the access control policies for space s. If space s is null, the access control policies apply to the kernel to which this operation was passed. The operation returns true once the access control policies have been added at the kernel. False can indicate that the policies were invalid, e.g. not conform to the policy language supported by the kernel or that the space s is not managed by that kernel.
m-rd(Query q, Space s)	RDF Graph	Applies the query q to the access policies applying to space s and returns the result as a RDF graph. The query can be any supported at the kernel (as defined in WP3). We can expect SPARQL to be supported. As policies will be self-contained RDF graphs, the answer is always made up of complete policies. In other words, if part of a policy matches the query, the entire policy is returned. If space s is null, the operation applies to the kernel access policies.
m-in(Query q, Space s)	RDF Graph	As m-rd but deletes the policies returned in the RDF Graph answer to the query.
m-update(Query q, Graph g, Space s)	RDF Graph	As m-in, but every policy deleted as a result of the operation is replaced by a new policy created by a replacing of the triples matched in Query q in the deleted policy with the triples contained in Graph g. If space s is null, this applies to the kernel access policies.

Table 4.2: Management API operations for security

A final case regards changes in kernel access policies affecting spaces. We will assume changes in policies on the kernel only affect operations after that change, i.e. if a space exists within a certain kernel because it was created/added at a time when that was permitted, then that space will continue to exist within the kernel even if the policies would no longer permit its creation. However, we assume kernel authorities are

the final arbitrators of what exists in their part of the system, overriding the rights of space authorities if necessary, and hence always have authorization to destroy(space) on any kernel under their authority.

5 Logging and auditing

5.1 Scoping Logging and Auditing

Logging and Auditing are essential features of systems that provide services under any sort of service level agreements, contractual obligations or legal requirements.

The typical Triple Space application operates under such constraints as described in the use cases WP8a and WP8b. We must therefore scope the notion of *logging* and of *auditing*.

We understand under *logging* a general mechanism that records specific, configurable aspects of the activity of a system in a time sequential order. Usually a system will offer a large variety of logging mechanisms, like:

- *Activity logs*: Records the activity of a system in form of requested action, originator, request parameters and result. Activity logs are usually used to document the interaction of a system with the external world.
- *Security logs*: Records all attempts to access a system; successful as well as failed attempts.
- *Error logs*: Records all exceptional conditions in correlation to the requested activity.
- *Debug logs*: Detailed technical log used to resolve technical problems during operation.
- etc.

Auditing is a more specific capability - using log based techniques. *Auditing* provides the capability to document and prove that a system operates in compliance with the regulatory, legal and contractual obligations imposed on it. Thus an *audit trail* will be used by auditors to check the compliance.

ISACA[1] defines audit as "The process of generating, recording and reviewing a chronological record of system events to ascertain their accuracy". This is in contrast to logging which is defined as "To record details of information or events in an organized record-keeping system, usually sequenced in the order they occurred".

The EPS use case for example, requires strict compliance with privacy regulations. Thus the audit must provide proof that no data have been improperly disclosed. A simple log documenting the access to the system might not be sufficient. The definition of the specific audit is therefore part of requirement definition of the system. In the content management use case, an auditor might request proof that no copyright infringement took place or that the access to the auctions operated under equal conditions for all participants. Such audit trails might not only include security related information but also timing and response time information.

Audit requires therefore the understanding of the systems intent and the definition of control mechanisms allowing to prove that the system behaved according to agreed policies. Such mechanisms rely on rules, event correlation and pattern recognition.

It is beyond the scope of this deliverable to define specific auditing mechanisms since they are very application domain dependent. However, we will illustrate how space based systems offer basic support for the implementation of such mechanisms.

[1]The Information Systems Audit and Control Association — `http://www.isaca.org/`

5.2 XVSM based mechanisms for logging and auditing

From an implementation point of view, logging is considered a cross-cutting concern. This means, that application code should not be concerned with logging and that logging — and therefore also auditing — should be handled independently from the application code.

The XVSM middleware supports the implementation of cross cutting concerns through *aspects* mechanisms.

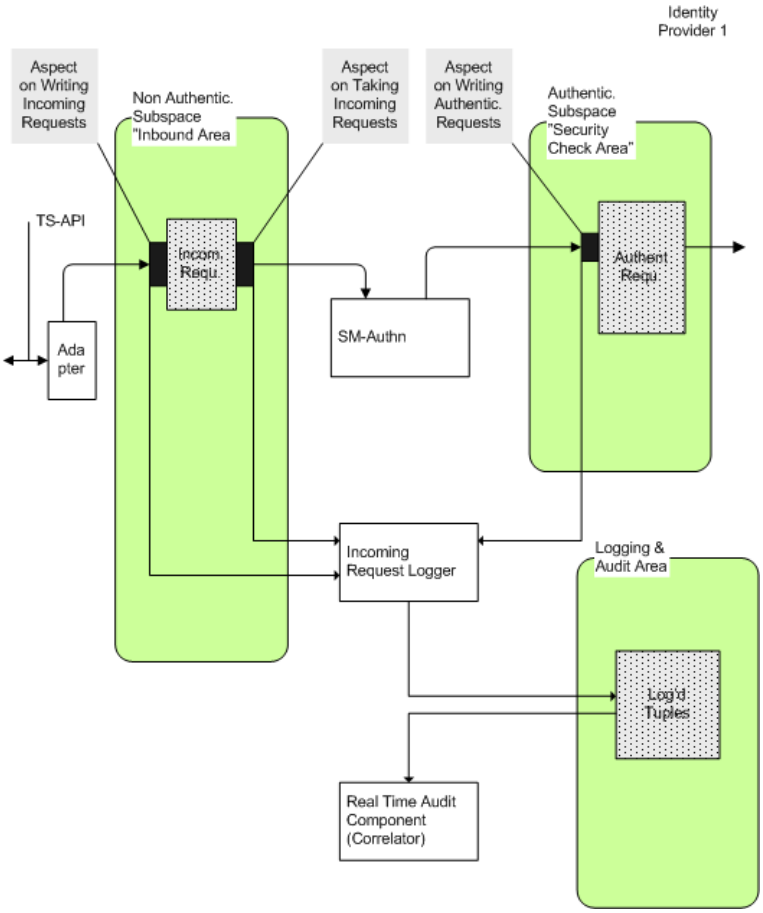

Figure 5.1: Example of aspects in XVSM

Aspects are functions that are executed before or after a specific space primitive (*in*, *out*, *rd*) executes in a specific subspace. They are similar to triggers in databases. Aspects can be dynamically bound at runtime to a subspace.

In the example above, an aspect is bound to the non-authenticated subspace on any *out* action and a second one on any *rd* action (denoted *writing* and *taking* in the figure). The function defined in the aspect inserts a copy of the tuples to a special *logging and auditing* subspace. These two aspects are therefore able to report on each incoming requests and on which of the TS-Authn processes processed the request. A third aspect in the example is bound to *Security Check Area* and monitors the inflow of authenticated requests.

Since all these records are stored in the *logging and auditing* subspace, we are basically able to correlate these events and compute a simple security audit. Such security audit could tell us, which of the TS-Authn processes were responsible for specific authentications.

The TS-Ontology described in Deliverable D2.2 [5] contains an *AccessLogEntry*. The *AccessLogEntry* is generated on each access to specific data. Such entries are produced by components in the *Internal Area*, for example by components bound to aspects on the respective subspace.

The logging model used by the components of the security manager is able to extend the concept of the *AccessLogEntry*: The log of a security manager relates to the result of the security process.

The *AccessLogEntry* relates to some known information that exists in the space, by documenting the individual accesses to the data. In contrast, the log of the security manager relates to the outcome of the security process based on the request itself. It might not relate to any known data - for example when the SM-Authn component is unable to resolve the identity of the request's originator. A *SecurityLogEntry* will therefore point to the involved requests and will document the outcome of the security process in the permit as well as in the case of deny.

6 Trust and reputation

Several independent authorities (kernels, spaces, clients and attribute providers) interact in the context of TripCom and trust plays a relevant role in their relationships. In this chapter, we briefly introduce the relevant trust and reputation concepts (section 6.1) and then analyse the different trust relationships existing between these authorities and whether and how we can support their establishment at the infrastructural level (section 6.2). Finally, we describe a solution for supporting the acquisition of reputation information (section 6.3) which can be exploited by clients, together with private local information, for assessing trust in counter-parties through the use of computational trust models.

6.1 Main concepts

We can informally define *trust* as an estimation of the particular level of the subjective probability with which an agent assesses that another agent will assume a certain behaviour at a given time and in a given context (adapted from [6]). We can identify three key aspects in this definition:

- trust is *subjective*: agent A may trust agent B while agent C may not;
- trust *depends on time*: agent A may trust agent B at time t_1 and may not trust it any more at time t_2;
- trust *is related to a context*: agent A may trust agent B with reference to a particular scope and may not trust it with reference to another scope (e.g. A trusts B for a certain kind of service and not for another one).

Reputation is a related but different concept, and we can informally define it as the shared perception of a community about the trustworthiness of a party in a particular context, based on its past behaviour. Just like trust, reputation depends on time and on the context. Unlike trust, reputation represents a shared, more "global" perception (it can be seen as a sort of "average opinion" inside a community), thus is not subjective. A *reputation management system* is a system that collects subjects' opinions based on past interactions with the object and condenses them in a reputation value, for instance by averaging the numerical judgements expressed by users in their opinions.

Reputation is important for trust, since it can be a primary *trust source*. Trust sources are elements that a subject takes into account in order to compute a trust value for a given object. Trust sources can be either *direct* (i.e. own opinions derived from own experience) or *indirect* (i.e. involving the opinion or judgement of a third party). Examples of indirect trust sources are:

- *evidences*, i.e. proofs that an entity satisfies certain requirements (e.g. some kind of certification issued by a known authority);
- *recommendations*, i.e. third parties' opinions collected by the subject (e.g. agent A tells agent B that agent C is "good", and agent A trusts agent B's judgement);
- *reputation*, i.e. the "average public opinion" about the object, as discussed above.

Other sources may involve *sociological information*, such as roles and relationships among individuals in a social network, and *prejudice*, consisting in the assignment of trust on the basis of specific trustee's characteristics or its membership to a particular (trusted) group.

A *trust management system* is a system that monitors trust sources and derives trust values related to objects. Since trust is a subjective property, trust management systems are related to subjects, each one adopting its own evaluation criteria and taking its own trust decisions. We can schematize the functionalities of a trust management system in three main activities:

1. *acquisition of information* relevant to trust (e.g. feedbacks, evidences);
2. *evaluation of trustworthiness* of a party through some sort of trust metric, computed on the basis of collected information.
3. *support of user decisions* (or, even, automation) depending on trust considerations.

6.2 Scoping trust and reputation in TripCom

Several trust relationships can be identified in the context of TripCom. Moving from an "infrastructural" level toward a more "applicative" one, in the following list and in Figure 6.1 we give a picture of the most important relationships, each one between a trustor actor (on the left) and a trusted entity (on the right).

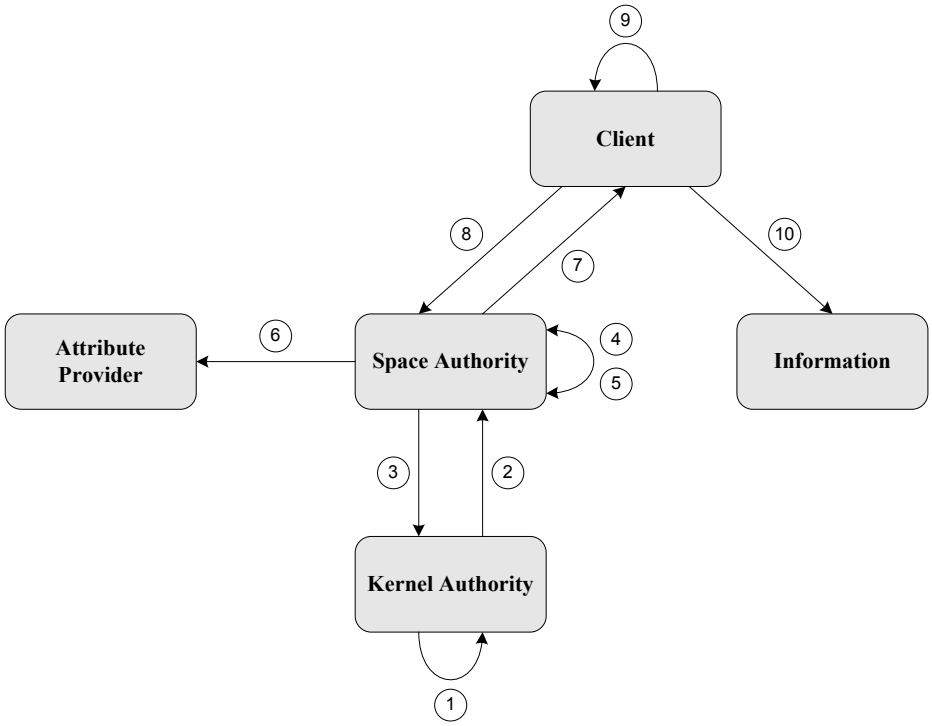

Figure 6.1: Trust relationships in TripCom

Kernel authority → Kernel authority (1). Although we focus here on the single-kernel scenario, in a distributed environment the elaboration of a client request

may span several kernels. Depending on the particular distribution strategy adopted, this process may require the direct interaction of kernels under different authorities, therefore requiring the existence of some sort of trust relationship between them.

Kernel authority → Space authority (2). When authorizing the creation of a triple space, the kernel authority have to trust the space authority to not abuse the resources provided. The establishment of this kind of trust, defined as *access trust* in the literature [7], is greatly influenced by the distribution strategy (e.g. whether a space spans multiple kernels) and will probably depend on several "out-of-band" factors (that is, external to the TripCom infrastructure), such as the adoption of regulations or agreements between involved parties which could provide for suitable guarantees or penalties for dishonest behaviour.

Space authority → Kernel authority (3). In a symmetric way, the space authority which creates a triple space on a kernel must trust the competence and the honesty of its kernel authority. This is particularly important since the kernel authority, similar to a hosting provider, physically controls all the stored data and can always access it using a low level interface (e.g. by reading and writing physical files). This kind of trust, called *provision trust* in the literature [7], will mainly depend on the same out-of-band mechanisms seen for the previous relationship (2).

Super-space authority → Sub-space authority (4). Different authorities may control triple spaces pertaining to a same hierarchy. When allowing the creation of a sub-space, a super-space authority must trust the sub-space authority; this decision is made explicit explicit through the AC policy, which identifies who is trusted enough to create a sub-space.

Sub-space authority → Super-space authority (5). Changing point of view, a sub-space authority must trust the super-space authority to act as agreed, e.g. to not abuse his dominant position by allowing unauthorized third parties to access the contents of the sub-space. In this case, the decision to trust a space authority will generally depend on the availability of information or on pre-existing agreements or relationships between parties, all of which represent out-of-band mechanisms not addressable by the TripCom infrastructure.

Space authority → Attribute provider (6). Client identification and role assignment are performed by the SM-TAM component by means of assertions issued by trusted attribute providers (AP). When accepting an assertion, the space authority is trusting the judgement of the AP regarding the certified characteristics of the client. Currently, this trust relationship is made explicit through the definition of trust filtering rules in the policy (see chapter 3).

Space authority → Client (7). From the perspective of a space authority, the access of a client to its space is a matter of access control, where the policy's rules define which clients are trusted for performing a particular operation.

Client → Space authority (8). A client accesses a space in order to read or publish data or to use a service based on it. Since these operations may expose the client to risks (e.g. the disclosure of sensible data), it is usually required

Table 6.1: Characterization of trust relationships

Rel.	Trustor	Trustee	Characteristics of trust
1	Kernel authority	Kernel authority	Depends on distribution strategy
2	Kernel authority	Space authority	Depends on distribution strategy Use of out-of-band factors
3	Space authority	Kernel authority	Use of out-of-band factors
4	Super-space authority	Sub-space authority	Related to access-control
5	Sub-space authority	Super-space authority	Use of out-of-band factors
6	Space authority	Attribute provider	Related to access-control
7	Space authority	Client	Related to access-control
8	Client	Space authority	Use of computational trust
9	Client	Client	Use of computational trust
10	Client	Information	Expressible in terms of 8 and 9

a trust relationship between the client and the authority controlling the space. The establishment of this relationship could take advantage from *computational trust* approaches, i.e. techniques that exploit available information, such as third parties' opinions, for deriving a quantitative trustworthiness estimate of counter-parties.

Client → Client (9). At the application level, interacting clients may have to trust each other in order to accomplish their goals. While the nature and relevance of this trust relationship depend on the particular application, generally speaking a computational trust approach could help the client in selecting and evaluating the trustworthiness of counter-parties.

Client → Information (10). Truthfulness of published information is of primary importance for clients. Trust in information quality and truthfulness can be traced back to the two trust relationships between the client and the author of the information (concerning contents) and between the client and the space authority which hosts the information (concerning integrity).

The relationships identified are summarized in Table 6.1, together with a classification of the nature of trust in the context of TripCom.

Although trust is a widespread aspect in the project, several relationships (2, 3, 5) can be classified as *out-of-band* and *out-of-scope* with respect to the infrastructure of TripCom: they are established through external mechanisms, such as parties' agreements, and cannot benefit from an infrastructural support. *Kernel → kernel* relationship (1), instead, depends on the distribution strategy to be yet defined by WP2, so we defer its analysis to a future stage of the project (task T5.4).

Currently, trust relationships concerning access control (4, 6, 7) are explicitly stated by means of *trust filtering rules* about AP in the TAM policy (see chapter 3). Such a simple approach may be refined in a subsequent phase of the project (again, task T5.4), when more feedbacks coming from the implementation of the first prototype will be available. In particular, we envisage the use of a *web-of-trust* approach in order to implicitly derive trust in AP. In any case, it is important to note that whatever the

trust mechanism adopted, it will operate entirely at the infrastructural level, since it deals with access control and should support the decision of trusting or not a given attribute assertion; therefore, this mechanism must support all the three main activities of a trust management system as listed in section 6.1.

The remaining two trust relationships (8, 9), namely *client* → *space* and *client* → *client*[1], are the least significant from the point of view of the infrastructure. In this case, trust evaluation and decisions depends on the business logic of the particular application running on clients, which is unknown to the infrastructure. Nonetheless, clients evaluating this kind of trust can adopt a computational approach, which benefits from shared and public information collected at the infrastructural level (e.g. opinions). Therefore, among the three trust management activities of section 6.1 we will concentrate on the first one and, especially, on the acquisition and exchange of shared reputation information which is "produced" in a collaborative way, such as client opinions. The other two activities, namely trust evaluation and decision, are out-scope from the point of view of the infrastructure, since they are characterized by a subjective nature and depend on the business logic of clients. In the overall vision, a client assessing its trust level in a counter-part will access the information maintained by the infrastructure, merge it with local trust information (e.g. direct experience and explicit trust judgements) and finally feed all the data to a local trust management system, which produces the estimate; trust decisions will be taken locally by clients, on the basis of this estimate.

6.3 Support for the acquisition of trust information

As seen in the previous section, the activity which benefits the most from an infrastructural support is the acquisition and management of public trust information (such as opinions, evidences and sociological data). Among this information, clients' opinions are especially relevant. In TripCom, they can represent the (numerical and textual) judgements of clients about entities such as other clients, published resources, whole triple spaces and, more generally, every identifiable object. Taken one by one, opinions can be interpreted as recommendations, while as a whole they can be aggregated to compute shared reputation values. Therefore, in this deliverable we will concentrate on the infrastructural support to the management of opinions and, particularly, on their acquisition, representation and interrogation.

The solution proposed comprises the following three steps, detailed in depth in the remainder of the section:

1. *Formalization of opinions and proposal of a "standard" representation*, generic enough to be employed by applications with different needs. The aim is to define a sort of "lingua franca", which supports interoperability among applications and, for instance, enables the aggregation of opinions regarding a same object across different applications (e.g. of e-commerce), in order to compute a global reputation value (e.g. about a merchant).

2. *Identification of the principal operations* for the management of opinions, with an *analysis of their security requirements*.

[1] we omit the *client* → *information* relationship (10), since it can be expressed in terms of the other two relationships

3. *Proposal of a basic solution for opinions' management*, based on the functionalities already offered by the TripCom platform and the access control mechanisms described in previous chapters; the solution realizes interoperability at a basic level and can be extended by clients (or, even, kernel components) in order to support more advanced functionalities.

6.3.1 Opinion modelling

An opinion is the *judgement* expressed by an *author* regarding a certain *object* in a given instant in *time*:

- The author is always a client[2], whose identity is established by the system from the assertions provided (see chapter 3)
- The object can be any entity, provided that it can be globally and unambiguously identified (e.g. by means of a URI); in particular, the object of an opinion can be another client or a triple space, judged for the quality of the contents or services provided.
- The judgement is a critical aspect to model, since it is difficult to conceive a metric general enough to satisfy the needs coming from different applications. In this phase of work, we assume that a judgement can be decomposed into a discrete trust score, for machine consumption, and an optional textual comment, for human consumption. The trust score expresses an estimate of the trustworthiness of the object, as seen by the author and on a discrete scale consisting of the tree values *negative*, *neutral* and *positive*. The rationale behind this simple metric is that opinions will be produced by different (and autonomous) clients, so it is important that everyone gives the same meaning to a given trust value; we believe that a simple discrete scale such as the one proposed can be clearly understood and adopted by clients, as opposed to a more expressive solution, such as a continuous scale or a compound value. In any case, the applicability of the proposed metric will be verified during the subsequent implementation and validation phases.

The definition provided can be modelled in ontological terms through the two classes Opinion and TrustScore, described as follows using the notation introduced in chapters 3 and 4.

Opinion

This class models an opinion; note that we model the object with a generic term *URI*, which underlines that the object of an opinion can be every identifiable entity.

```
Opinion hasAuthor <1> Agent
Opinion hasObject <1> URI
Opinion hasComment <1> string
Opinion hasTrustScore <1> TrustScore
Opinion hasCreationDate <1> xsd#dateTime
```

[2] *agent* in the triple space ontology

TrustScore
This class models a discrete trust score and is defined in terms of the enumeration of the three possible values: *Negative*, *Neutral* and *Positive*.

6.3.2 Operations and security requirements

We can identify two families of operations for the management of opinions:

1. *Editing operations* include the creation, update and removal of opinions. Note that while the first one is necessary for the system to operate, the latter two can be removed if we impose that opinions are immutable after their creation. With this assumption, the system will maintain all the opinions with the same author and object[3]; such an "history" can be exploited for deriving more accurate reputation values.

2. *Query operations* enable the retrieval of opinions on the basis of their object, author or other complex criteria or templates. In particular, we can envisage complex queries which combine criteria over opinion's properties with criteria affecting other data stored in related triple spaces (e.g. a query for all opinions about objects of a given category or with a specified property value).

Starting from these operations, we can identify the following security constraints:

1. Execution of described operations should be regulated by access control policies, which can be the same policies described in chapter 3 and 4 if opinions will be maintained in a regular triple space.

2. The author of an opinion must correspond to the identity of the client which creates it; if modifiable, opinions can be edited or removed only by their original creator.

Additionally, it would be worthwhile to allow the author of an opinion to remain anonymous to other clients (or even the infrastructure), in order to limit the risk of reprisals and of disclosure of sensible information about acquaintances or previous interactions between authors and objects of opinions.

6.3.3 Basic infrastructural support

A basic reputation system using the opinions and providing the operations previously described can be realized using only the functionalities already provided by the TripCom infrastructure, together with the security mechanisms described in chapters 3 and 4.

We assume that a space authority, controlling one or more triple spaces, wishes to store opinions about entities relevant to a particular application domain (e.g. the products sold in an e-commerce system) and expressed by specific clients (e.g. the buyers or all the registered users); the final goal is to enable clients to compute reputation values from these opinions, or to use them as recommendations (e.g. when they are expressed by trusted clients).

[3]removing older opinions if required by storage constraints

The most natural way to handle opinions in TripCom is to store them in a regular triple space. To this point, the space authority can arrange for a dedicated space, that we call *reputation space*, which memorizes opinions and possibly other reputation-related information. Access to this space will be governed by an AC policy with the following rules (see Figure 6.2):

1. Every authorized client (e.g. all the registered buyers of the system) can read and add new opinions.

2. Nobody can update or remove pre-existing opinions (with the obvious exception of the authority of the space).

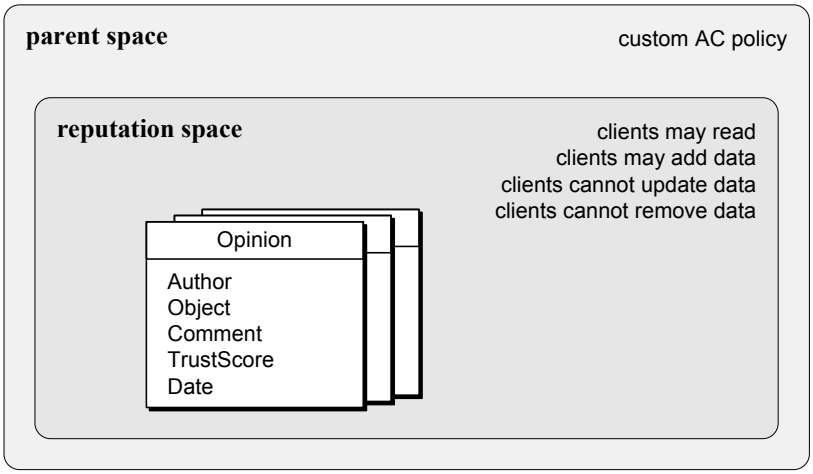

Figure 6.2: Reputation space and AC rules

Given these rules, all authorized clients are able to add opinions but not to change or remove them. Therefore, opinions are immutable and accumulate in the system (as we pointed out when describing editing operations in 6.3.2), so a proper mechanism should be arranged for removing out-dated opinions. With this approach, clients can query for existing opinions using the regular TS API (provided they are given the identifier of the target reputation space); even complex queries and queries spanning multiple spaces can be executed, without the need to introduce new reputation-specific operations in the API. Moreover, a query for opinions against a root space will extract all the opinions maintained in the reputation spaces descending from it, even if under different authorities, as long as read access is granted by individual (sub-)spaces' policies. Retrieved opinions can be used as recommendations, or they can be aggregated by clients in order to compute reputation values about specific entities.

One point deserving more attention is *author identity*. We already stated that every opinion should be associated to an author. This requirement is needed in order to avoid a client to post several anonymous opinions (i.e. opinions without an author) about the same entity, in order to influence its reputation to a greater extent. By specifying an author, these opinions can be recognized by clients, which are then able to use only the most recent ones from the same author.

It is also important that the reported author corresponds to the client who creates the opinion. Author validity must be enforced by the system, in order to prevent a malicious client (though authorized) to post multiple opinions using different identities.

To this end, we can use as author the client's identity already assessed by the system using the assertions provided (see chapter 3). Here, we assume that this information can be accessed by clients (at least, on a space by space basis), so we do not store authors as data but we use this information instead. Indeed, this assumption is also a pre-requisite of any reasonable reputation system in TripCom: if we cannot associate an author to a given piece of information published on a triple space (whatever it represents), then we cannot collect opinions about its author since it is unknown.

While we advocated so far for the inclusion of author information, an anonymity requirement is not incompatible with the scheme described. A form of pseudo-anonymity can be achieved by using *pseudonyms* as authors, instead of identities which allows other client to unambiguously identify their owners. This solution requires that each client will be associated only to a pseudonym and will maintain it over time; otherwise, clients would be able to present multiple identities in subsequent interactions with the system, so the final effect would be same one achieved through the anonymous opinions seen before. Therefore, the association between a client and a pseudonym must be certified by a trusted party (from the point of view of the space authority). In particular, we can envisage the use of attribute providers issuing special *"pseudonym assertions"*, which entitle their owner to exercise the use of a pseudonym.

As already stated, the approach described can be directly used by space authorities, in order to support clients in assessing trust from the opinions published in reputation spaces. However, it is general enough to serve as a basis to more advanced uses at the applicative level. In particular, this solution can be exploited by external applications, wishing to implement a reputation system for their own use, or even by full-fledged *reputation services*, which collect and manage opinions in order to provide (or even sell) reputation information to their users. In all these cases, opinions will be still represented using the approach described above, so interoperability across different triple spaces and applications is assured.

In short, the basic solution described has the main benefit to exploit the functionalities already provided by the infrastructure and, especially, by its security manager. However, more sophisticated solutions may be conceived in a subsequent phase of the work, when feedbacks about the first prototype will be available. In particular, more expressive AC policies may allow the use of *modifiable opinions*, since we can enforce the security constraint that only the original author can update or remove an opinion.

7 Towards a distributed architecture

7.1 Distributed triplespace scenario

The distributed triplespace scenario adds complexity to the security model presented above. Different kernels will be managed by different authorities, and we cannot assume a central authority that rules all of them. Therefore, in this case we must take into account kernel authorities and the trust relationships between them and the data authorities. Moreover, in this scenario it may be the case that a single space can be distributed over multiple kernels, and this adds another element of complexity also with respect to security.

As described in [8], we can use two different approaches dealing with the distribution of an operation which has to gather data stored on different kernels:

- the kernel contacted by the client can tell the client which other kernels to contact, and then the client contacts them directly;
- the kernel contacted by the client has to contact the other kernels, collect the results and return them to the client.

As described in D6.3, Triple Space will follow the second approach, which preserves the space decoupling aspect of the tuplespace model as opposed to client-server communication such as presented in the first approach.

Of course, in the first approach operations are not forwarded from a kernel to another, but are always directly requested by the client to the relevant kernels. Therefore, authentication and authorization are not much different from the single kernel scenario. On the contrary, in the second approach there are kernel to kernel communications and operation requests, which obviously do not exist in the single kernel scenario. In this case, we must consider different alternatives related to authentication and authorization:

- the originating kernel contacts other kernels using only its own identity, thus exploiting kernel-to-kernel trust relationships (the other kernels cannot apply access control policies based on client identity);
- the originating kernel contacts other kernels using its own identity and a delegation stating that it acts on behalf of the client (the other kernels can use both information for access control);
- the originating kernel contacts other kernels impersonating the client (the other kernels cannot distinguish this from being contacted directly by the client).

Note that besides the “originating kernel” (which received the request from the client) and the other kernels which can receive the request to process it, other kernels may play the role as forwarders of the requests being transferred across Triple Space. These kernels may also wish to check the identity of the client which made the request or the kernel which forwarded the request to decide if they forward a request onto other kernels or not. In D6.3 we foresee kernel-to-kernel communication containing the client identity, and as specified in this deliverable, this identity will be expressed as a set of attributes. The communication will also contain the identity of the originating kernel

so that responses can be returned to it. Hence, access could be processed according to both identities: the trust relationship that the receiving kernel has to the originating kernel, as well as the access control policies on that kernel for the client which made the request.

7.1.1 Authentication

Authentication in a distributed environment is influenced by the distribution approach. In client to kernel communication, authentication can be the same as in the single kernel case, but in a multikernel case there can be kernel to kernel communication. With respect to the different alternatives presented above, we can identify the following cases in authentication:

- for kernels passing requests to other kernels with their own identity, a kernel to kernel authentication scheme is needed;
- for kernels contacting other kernels on behalf of clients (i.e. with client delegation), a delegation mechanism is needed (i.e., a way to produce, manage, verify delegations);
- for kernels impersonating the client to other kernels, an impersonation mechanism is needed (i.e., a way for the first kernel to act and authenticate as if it were the client).

Kernel-to-kernel authentication will be important for secure communication *within* the Triple Space, so that other processes may not imitate kernels and provide false requests. Authenticated kernels will need to be trusted, that the requests they pass to other kernels are genuine requests from genuine clients.

7.1.2 Authorization

In a multikernel environment, a space may span over more than one kernel. [1]

Nevertheless, the access control policy is still related to the space, irrespectively of the kernels that manage that space. Therefore, the distribution system must take care of this and guarantee the consistency of the security policy of a given space among all the kernels managing that space. Furthermore, given that kernels will have identities, access control policy may take also kernels into account as possible subjects of those policies.

Another issue arises with respect to the creation of a space. As in the single kernel case, creating a space implies choosing a name and a logical position in the subspace hierarchy on a kernel. However, in a multikernel environment, it also implies choosing one or more kernels on which the new space will reside, i.e. a "physical" position in the infrastructure. This may not be a problem if the parent space resides only on one kernel, and if we want to impose that the child space cannot reside on another kernel. However, if we do not want to limit the infrastructure in that way, and in any case when the parent space spans over more than one kernel, a mechanism is needed to regulate the possibility to link a space with a kernel.

[1] However, it shall be noted that the Triple Space approach will focus on spaces being fully contained in one kernel. Multikernel spaces are a potential extension of this to allow for self-organisation of Triple Space data across kernels.

In our information publishing example introduced in section 2.2, we can see buying a domain name as the process of acquiring the right to use a unique name and a given position in the logical hierarchy. At the same time, buying some space from a hosting company gives the right to "link" that name to a particular machine, this link being expressed in the DNS.

Creating a space in a multikernel environment means having the right to use a particular name and position in the logical hierarchy, as in the single kernel case, but also having the right to put it on one or more kernels. Unlike the first two rights, the last one is not exclusively related to a space, but also depends on the specific kernel(s) we are trying to create a space on, i.e. on the infrastructure layer. This issue can be regulated by some out-of-band mechanism or by a specific kernel policy; in the latter case this policy cannot be linked to a specific space, like the previously mentioned policies, because it substantially deals with the right of creating something on a particular kernel, and not in a particular space.

It is worth noting that this issue is not specific to the creation of a space, but it arises also when a space is moved from one kernel to another (i.e. same logical position in the hierarchy but different "physical" position on the infrastructure), or extended over other kernels. We propose to support moving and spreading spaces across multiple kernels in the Management API (see the next section).

7.1.3 Security policies

Another case is replication of security policies. We can assume that access policies for a space will be replicated across all kernels hosting that space. The disadvantage is this adds extra load to the Management API operations for security policies [2] (changes need to be propagated to all replicas in the Triple Space) as well as needing secure kernel-to-kernel communication. The other possibility is a central storage of the policies on a single kernel (so that we say every space has a "primary" kernel where its access policies are stored). Then maybe we avoid any transmission of security information in the Triple Space network outside of the secure Management API operations between a client and the primary kernel. However we have the risk of the primary kernel becoming unavailable and then no security management can take place.

Each kernel is automatically under the kernel authority with respect to the creation and deletion of spaces on that kernel. Kernels are expected to be quite stable in terms of their kernel authority, i.e. we do not expect kernels typically to change ownership. Spaces, on the other hand, are tied to the kernel they were created within. We assume they may be distributed across kernels sharing the same authority for reasons of performance and scalability. The space has the same access policy on every kernel that hosts it. A space creator may also wish to further distribute the space on other kernels, or even move it fully, providing authorization is possible (we consider the action to be equivalent to a create operation on that node). Hence we define two further operations in the Management API: m-add(space,kernel) and m-move(space,kernel). Space (and hence data) mobility (in terms of kernels) can be very beneficial for system performance but as this goes across kernel boundaries we restrict such mobility to what is explicitly permitted by the kernel and data authorities.

[2] see Section 4.5

7.1.4 Conclusion

We have made some initial observations on the security handling in a distributed Triple Space. Generally, kernels host spaces and hold the access policies for themselves and their hosted spaces locally. They also individually authenticate clients communicating with them, providing cookies and internally associating identified clients to their attributes for the access control policies. We avoid transferring authentication information between kernels. Clients who directly communicate with a new kernel must repeat the authentication procedure. We note however that kernel-to-kernel communication will take place (e.g. forwarding of API requests) and hence should be secure, including the authentication of kernels to other kernels. In the case that spaces are shared across kernels, space policies need to be synchronized across all hosting kernels. This can be done by propagating Management API changes to space policies to the other kernels which host that space.

8 Security functionalities and use cases

8.1 Digital Asset Management

Marketplace systems, as most of other kind of systems, need to guarantee information integrity and availability, as well as to avoid information destruction, modification or revelation without having properly rights of use. Due to this Security Manager will be in charge of providing all required security functionalities, as authentication and authorization ones[2]:

In other words, the Security Manager must:

- Check the credentials of each participant in order to ensure that the participant is who it intends to be.
- Check space policies: Each time a new space is created it is needed to provide specific rights of use. The use of the space is based on roles and policies that are specified by the creator of the space. Also these policies will be used by the Security Manager for authorization purposes.

Next subsections explain the spaces needed in DAM scenario with their policies, and the interaction among this use case participants and Security Manager through TS API.

8.1.1 Spaces and authorization policies

In this section we are going to describe all the spaces that need to be created in the marketplace scenario, as well as the policies these spaces have.
The involved roles are the following [2]:

- *Service Provider* or SP, that provides content services to subscribers by storing a catalogue of proffered media content which it offers through a service portal.
- *Content Provider* or CP: SP needs media content to provide these content services. A CP is a supplier which owns media content and wants to trade with it.
- *DRM Provider* that provides security functions such as privacy, integrity, or authentication which are especially suited for multimedia content .
- *Content Distributor* or CD that is an intermediary that not only provides the subscriber access to the media content; it might also sell value-added services to the content provider, providing storage and bandwidth functionalities.
- *Final Users* that purchase digital contents provided by the SP.
- *Auction Manager* that is the role in charge of creating and managing auctions for different SP.

For security purposes we are only take into account SP, CP and Auction Manager roles.

The marketplace system needs to create the following spaces, that correspond to different data stored in marketplace use case, with the following policies [1]:

- Content Catalogue space: This space, which stores information about contents that CPs publish, is going to be created by the Auction Manager(role explained in [1]). The policies associated to this space are the following
  - All SP can read the data included in the space.
  - Auction Manager can read/write the data included in the space.
  - All CPs can read the data included in the space.

  Only the Auction Manager is able to change the contents of the Content Catalogue space. So when a CP wants to publish a new content or when a CP wants to change the features of a specific content it should write the information in a temporal Content Catalog space. When this information is written the Auction Manager checks it and if it is right and the CP is a registered one, it updates Content Catalog space. So the creation of a temporal Content Catalogue Space is needed for this purpose.

- Temporal Content Catalogue space: Explained in the previous item. The policies associated to this space are the following
  - All SP can read/write data.
  - Auction Manager can read/write data.
  - All CPs can read/write data.

- Auction space: It is going to be created by the Auction Manager. When a SP provider creates an auction, this space is created for publishing all the bids created by the authorized CP. The policies that this space should have are the following:
  - SP that owns the auction can read the data.
  - Auction Manager can read/write/change the data of the space.
  - All authorized CPs can read the data.

  As in the Content Catalogue space is needed to create a temporal space for updating data of the Auction Space. When a CP wants to publish a bid, it will do it in this temporal space. Then Auction Manager will check it and if it is the winning one and CP has rights, it will update Auction space.

- Temporal Auction space: Explained in previous bullet. Policies associated are the following:
  - All SP can read/write data.
  - Auction Manager can read/write data.
  - All CPs can read/write data.

- Service space: Each SP has its own space, with two different parts with diferent rights. One for public information (information related with the services offered) and private information (related with users, black lists and so on) The policies that this space should have are the following:

  - Only SP has access to its private space.
  - Regarding public information other SP can read information, but not update or delete it.

- Negotiation and contract space. It is created by the SP to formalize an auction. Several negotiation messages will be published in it for the negotiation between service providers and selected CP. The policies that this space should have are the following:
  - Only the SP that creates the space has rights to write/read messages in this space.
  - Only selected CP has access to read/write in this spaces.
- CP space: Where CP stores its information. The policies that this space should have are the following:
  - Only CP has access to its private space.
- Supervision space: this space is created by the SP to allow users to evaluate the services consumed. The policies that this space should have are the following:
  - Only the SP that created the space has access to this space, the rest of the SP have not access.
  - Final SP's customers can read the contents.

  To include the evaluation of the service made by final users it is needed to create a temporal Supervision Space. Once a final customer has included the data, SP will check it and if it is right it will upload Supervision space.
- Temporal Supervision space: Explained in previous bullet. Policies associated are the following:
  - All Final customers can read/write data.
  - Only the SP that creates the space can read/write data.

8.1.2 Digital asset Management use case

Once the spaces have been explained in previous section, the use made by DAM participants of the security functionalities provided by the TS is going to be explained in the scenarios that are explained in this section.

Roles participants are the following:

- Attribute Provider: in charge of providing assertions.
- Security Manager: in charge of all related with security issues, like authentication, trust, authorization..
- Participant: interacts with the TS.

Previous participants are going to take part in the next scenarios:

- *Content Publication*

  - One CP wants to publish a content in the Content Catalog. First of all it needs its credentials. To this end it contacts one Atribute Provider to get its assertion. In this assertion there will be some attributes specifying its role(CP).
  - Once the CP has its assertion, it publishes (writes) content fetures through TS API in the temporal Content Catalogue space. Previously the CP has been authenticated and authorizated by the Security Manager, using its assertion. Security Manager has to check that it has rights to write in temporal Content Catalog space, becase space policies specify that all authenticated CP can write in it.
  - Auction Manager checks the data inserted by the CP. If the data is correct and the CP is a registered one, Auction Manager removes it from the temporal Content Catalogue space and writes it in the Content Catalogue space. Otherwise, Security Manager only removes it from the temporal space.

- *Bid Publication*

  The schema followed by the participants in this case is similar to previous one, but the spaces involved are temporal Auction space and Auction space. The roles are the same.

- *evaluation publication*

  - A customer that wants to publish a evaluation has first to have its credentials. To this end it contact with one Atribute Provider to get its assertion. In this assertion there are some attributes that specify its role(Customer).
  - Once the customer has its assertion he/she publishes (writes) the evaluation through TS API in the temporal Supervision space. Previously it has been authenticated and authorizated by the Security Manager, using its assertion. Security Manager has to check that he/she has rights to write in temporal Supervision space, because space policies specified by the SP points out that all authenticated customer can write in it.
  - SP checks the data inserted by the customer. If final customer is a client of the SP this information is inserted in the Supervision space, if not it is only removed from the temporal space.

- *Negotiation publication*

  - In this case the policies established by the SP specify that only one specific CP and itself have access to it.
  - CP and SP can read/write the needed negotiation messages in the Negotiation space for specifying the contract.

8.2 European Patient Summary scenario

This section describes the security functions used by the EPS infrastructure that is built on top of the TS infrastructure. The following subsection discusses about how the European hierarchy of health authority and the treated citizen can be reflected inside

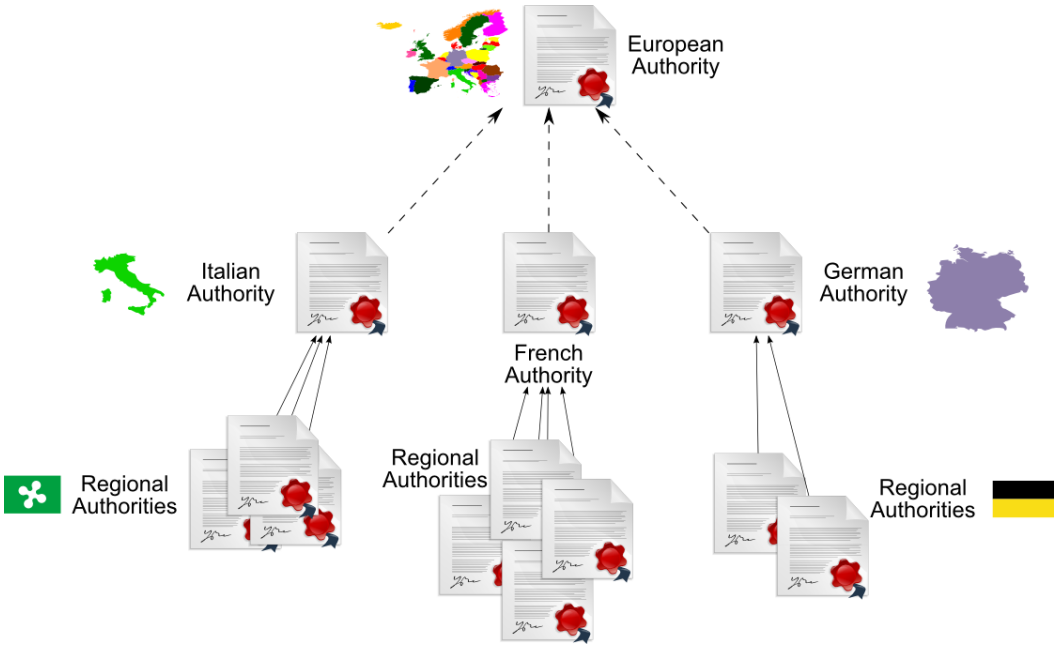

Figure 8.1: The Hierarchy of European Health Authorities

the triplespace in a hierarchy of subspaces. We provide also details about the roles of the users of the EPS infrastructure in those subspaces and some example policies for accessing citizen's data. The next subsection describes a use case based on the EPS scenario that uses the security functionalities provided by the TS accordingly to the subspaces structure.

8.2.1 Spaces and Authorization Policies

Figure 8.1 shows an example about the typical structure of health authorities in Europe. There are some national authorities that are internally subdivided in regional authorities that can be further subdivided in local authorities. This subdivision is officially recognized and driven by the national authorities that delegates the treatment of citizens on a geographical basis.

Actually, an European authority doesn't exist. Anyway, the European challenge of a Patient Summary at European level implies that a minimal common set of rules, data structure and security policies is defined at European level. We indicated the links with this common set using dashed arrows to emphasize the difference with the official rules drawn as straight arrows.

Within the scope of the EPS scenario, the European authority provides a set of recommended roles for the EPS users. We assume that these roles are defined and structured as follow:

Medical Staff These roles include all the medical staff that access the EPS infrastructure. The specific sub-roles are the following:

- General Practitioner
- Specialist
- Emergency Doctor

- Researcher

Medical Support These roles include all the staff that supports the medical staff, such as:

- Paramedic, such as a Red Cross rescuer or civil protection agent
- Nurse

Administrative Staff These roles include all the non-medical personnel that can have access to the EPS infrastructure for performing administrative and insurance related activities:

- Front Office Staff, such as Acceptance/Dismissal that is responsible for assigning/unassigning a specific citizen to one or more specific specialists
- Back Office Staff, who have to create a new summary, to access the head information of the summaries, to assign a specific general practitioner to a specific citizen

TS Administrator of the Authority This is the role of the administrator of the authority that, when needed, defines the roles and configures the TAM and AC policies that are enforced on the spaces it administers.

Each health authority of the hierarchy inherits those roles from the upper level. Accordingly to national, regional or local healthcare regulations, it's possible that some authorities redefine the roles by renaming or extending the roles list.

Beside the roles, there are permissions to access the data of the citizens managed by the health authority in the EPS. Such permissions are: Read/Write triples, Subscribe/Unsubscribe, Read Audit, Change Permissions.

Finally, policies link the roles with data in the summaries and the relative permissions to access this data. Examples of policies in function of the role of the user and the data accessed are:

General Practitioners Read the whole summary of the treated citizens; Write to the body of the summary of the treated citizen; No access to the summary of non-treated citizens.

Specialist Read the whole summary of *temporary* treated citizens; Write to some specific sections of the summary (e.g. Problems, AdvanceDirectives, Procedures, Medications, Encounters, Alerts, Immunizations, ...); No access to not temporary treated citizens.

Emergency Doctor Read the whole summary of citizens just after an identification; Write to some specific sections of the summary (Problems, Medications, Encounters, Alerts).

Researcher Read data from the summary of the citizens only if the query has been approved by the Privacy Administrator.

Paramedic Read specific sections (Problems, Alerts, Immunizations, Procedures, ...); Write to some specific sections of the summary (Medications, Encounters, Alerts, ...).

Nurse Read specific sections (Procedures, Alerts, VitalSigns).

Front Office Staff Change permission on a specific summary to one or more specific specialists

Back Office Staff Write new summary, Read and Write head information, Change permission on a specific summary to a specific general practitioner

From the TS perspective, the hierarchy of health authorities results in an equivalent hierarchy of subspaces. Those subspaces don't overlap and inherit the policies from the upper level. In case of changes or refinement of policies in the lower level, the consistency of the policies is due to the lower level authority that change the policies.

Inside the subspace of any local authority, there are the patient summaries of the managed citizens. Each summary is a specific subspace of the subspace of the authority. In this way, each summary can inherit the security policy of the authority and can add other specific policies (e.g. only the GP responsible of the citizen can access the data in the summary, the other GP must haven't any access). Also the summaries are subdivided in subspaces, this is due to the fact that some specific policies require to define access permission on part of the summary (e.g. problems, alerts, ...).

8.2.2 Shared Care Path use case

This subsection reports a use case based on the EPS scenario that uses the security functionalities provided by the TS according to the subspaces structure described above.

- A specialist that needs to access the data from the EPS has to authenticate herself with the EPS through her own local application (e.g. the electronic Health Record - eHR). The local application contacts the Identity Provider (IDP) of the local health authority and provides the credentials of the specialist.
- As result, the IDP returns a certificate that includes a list of attributes that can be mapped on her role in the EPS infrastructure.
- The local application provides the certificate to the TS infrastructure with information about the subspaces to access (derived by the identification of the summary of the citizen to access)
- The TS returns back the security cookie to be used for the communicating with the TS over the identified subspaces.
- The local application can now submit the necessary queries to the TS via the TS API providing the security cookie just received. If the local application asks for the whole summary, the TS will return only the data which the user may access, accordingly to the security policies defined in the subspaces queries and the role of the user.

9 Conclusions

This document contains the first design of the security and trust support model for the Triple Space. The resulting design aims at adding security functionalities to the Triple Space infrastructure by adapting security solutions to triplespaces and exploiting the space-based technology of the kernel. The design is targeted at the single-kernel environment, and will be expanded in a multi-kernel direction in task T5.4. Following use case needs and general-platform related requirements this model mainly focuses on access control features that are provided by the Triple Space to clients when they access to spaces and subspaces. Access control decisions are taken following two subsequent policy filtering steps: TAM, dealing with trust in assertions that carry client-related attributes and mapping from attributes to roles, and AC, that defines access control policies on the basis of the role and of the target resource. Policy inheritance is designed to be coherent and exploit the tree-structures of the Triple Space, and policies are expressed at the granularity level of spaces and subspaces, in order to make things simple enough to be usable and powerful enough for clients' security needs. This document also suggests an internal organization of the security flow for the kernel, exploiting the coordination middleware features for interaction among the functionalities of the Security Manager and the other components in the kernel. Furthermore, the integration middleware is meant to be the element where logging functionalities are provided. A model of reputation information that can be adopted by services as a foundation to build a trust management system is then described in this document. Given that only the clients can take trust decisions, the model for reputation information is provided by the generic TS platform, while every client will be able to decide what kind of use it will make of such information according to its domain-dependent perspective. Though this document is specifically addressed to the single-kernel environment, some considerations about multi-kernel security issues are taken into account too. In the final part of the document, TripCom DAM and EPS use cases are used to show how they can use the features (basically access control) that are provided by the solutions designed in this deliverable.

An extended version of the security model will be designed in the second phase of the project, with refinements coming from the evaluation of the first implementation, and taking into account distribution issues.

A Triple Space Security Ontology

The following listing is the Triple Space Security Ontology in WSML syntax.

```
wsmlVariant _"http://www.wsmo.org/wsml/wsml-syntax/wsml-flight"

namespace { _"http://www.tripcom.org/ontologies/ts-sec-onto#",
    rdf _"http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    rdfs _"http://www.w3.org/2000/01/rdf-schema#",
    xsd _"http://www.w3.org/2001/XMLSchema#",
    foaf _"http://xmlns.com/foaf/0.1/",
    dc _"http://purl.org/dc/elements/1.1/",
    ts _"http://www.tripcom.org/ontologies/tsonto#" }


ontology _"http://www.tripcom.org/ontologies/ts-sec-onto.wsml"


importsOntology _"http://www.tripcom.org/ontologies/tsonto-core.wsml"


concept ts#Space
    hasSecurityData impliesType (1) SecuritySpace

concept SecuritySpace
    hasTrustFilteringRule impliesType TrustFilteringRule
    definesRole impliesType Role
    hasRoleMapping impliesType RoleMapping
    hasPolicySet impliesType (1) PolicySet



concept TrustFilteringRule
    hasAttributeProvider impliesType (1) AttributeProvider

concept PreestablishedTrustRule subConceptOf TrustFilteringRule
    hasAttributeConstraint impliesType (0 *) AttributeConstraint

concept AttributeProvider




concept Role

// most instances of Role are user-defined; here go the predefined ones
instance admin memberOf Role
instance banned memberOf Role
instance everyone memberOf Role

concept RoleMapping
    hasAttributeConstraint impliesType (0 *) AttributeConstraint
```

```
    hasRole impliesType (1 *) Role

concept AttributeConstraint
    hasAttributeName impliesType (1) _string
    hasAttributeValue impliesType (0 1) _string



concept PolicySet
    hasPolicySet impliesType PolicySet
        // implied by the space hierarchy
    hasPolicy impliesType (1) Policy
        // implied by the space ownership
    hasPolicyCombiningAlgorithm impliesType (1) PolicyCombiningAlgorithm
        // default is First-applicable
    hasTarget impliesType (1) Target
        // this is implied, the target only specifies the ts:Space

concept PolicyCombiningAlgorithm

instance Deny\-overrides   memberOf PolicyCombiningAlgorithm
instance Permit\-overrides memberOf PolicyCombiningAlgorithm
instance First\-applicable memberOf PolicyCombiningAlgorithm
instance Last\-applicable  memberOf PolicyCombiningAlgorithm



concept Policy
    hasRules impliesType (0 1) RuleSeq
        // Rules in RDF are in an ordered rdf:Seq container,
        // in WSML it's a head-tail list
    hasRuleCombiningAlgorithm impliesType (1) RuleCombiningAlgorithm
        // default is First-applicable
    hasTarget impliesType (1) Target
        // this is implied, the target only specifies the ts:Space

concept RuleSeq
    hasRule impliesType (1) Rule
    hasTail impliesType (0 1) RuleSeq

concept Rule
    hasTarget impliesType (0 1) Target
    hasEffect impliesType (1) Effect

concept Effect

// there are only two members of Effect
instance Permit memberOf Effect
instance Deny memberOf Effect

concept RuleCombiningAlgorithm
```

```
instance Deny\-rule\-overrides   memberOf RuleCombiningAlgorithm
instance Permit\-rule\-overrides memberOf RuleCombiningAlgorithm
instance First\-applicable\-rule memberOf RuleCombiningAlgorithm

concept Target
    // A policy target points to exactly one ts:Space and no actions or
    // roles. A rule target represents a set of subjects and a set of
    // actions (at least one of each), and no ts:Space.

    hasSubject impliesType Role
    hasAction impliesType Action
       // hasSubject and hasAction only on rule target
    hasResource impliesType (0 1) ts#Space
       // hasResource only on policy target

concept Action

instance Read memberOf Action
instance In memberOf Action // means Read and delete
instance Out memberOf Action
instance Subscribe memberOf Action // also means unsubscribe
instance Create memberOf Action // also means destroy


concept AccessDeniedLogEntry subConceptOf ts#AccessLogEntry
```

References

[1] David de Francisco Marcos et al. Eai prototype application. Technical report, TripCom Project Deliverable D8.2, 2007.

[2] David de Francisco Marcos et al. Tripcom requirements analysis and architecture profile for eai applications. Technical report, TripCom Project Deliverable D8.1, 2007.

[3] G. Joskowicz e. Kühn, J. Riemer. Extensible extensible virtual shared memory (xvsm) - architecture and application. Technical report, Institute of Computer Languages, Vienna University of Technology, Austria, 2005.

[4] Dario Cerizza et al. Security and trust requirement analysis and state-of-the-art. Technical report, TripCom Project Deliverable D5.1, 2006.

[5] Reto Krummenacher et al. Specification of the triple space ontology. Technical report, TripCom Project Deliverable D2.2, 2007.

[6] Diego Gambetta. *Can We Trust Trust?*, chapter 13, pages 213–237. Basil Blackwell, 1988. Reprinted in electronic edition from Department of Sociology, University of Oxford, chapter 13, pp. 213-237".

[7] Tyrone Grandison and Morris Sloman. A Survey of Trust in Internet Applications. *IEEE Communications Surveys and Tutorials*, 3(4), September 2000. http://www.comsoc.org/livepubs/surveys/public/2000/dec/index.html.

[8] Martin Murth, Gerson Joskowicz, eva Kuhn, Dario Cerizza, Davide Cerri, David de Francisco, Alessandro Ghioni, Reto Krummenacher, Daniel Martin, Lyndon Nixon, Nuria Sanchez, Brahmananda Sapkota, Omair Shafiq, and Daniel Wutke. Triple space reference architecture. Technical report, TripCom Project Deliverable D6.2, 2007.

[9] OASIS. extensible access control markup language 2 (xacml) version 2.0 3 oasis standard, 1 feb 2005. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

[10] OASIS Security Service TC. Security assertion markup language (saml) v2.0. http://www.oasis-open.org/specs/index.php, March 2005.

[11] M.D. Welsh. *An Architecture for Highly Concurrent, Well-Conditioned Internet Services.* PhD thesis, University of California at Berkeley, 2002.

[12] Xvsm home page. www.xvsm.org.