D1.2
SUPERCLOUD Self-Management of Security Specification

Project number:	643964
Project acronym:	**SUPERCLOUD**
Project title:	User-centric management of security and dependability in clouds of clouds
Project Start Date:	1st February, 2015
Duration:	36 months
Programme:	H2020-ICT-2014-1
Deliverable Type:	Report
Reference Number:	ICT-643964-D1.2/ 1.0
Work Package:	WP1
Due Date:	Oct 2015 - M09
Actual Submission Date:	5th November, 2015
Responsible Organisation:	IMT
Editors:	Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens
Dissemination Level:	PU
Revision:	1.0
Abstract:	This deliverable describes the specification of Security Service Level Agreement and Security Self-Management that will form the foundations of security resource requests for customers, specifying their security policy requests and negotiation capabilities as well as the requested audit levels that provide information and feedback about actual enforcement across service providers.
Keywords:	Multi-cloud, Self-Management, Security Service Level Agreement, Authorization, Trust Management

This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 643964.

This work was supported (in part) by the Swiss State Secretariat for Education, Research and Innovation (SERI) under contract number 15.0091.

Editors

Reda Yaich, Sabir Idrees, Nora Cuppens, Frédéric Cuppens (IMT)

Contributors (ordered according to beneficiary numbers)

Marc Lacoste, Nizar Kheir, Ruan He (ORANGE)
Khalifa Toumi (IMT)
Krzysztof Oborzyński (PH HC)
Meilof Veeningen (PEN)
Paulo Sousa (MAXDATA)

Disclaimer

Executive Summary

In this document, we present a preliminary architecture of SUPERCLOUD security self-management. We first identify and describe the design requirements of the *user-centric self-management* of multi-clouds security. Then we review state-of-the-art looking for candidate solutions to address these requirements. In this survey, we introduced basic concepts and existing approaches related to *Security Service Level Agreements*, *Authorization/Access Control* and *Trust Management*. In the second part of the document, we present a preliminary architecture of security self-management. We describe the main building blocks we identified to address SUPERCLOUD use-cases requirements. We conclude with a summary and a presentation of ongoing and future steps.

Contents

List of Figures

Introduction

Enabled by performant virtualisation techniques and broadband internet connectivity, cloud computing has evolved from technologies like grid computing and Web Services to demonstrate in few years a clear advantage with respect to traditional on-premise solutions in term of scalability, elasticity and cost reduction [10, 56]. Nevertheless, moving to the cloud, and especially to multi-clouds, brings to discussion serious security and privacy risks with a high potential harm to customers' and users' data and services [132].

In this deliverable we focus on security, trust and privacy issues to provide a preliminary specification of Security Service Level Agreement and Security Self-Management that will form the foundations of security resource requests for customers, specifying their security policy requests and negotiation capabilities as well as the requested audit levels that provide information and feedback about actual enforcement across service providers.

Motivation and Challenges

Along with the benefits mentioned above, migration to the multi-clouds poses serious risks that naturally makes customers and users anxious about the integrity, confidentiality and availability of their data and services. The security risks raised by multi-clouds are different than those faced in traditional cloud solutions. Some of these issues are new (e.g., remote processing of sensitive data over multiple providers), while others are just exacerbated by the multi-cloud approach [85]. Of course, the degree of risk depends on many factors which makes migration to multi-clouds a very complex decision. For example, deployment models, providers trustworthiness and data sensitivity are three different factors that brings to discussion different security issues. In SUPERCLOUD, multiple factors are combined, making the design of secure and trustworthy environment a highly challenging objective. Both security, privacy and trust issues are raised in more complex configurations with respect to traditional systems [85, 132].

With respect to ***Security***, multi-cloud configuration offers several challenges such as ***loss of control*** for the customer/user, ***lack of interoperability*** between security policies and mechanisms offered by each provider, and ***complexity of administration and management*** of security over multiple security domains. In addition, threats such as *denial of service attacks*, *side channel attacks*, *man-in-the-middle attacks* and *inside-job attacks* are generally exacerbated by cloud but not specifically caused by [45, 44].

When it comes to ***Privacy*** issues in multi-cloud systems, the context becomes very important as the risks differ according to the type of scenario. With respect to that, multi-clouds pose significant challenges to institutions that handle personal information. For instance, in the SUPERCLOUD project use-cases, all scenarios raise significant challenges that relate to ***compliance with regulations and legal standards*** (e.g., HIPAA) while collecting, transferring, processing, sharing and storing personal medical records. More generally, *outsourcing* of data and services is systematically associated with decrease of *data control*. Any outsourced *data* or *services* are exposed risks due to loss, abuse and manipulation resulting in ***violation of its integrity, confidentiality and/or availability*** [87].

Finally, **Trust** is also an important concern in multi-clouds. The infrastructure offered by an untrusted provider can be considered as an hostile environment wherein security objectives cannot be guaranteed. Thus, despite deploying appropriate security mechanisms, cloud provider must adopt an adequate *trust management model* to gain *customers* confidence. *Trust* is also required to manage relations inside the cloud infrastructures, *cross-layers* within the same domain of trust, and *cross-providers* to ensure the continuity of service in multi-cloud configurations.

User-Centric and Self-Managemed cloud Security

In the light of the challenges above, a user-centric control and management of security appears to be a necessity more than a desire for multi-cloud customers. However, when the dynamic and complex aspect of security management is coupled with the scalability and heterogeneity of muti-cloud environments, it becomes very challenging for a human to manage them. A user-centric management of security in multi-cloud systems is challenging at least for the three following reasons:

- **Heterogeneity**: security mechanisms deployed in cloud infrastructures differ from a provider to another. Furthermore, the security solutions used within the same infrastructure are often heterogeneous in terms of services (e.g., identity management, intrusion detection) and require high agility to be coherently configured. As a consequence, the control of clouds that span multiple providers is very challenging, and almost an impossible task for a human;
- **Dynamicity**: cloud systems evolve in dynamic environments where resources are unpredictably created and updated, and users can join and leave the cloud at will. As a consequence, security services need to be adapted constantly to fit environmental changes and continue achieving their objectives;
- **Scalability**: Multi-clouds are distributed and decentralized systems that span several cloud infrastructures administrated by distinct providers. The delivery of expect services implies a seamless orchestration of several components that belong to distinct security domains. The coordination and orchestration of such systems is a complex and error-prone task.

To that aim, the SUPERCLOUD project advocates the joint adoption of *User-Centric* and *Self-Managed* security cloud infrastructure. While *User-centricity* aims at bringing back the control to of security configuration to the user, *Self-Management* facilitate and automates this objectif by adapting design principal to security. **Autonomic computing** is a good candidate for addressing the above challenges as it enables software systems to manage themselves without or with a limited human intervention. We assume that this can be achieved in SUPERCLOUD through four objectives that we list hereafter.

- *Self-Service Security* aims to bring back to the user the control of the security of his resources. This control shall be achieved in a fine-grained and flexible way.
- *Self-Managed Security:* to easily and in an automated manner instantiate security requirements into actionable security mechanisms to detect and react to threats. The objective is also to build a security management system that is able to manage itself with a minimal intervention of a human via full automation of the security management process.
- *End-to-End Security* to overcome heterogeneity of security mechanisms and provide an uniform control management to the customer.
- *Resilience:* to enhance the robustness of the multi-cloud and avoid relying on a single provider.

Organization of the document

The rest of this document is organized in four chapters. In Chapter 2, we identify and describe the design requirements of the *user-centric self-management* of cloud security. Then in Chapter 3 we introduce basic concepts and existing approaches that relate to *Service Level Agreement and Security Service Level Agreements* (cf. Section 3.1), *Authorization and Access Control* (cf. Section 3.2) and *Trust Management* (cf. Section 3.3). In Chapter 4 we present a preliminary architecture of self-management of security. We describe the main building blocks we identified to address the requirements stressed in Chapter 2. Finally, in Chapter 5, we conclude with a summary and a presentation of ongoing and future steps.

Requirements Analysis

Introduction

Multi-clouds, as well as cloud federations, are demonstrating a clear advantage with respect to traditional mono-provider cloud solutions. However, this new cloud model further exacerbates security, trust and privacy issues. In the meanwhile, *multi-clouds* raise new threats and issue that need to be identified and addressed to enable its wide adoption.
In this section, we present and analyse security requirements that should adhered to during the specification of Self-Management architecture. A security requirement is a constraint on the functioning of the system in order to achieve security goals. The decision to put all the attention of this section on security, trust and privacy requirements stems from the criticality of theses aspects in the SUPERCLOUD project.

We first present requirements towards security properties of SUPERCLOUD based on use cases provided by **Philips Healthcare** and **MAXDATA**. Then we generalize these to cover others use-cases such as Network function virtualization (NFV) scenario or any other scenarios wherein SUPERCLOUD could be used.

Healthcare requirements

In this Section we describe the security objectives and requirements of two SUPERCLOUD partners, namely:

- Philips Healthcare and Philips Research and their Medical Imaging Platform
- MAXDATA that provide the Healthcare Laboratory Information System CLINIdATA®LIS

Healthcare Laboratory Information System

The healthcare laboratory information system, henceforth mentioned as CLINIdATA®LIS, is a cross-platform web application where server components may run on any common operating system (e.g., Linux, Mac OS X, Solaris, Windows) and relational database (e.g., MySQL, PostgreSQL, Oracle, SQL Server). This system needs to integrate with dozens of other clinical and non-clinical information systems (e.g., ICU, patient identification, billing, regional health portals) and includes a set of real-time interfaces with physical electronic equipments, namely automated analysers.
CLINIdATA®LIS computes and stores medical data along with other personal data, so the SUPERCLOUD infrastructure should comply with Directive 95/46/EC and the soon to come General Data Protection Regulation (GDPR). This implies previously stated requirements regarding integrity, confidentiality and privacy, and the following requirements:

- **Location Self-Management** Users should be able to self-manage the set of possible physical locations, at country level, where users' data may be stored and computed.

CLINIdATA®LIS is used by different types of healthcare organizations, ranging from small laboratories with a few dozens of professionals and hundreds of transactions per day, to very large hospital

clusters with thousands of professionals and tens of millions of transactions per day. Hence, these organizations should be able to control how their resources are protected using SLAs including for instance agreed levels of availability, redundancy, backup, disaster recovery, etc. This implies the following requirement:

- **Architecture and Data Self-Management** The user should be able to self-monitor actively its allocated resources, while enabling a high level of customizability of the computation / storage / network architecture and its services, including agreed levels of availability, redundancy, backup and disaster recovery.

As mentioned before, CLINIdATA®LIS includes a set of real-time interfaces with physical electronic equipments, namely automated analysers. These interfaces are used mostly to send commands to analysers and to receive exam results. Both communication flows have real-time requirements, involve computation and need to get data from storage. This implies the following requirement:

- **Self-Management of Real-Time Properties** The user should be able to self-manage the real-time guarantees of the computation / storage / network planes, including the definition of predictable and bounded computation / communication / storage access times.

In order to ensure the correct operation of CLINIdATA®LIS, the SUPERCLOUD self-management infrastructure should ensure that it is able to deliver the agreed SLAs, and degrade in a graceful way when it is not possible to comply with the SLAs. This implies:

- **Automatic capacity management** The SUPERCLOUD infrastructure should ensure that it is able to deliver the agreed SLAs both in terms of computation, storage and network. When it is not possible to deliver the agreed SLAs, services should degrade/adapt gracefully in a dependable way, according to a set of predefined rules agreed with the user[1].

Medical Imaging Platform

There are three main Philips Healthcare use-cases that aim at deployment into SUPERCLOUD infrastructure, namely:

- Cloud data storage and disaster recovery use-case
- Cloud data storage and processing use-case
- Distributed cloud data storage and processing use-case

Cloud data storage and disaster recovery use case

Cloud data storage and disaster recovery use-case focuses on helping hospital to ease management of the patient data stored in the hospital archive. Current hospital archives are on-premise solutions that need to handle all clinical data, especially imaging studies which can be as large as 1GB. Therefore, it would be attractive to offload this data into cloud, such that storage size on-premise archive can be limited. For example, the data from the last 6 months is stored on premise, whilst a cloud stores it for a longer period (e.g., 10+ years). In this use-case, the cloud becomes an extension of the hospital archive. The core value of this solution is to ensure that patient data is not lost. As a result the cloud storage is also a disaster recovery solution for the hospital. The workflow of data extends to the cloud but performance is not primary concern here, as patient data is always prefetched from the cloud to the on-promise hospital archive prior to the scheduled examination.
From security self-management perspective it is needed to enable selection of:

[1]For instance, the user may define that in overload scenarios when computation power is not enough to answer requests according to the agreed response time, a percentage of requests should be dropped to ensure that accepted requests comply with the defined SLA, or in alternative the user may want to sacrifice response time and don't drop any request.

- Compliance with security and privacy requirements such as the European Commission Directive 95/46 EC, the United States Health Insurance Portability and Accountability Act (HIPAA), etc.
- Medical data privacy measures
  - Data is stored in such a way to minimize the security breaches leading to data tampering and corruption
  - Data must not be interpretable in transit and rest, covering various system layers, such as disk, file system, and database;
  - Role based access control
  - Detailed audit trails and activity monitoring
- Medical data protection level against loss or corruption ensuring high availability
- Medical data location restriction
  - Guaranteed cloud storage within certain countries or regions
  - Distributed storage, across multiple cloud and countries, in a privacy compliant manner

Cloud data storage and processing use-case

Cloud data storage and processing use-case focuses on easier access of the medical personnel to the medical data processing applications. The access to the applications can be possible then not only from the hospital lab where the equipment is installed but also from PCs in the hospital/home or secured mobile devices. This way the doctors collaboration can improve as well due to easier availability of data. This use-case involves only a single hospital organization.
From security self-management perspective it is needed to enable selection of:

- The same security demands as in *Cloud data storage and disaster recovery* use-case
- Storage and processing isolation per hospital group
  - Special permissions are required to access data outside own hospital context (role-based-access control)
- Level of performance guaranties
  - Latency of accessing data in the cloud by the clinical applications from various devices and by various roles.

Distributed cloud data storage and processing use case

Distributed cloud data storage and processing use-case focuses on easier access of the medical personnel to the medical data across hospitals, i.e., this use-case ensures patient centric view to the healthcare professional by showing all data of the patient; so data is not only managed by the local hospital, but also managed by other hospitals. This would enable providing the complete longitudinal patient record. This use-case has highest complexity, as it involves multiple hospital organizations managing patient data. Performance and latency are critical, as imaging results and user interaction must be streamed semi real-time to the clinical user. The user may view mashup of data from multiple clouds and hospitals, or search for comparable reference studies (across the clouds), to assist in diagnosis of the treated patient. Advanced processing may even include comparing large study data, across clouds. As a result, the identity management of the clinical user is becoming critical in this use-case, as it is accessed from multiple organizations.
From security self-management perspective, it is needed to enable selection of:

- The same security demands as in *Cloud data storage and processing* use-case

- Medical data privacy measures
  - Identity management across clouds of healthcare professionals
  - Auditing across clouds of accessed patient data
  - Privacy of stored data, whilst access and queries are still supported
- Interoperability across clouds
  - Data exchange should be possible between Philips managed cloud solutions and 3rd party vendor solutions in full compliance with all agreed privacy and security measures
- Performance guarantees
  - Search performance using privileges across clouds.

Detailed requirements summary:

General:

Standard compliance: security and privacy requirements coming from the following standards are mandatory:
- European Commission Directive 95/46 EC,
- United States Health Insurance Portability and Accountability Act (HIPAA).

Security operational management: Storage must be robust against any security breaches by proper security patch management, secure data sanitization, etc.

Privacy:

Distributed storage: storing data across multiple clouds and countries has to be done in privacy compliant manner.

Availability:

Data availability: 99.99%.

Data lifespan: Storage must ensure data availability for agreed period, i.e.,10+ years.

Data location: Data has to be guaranteed to stay in prescribed legal boundaries.

Efficient data access: guaranteed search performance while using access control across clouds.

Fault tolerance: Data may not get tampered and must be complete and correct.

Confidentiality:

Data security: Data must not be interpretable in transit and storage, covering: disk, file system and database layers.

Data access monitoring: Detailed audit trails and activity monitoring.

Interoperability:

Distributed data access: Queries of ANY data over available clouds shall be possible in compliance with privacy and security rules.

Multivendor data access: supporting Philips managed cloud solutions and 3rd party vendor solutions.

User control:

Data access control: Special permissions are required to access data outside own hospital context (role-based-access control).

Data identification: Identity management across clouds of healthcare professionals.

Generic Requirements

In this section, we extend the requirements provided by SUPERCLOUD partners with general and generic design requirements for Self-Management of Security.

DR1: Availability

Availibility is a key property in terms of *quality of service* and *provisioning*. The security objective here consist in minimizing the risk of threats that can impact the ability of the system to deliver services to its users. This objective can be easily quantified but the metrics and mechanisms used to evaluate are often disputable and need to be redefined and/or adapted for the SUPERCLOUD purpose. We list hereafter the different requirements to achieves *Availbility*.

DR2: Integrity

Users should be able to trust the system in preventing any non *authorized* alteration of their data and resources. In the context of SUPERCLOUD, it is important that data and services that run on cloud providers infrastructures remain integer. The integrity of the *network* communication with and between the different components of the SUPERCLOUD architecture is also important and must be guaranteed.

DR3: User-Centric Security Control

When dealing with the quality of protection required by data and services to be run on a multi-cloud, it is not normal to accept that a single level of security fits the needs of all customers. As it is very difficult to sample beforehand customers security requirements, SUPERCLOUD providers shall provide facilities for customers to define personalized self-security services settings to be deployed in the user's clouds (U-Cloud). Thereby users can control the protection level of their cloud resources. As previously identified by Philips, *User-Control* include control over Data access (already covered by Fine-grained Access control requirement) and control over identification, which include identity management mechanisms.

DR4: Fine-grained Access Control

SUPERCLOUD shall facilitate granting differential access rights to a set of customers, tenants and infrastructure components such as virtual Machines. Such fine grained access control brings high flexibility in specifying access rights. Several techniques are known to implement fine grained access control [51, 5].

DR5: Horizontal and Vertical Privacy Preservation

Data of SUPERCLOUD customers shall be encrypted and accessed by authorized users only. The data shall be self-protected and have security parameters both the provider (vertical privacy) and other customers. Unauthorized entities, including the cloud service provider, cannot gain access to data or gain metadata about the data, when authorized operations are carried out on the data.

DR6: Interoperability and Adaptiveness

In current cloud ecosystems, cloud providers are using different SLA specification mechanisms making their comparison, execution and interoperability difficult. Shifting from classical one-to-one Customer-Provider partnerships to one-to-many complex contracts call for more sophisticated SLA specification languages. Multi-cloud and Federation of clouds call also for interoperability of security of mechanisms to facilitate the uniform deployment of customers security requirements.

The SUPERCLOUD self-management infrastructure should provide users with mechanisms to react to new threats or SLAs violation. These mechanisms could include for instance activation or deactivation of policies, remediation actions and/or life migration tools.

DR7: Isolation

Due to the multi-provider and multi-tenancy nature of SUPERCLOUD architecture design (cf. D1.1 and D1.2), data, services and resources of different users can be hosted in in the same location. Such situations increases the risk of security breaches and exacerbates the feeling of distrust among users. The system should deploy strong and flexible isolation technologies to guarantee clouds integrity and confidentiality while allowing collaboration and resources sharing.

DR8: Location Awareness and Control

Compliance with legal requirements impose to some cloud customers to control the location of their data. To that aim, SUPERCLOUD must provide mechanisms for customers and users to control where their data and services are processed and stored at country and/or continent level. The control of location could not be complete without providing mechanisms that allow users/customers to monitor the current location and eventually any transfer of its data.

DR9: Monitoring, auditing and investigation

SUPERCLOUD self-management infrastructure need to provide a sound and trustworthy monitoring mechanisms that are able to collect all relevant information to detect and predict violation of active SLA. This calls also for a mechanisms to extract monitoring objectives from active SLAs. The mechanisms shall allow automatic conversion of Service Level Objectives into Monitoring, auditing and investigation objectives.

DR10: Assurance, Guarantees and Remediation

Use cases addressed in SUPERCLOUD include Health-care systems. In theses systems, the manipulated data is very sensitive which disclosure is governed by hard legal requirements. However, current SLA standards, protocols and frameworks do provide sufficient mechanisms to prevent and manage security and privacy violations.
The designed Security Service Level Agreement architecture shall provide adequate assurance and remediation mechanisms to incite provider to comply with legal texts.

DR11: Autonomic Security

Distributed cloud infrastructures take complexity to the next level compared to single clouds. Such complexity is in turn source of many vulnerabilities. Vertically, vulnerabilities across infrastructure layers add up to heterogeneity of defenses, which must now also be considered horizontally across cloud providers. Corresponding protection mechanisms only have partial view of threats, either in layers, or across providers[2]. To mitigate such complexity, security automation is required.
Autonomic security management has been gaining momentum as simpler, faster, and more flexible approach to respond to threats. Automation can cover different aspects such as *policy generation* and *policy execution and deployment*. It may also address detection and reaction to attacks without, or with a minimum of external intervention. SUPERCLOUD being both a multi-provider and multi-layered infrastructure, autonomic security management requires coordination of multiple security feedback loops according to both such dimensions, with seamless unification to elaborate the security response.

[2]For instance, SUPERCLOUD will leverage on heterogeneous network infrastructures coming from multiple providers. Each provider may have a partial view of network security incidents that may affect the SUPERCLOUD users, while an attack could leverage the same artifacts across providers. This requires a comprehensive approach for attack remediation to coordinate counter-measures across providers.

DR12: End-to-end Security

Security technology heterogeneity across layers and cloud providers hampers uniform security SLA guarantees. Therefore, the self-management framework should provide an end-to-end response from SLA specification to threat detection, elaboration of mitigation strategy, and enforcement. In addition, the infrastructure cannot be fully trusted, many layers being highly vulnerable. Multi-provider security service heterogeneity is also source of threats. Self-management should thus also manage trust to guarantee authenticity and integrity of the link between a VM and its cloud resources, across layers, provider domains, and between users, e.g., by composing chains of trust.

Summary of Security Requirements

Multi-clouds or distributed cloud infrastructure (DCI) face major security challenges that are slowing down its wide adoption by stakeholders. The requirements listed above are especially challenging how *security*, *privacy* and *trust* are managed in current cloud systems. The fulfilment of these requirements involve many disciplines, ranging from *Service Level Agreements* (SLA) to *Intrustion detection and prevension*.
In Table 2.1, we map the aforementioned requirements to the security related disciplines they are challenging. The focus have been put on six disciplines, namely *Service Level Agreement* (SLA) *Access Control* (AC), *Trust Management* (TM), *Cryptography* (CR), *Fault Tolerance* (FT) and *Intrusion detection and Prevension* (IDS/IPS). Of course, multi-clouds are challenging additional security related disciplines, but the focus is put in the next Chapter on these disciplines, as they play a central role in achieving *user-centric* and *self-managed* control of security advocated in the SUPERCLOUD project.

	SLA	AC	TM	CR	FT	IDS/IPS
DR1: Availability		✗		✗	✗	✗
DR2: Integrity	✗	✗		✗		✗
DR3: User-Centric Security Control	✗	✗	✗	✗	✗	✗
DR4: Fine-grained Access Control	✗	✗		✗		✗
DR5: Hor. and Ver. Privacy Preservation	✗		✗	✗		
DR6: Interoperability and Adaptiveness	✗	✗	✗	✗		✗
DR7: Isolation		✗	✗			
DR8: Location Awareness and Control		✗			✗	
DR9: Monitoring, auditing and investigation			✗	✗	✗	✗
DR10: Assurance, Guarantees and Remediation	✗		✗		✗	
DR11: Autonomic Security	✗			✗	✗	✗
DR12: End-to-end Security	✗			✗	✗	✗

Table 2.1: Initial design requirements for Self-Management of security in Multi-clouds.

- *Availbility* is already managed by current SLA technologies. However, when multiple and complex security mechanisms has to be executed within multi-clouds, this requirement can be hard to fulfil. Thus one of the main challenges in SUPERCLOUD would be to preserve *availbility* while applying in a decentralized way *access control*, *trust management*, *cryptography*, *fault tolerance* and *intrusion detection*.
- *Integrity* is a complex concept that needs to be well defined in SLAs. In addition, this requirement calls for new decentralised *access control* mechanisms that combines fine-grained *authorizitions*. For *Cryptography* and *Intrusion detection*, this requirement only exacerbates exiting challenges, notably with respect to the highly distributed and decentralised nature of multi-clouds.

- *User-Centric Security Control* will necessarily take into account the definition of user security needs. Thus the challenge for SLAs would be the ability to capture all users needs. Further, users will have a natural desire to manage security by themselves. This has two significant consequences: *Cloud providers may receive contradicting demands* and *Users will need tools that may interfere with the infrastructure.* This significantly impacts the traditional ways of enforcing security at all levels.
- *Fine-grained Access Control* has a strong impact on how *Access Control* is achieved in existing models. A more fine-grained approach to cover the different resources' abstractions at *compute*, *storage* and *network* levels is needed. This requirement has also many impacts on the definition of SLAs as a mean of capturing access control requirements, as well as on *cryptography* mechanisms to achieve *self-protected* data.
- *Horizontal and Vertical Privacy* has an important impact on how these aspects are specified in the SLAs, and how based on these SLAs, *access control* and *cryptography* mechanisms are coordinated to ensure *privacy* with respect to *providers* (vertical) and *customers* (horizontal) at the same time. The combination of both aspects of *privacy* preserving mechanisms is a challenge for both disciplines.
- *Interoperability and Adaptiveness* are complementary requirements. For SLAs, it is important to be able to make customers' and providers' specifications interoperable, and if not to envision the appropriate adaptation mechanisms. Similarly, *multi-cloud* will make several providers cooperate in order to execute the cloud services of a single client. If their *access control*, *Trust Management*, *Cryptography* and *Intrusion Detection* mechanisms are not interoperable, this objectives could not be guaranteed.
- *Isolation* can be achieved using *Access Control.* In the meanwhile, it can impede the proper enforcement of *Acess Control* mechanisms. So *Multi-clouds* raise the challenge of enabling *Access Control* and *Isolation* coexist in the same environment. *Isolation* is also challenging *Trust Management* as mechanisms for building *chains of Trust* has to be adapted in order to stablish trust across isolated domains.
- *Location Awareness and Control* imposes additional constraints and reduces possibilities to *Fault Tolerance* mechanisms as they have to guarantee the same availability level with less resources. Its is also a challenging issue for *Access Control* as appropriate schemes need to be enforced to permit data processing only in *allowed* locations.
- *Monitoring, auditing and investigation.* The distributed and decentralized nature of multi-clouds makes difficult the collection and processing of monitoring data. In addition, it is very hard to monitor security mechanisms. For instance, how to monitor *encryption* of data at rest or under processing.
- *Assurance, Guarantees and Remediation* This requirements is related to the previous one. It is very difficult to provide assurance and/or guarantees when we can hardly monitor and assess the respect of these guarantees.
- *Autonomic Security.* Current technologies provide limited automation capabilities. The challenge in SUPERCLOUD would be the integration of all aspects of security in a system that requires minimum intervention of humans.
- *End-to-end Security* calls for a seamless coordination of all disciplines to overcome heterogeneity of security technologies across infrastructures and to manage trust relationships between different layers and across cloud providers. This unified user experience of security has not been achieved yet because considered as *utopic* in many scenarios. But in SUPERCLOUD, this objective reveal to be a necessity more than a desire.

Background

In this chapter, we provide a background on basic aspects of security disciplines that need to be developed to achieve *Self-Management* of *user-centric* Security. We will also review state-of-the-art looking for candidate solutions to address the requirements presented in the previous chapter.

Concretly, we will see in Section 3.1 how the consideration of security issues, and thus almost all the design requirements, are still at their infancy in *Service Level Agreement* mechanisms. In Section 3.2, we review existing models to select the one that best fits *user-centric* (DR3) *fine-grained* (DR4) and *interoperable* (DR6) requirements. Similarly, in Section 3.1 we will survey existing *Trut Management* mechanisms. With respect to *Cryptography* and *Fault Tolerance* we rely on the Deliverable D3.1 wherein an extended review of the literature has been provided. Likewise, *Intrusion Detection and Prevention systems* has been presented in the Deliverable 4.1.

Introduction to Service Level Agreement

In cloud computing, as in any other service oriented model, Service Level Agreements (SLAs) are becoming progressively more common way to manage the quality of services with and within organizations. In this Section, we review existing literatures on Service Level Agreement technologies. We split down this section into three sub-sections. In subsection 3.1.1, we present basic concepts that constitute the foundations of Service Level Agreement Technologies, then in Section 3.1.2 we present dominant SLA standards, languages and frameworks. Finally, in Section 3.1.4 we present what have been done up to know in the consideration of security requirements in SLAs and highlight the challenges facing this objective.

Foundations of SLAs

Given the diversity of disciplines using SLAs and the numerous interpretations that have been proposed in recent years, we propose to start by presenting essential concepts that are necessary to understand this review and later the SLA management architecture we propose in the SUPERCLOUD project.

Service Level Objectives

A Service Level Agreement outlines a specific service quality commitment between SLA actors. For instance, in the SUPERCLOUD SLAs can be a Cloud Service Provider and a Cloud Service Customer. The SLA includes languages describing expected service quality in terms of technical and non-technical metrics. These individual metrics represent the Service Level Objectives (SLO), called also terms. From our literature review, we identify two types of Service Level Objectives, *Performance Level Objectives* and *Security Level Objectives*.

Performance Level Objectives reflect traditional Quality of Service metrics which have been extensively investigated in the last decades. For illustration purpose, we provide hereafter some examples of *Performance Level Objectives*.

1. Availability
2. Capacity
3. Response Time
4. Capability
5. Support
6. Reversibility

Security Level Objectives reflect the Quality of Protection [49] offered by the CSP. With respect to that, Bernsmed and colleagues [17] proposed a framework for security in Service Level Agreements for Cloud computing. In this work, the authors classified Security SLO into five categories; Secure resource pooling, Secure elasticity, Access control, Audit, verification and compliance and incident management & response.

We present here after an extended list of Security Service Level Objectives, which most of them originate from [33]:

1. Reliability
2. Authentication
3. Authorization
4. Encryption
5. Intrusion Detection
6. Incidents Management
7. Logging and Monitoring
8. Auditing and Security verification
9. Vulnerability Management
10. Data classification
11. Fault Tolerance mechanisms (mirroring, backup and Restore)
12. Data lifecycle Management
13. Data locality awareness and management
14. Data portability
15. Purpose specification
16. Data minimization
17. User-controlled use, retention and disclosure of data
18. Transparency
19. Accountability
20. Intervenability

We refer the reader to the guidelines [33] for a detailed description of each Service Level Objective.

Specification

SLA specification details the strategy and processes of specifying the agreement terms in a specific format. The specification implies the use of specification languages such as *WSLA, WS-Agreement, or SLAng* (cf. Sections 3.1.2, 3.1.2 and 3.1.3).

Service Level Templates

SLA templates are generally used by Service Provider to advertise the type of offers they are willing to offer.

Service Level Offer

An SLA offer represent an adaptation of an SLA template to meet the customer specific requirements. The SLA offer can also be built from scratch to allow customer to express freely his requirements without being burdened by a particular SLA Template.

Service Level Negotiation and re-negotiation

The process of negotiation tries to ensure an optimum service provision arrangements. All SLA aspects should be negotiated including responsibilities and penalties. With respect to Hiles [58], SLA negotiation facilitates the growing understanding of requirements and the constraints of entities involved in the SLA process (i.e. Providers and Customers).

Service Level Agreement

Negotiation should result in an agreement. An agreement is a description of the expected service quality delivered by a provider to a customer. The service quality is expressed in terms of technical and non-technical parameters and the related metrics with which provision of theses requirements is being measured [38].

Service Level Execution

Service Level Execution consist in converting Service Level Objective specified in SLAs into configuration parameters to be enforced in order to make the agreement effective. The execution of SLAs often involve an SLA execution manager that handles the extraction of SLO and their conversion into operational commands.
The execution of an SLA consist also in developing monitoring techniques that guarantee the appropriate enforcement of SLA greements. This aspect will be detailed hereafter.

Service Level Monitoring

The secret of a successful service level agreement depends greatly on the ability of the provider to provide the customer with comprehensive and accurate mechanisms to measure the performance of the service his is using [105]. So Monitoring calls and motivates the needs for adequate *trust management mechanisms* as this process relies on credible and reliable information that the provider can offer to its customers. The more monitoring mechanisms are accurate the more the provider will gain credibility, and hence trust with respect to its customers.
Service Level Monitoring means also that Service Level objectives must be meaningful, measurable and monitored constantly. In this process, the current levels of service are to be regularly compared with agreed levels.

Service Level Arbitration

In case of discrepancy between monitored service levels and agreed service levels both Customers and Providers are expected to identify and resolve the reason(s) for disputes. If the problem is not resolved, dispute resolution may involve the implication of an arbitrator. An arbitrator is a neutral third party that tries to identify the responsibilities of SLA conflicts.

Rewards and Sanctions

The detection and resolution of a conflict may include the application of a reward and/or a sanction. The reward could represent a compensation to the entity that suffered from the SLA violation, whereas Sanctions and penalties can be applied to the responsible of these violations. Rewards and Sanctions may include reduced or increased resource allocations or the ability to retain income.

SLA Lifecycle

A clearly defined lifecycle is essential for effective realisation of an SLA [120]. An early research [95], on SLAs defined SLAs lifecycle in three high level phases, which are the *creation phase*, *operation phase*, and *removal phase*. In the same period, Sun Microsystems Internet Data Center Group [121] defines

a practical SLA lifecycle in six steps, which are "discover service providers", "define SLA", "establish agreement", "monitor SLA violation", "terminate SLA", and "enforce penalties for violation".

More recently, the Telemanagement Forum [47] identified in their Handbook Solution Suite a six phases SLA lifecycle, these cycles include *template development*, *negotiation*, *preparation*, *execution*, *assessement*, and *termination and decomission*. Even though the lifecycle description was originally destined to telecommunications industry, its generic nature open doors to its adoption in other disciplines. Especially because of the explicit mention to negotiation phase and an important step priore to SLAs execution. In 2011, Bernsmed and colleagues [16] defined a similar SLA lifecycle in six phases; "publishing", "negotiation", "commitment", "provisioning", "monitoring" and "termination".

After this short review of existing SLA lifecycle approaches, we believe that 3 steps[1] are clearly not sufficient to cover all steps of an SLA process. At the opposite, it appears that their is currently an agreement on the minimums steps required to establish an SLA that are six steps. Even if their names differ, we believe that the authors refer to more or less the same phases. Finally, the six steps SLA lifecycle is more reasonable and provides detailed fine grain information, because it includes important processes, such as re/negotiation and violation control.

Short Survey on SLA Languages, Standards and Frameworks

In this section, we provide a short literature review on existing SLA languages and frameworks.

WSLA

Proposed by IBM in 2001, Web Service Level Agreement (WSLA) [68] allow the specification, enforcement and monitoring of SLAs. WSLA language is based on XML and (defined as an XML schema) and allows the creation of machine-readable SLAs in Web Services environments.However, the WSLA language is extensible to deal with other service-based technologies and other technical fields such as Cloud computing or any other inter-domain agreements management. The WSLA framework comprises several monitoring services that can be replaced with services developed by third parties.

An SLA created using the WSLA language typically contains the following sections: a description of the parties and the interfaces they expose to the other parties, a definition of the services and their operations, obligations of the agreement, action guarantees, and other information such as pricing or penalties.

WS-Agreement

WS-Agreement [8] is a specification, a language and a protocol developed by the Open Grid Forum (OGF)[2]. WS-Agreement defines an XML-based language for specifying agreements as well as a protocol for advertising offers (i.e., service capabilities) of providers. The specification defines also a monitoring compliance mechanism.
WS-Agreement agreement specification is represented as an XML Scheme that defines the overall structure of an agreement. Moreover, the protocol allows automatic negotiation and SLA establishment. The specification consists of three parts which may be used in a composable manner: a schema for specifying an agreement, a schema for specifying an agreement template, and a set of port types and operations for managing agreement life-cycle, including creation, expiration, and monitoring of agreement states.
Unlike WSLA, WS-Agreement language is quit extensible and allow the definition of domain-specific service level objectives. In WS-Agreement, Templates and offers are created to embody customizable

[1] such as defined by Ron and Alika [95]
[2] `http://cloudindustryforum.org`

aspects of an agreement. However, in WS-Agreement it is not possible to specify metrics to be associated with the parameters that specifies SLOs in the agreement.

WS-Re2Policy

Web service requirements and reactions policy language (WS-Re2Policy) [93] is a policy language for distributed SLA monitoring and enforcement. WS-Re2Policy language is based on Event-Condition-Action (ECA) rules paradigm to specify requirements and reactions in a single policy. In addition, a supportive architecture to implement WS-Re2Policy in the areas of Web services and SOAs is proposed. The WS-Re2Policy language was designed as an extension to the World Wide Web Consortiums WS-Policy framework. The language is compliant to WS-Policy and can be extended by other WS-Policy compliant languages like WS- SecurityPolicy.

RBSLA

Rule-Based Service Level Agreements (RBSLA) [91] is a project that uses knowledge representation concepts for the specification of SLAs. It provides a set of abstract language constructs (based on RuleML) represent, manage, enforce and automatically monitor Service Level Agreements. The rule based SLA approach consists of three layers: (i) *The logical core*, an expressive KR combining several logical formalisms, (ii) A declarative, *high-level Rule Based SLA* language (RBSLA) extending RuleML in order to address interoperability with other rule languages and tool support, and (iii) *The Rule Based Service Level Management tool* is divided into two partitions: the Contract Manager (CM) and the Service Dashboard (SD). The CM is used to manage, write, maintain and update SLA rules and supports different roles such as the (rule) expert or the business practitioner. The SD visualizes the monitoring and enforcement process in the contract life cycle and supports further SLM processes.

SLAng

SLAng [103] is an XML based service agreement language. SLAng defines seven different types of SLA (i.e., Application, Hosting, Persistence, Communication, Service, Container and Networking). These types are used to regulate the possible agreement between the different types of parties involved in the agreement (e.g., application, web service, component, container, storage and network). The authors differentiate between *horizontal* and *vertical* SLA. *Horizontal SLAs* are contracted between different parties providing the same kind of service. For example, two VMs can collaborate for replication purpose. *Vertical SLAs* regulates the support parties get from their underlying infrastructure.

CSLA

Cloud Service Level Agreement language (CSLA) is a language that allows to specify SLAs in any language for any cloud service (XaaS) [71, 72]. CSLA addresses intrinsically (i) QoS uncertainty in unpredictable and dynamic environment and (ii) the cost model of Cloud computing. CSLA is based on the Open Cloud Computing Interface (OCCI) and the Cloud Computing Reference Architecture of the National Institute of Standards and Technology (NIST).

Security in Service Level Agreement

Even tough traditional Quality of service objectives that focus on performance are known to be key issues, all recent surveys on the adoption of cloud technology highlight *trust* and *security* as the main barrier for Customers. Thus, is appears to be natural that Service Level Agreements should explicitly state the obligation of the providers in terms of implemented security mechanisms, their effectiveness and the implication of possible mismanagement [16]. The ability of a Provider to deliver Cloud services that comply with a set of security and legal requirements is called *Quality of Protection* [16, 17, 39].

Security SLAs tries to address questions such as *How to allocate cloud resources according to security requirements* and *how to check compliance and detect eventual violations to theses requirements.* There have been some initiatives that consider security aspects in SLAs. In this section, we review this works and try to identify main obstacles towards the achievement of this objective.

Initiatives

The Cloud Security Alliance [7] is a non profit organization that aims at promoting the use of best practice to increase the security level of Cloud infrastructures. The most relevant initiative of CSA has been the Cloud Assessment Initiative Questionnaire (CAIQ). The questionnaire destined to Cloud Providers to document the implemented security measures. This questionnaire will help Cloud Services Customers to understand security coverage and guarantees of cloud offers. CSA proposed also the Cloud Controls Matrix (CCM). This matrix is used by Cloud Customers to evaluate the risk implied in leveraging a particular cloud service provider.

In Cloud Security Level Agreements (SecLA) [83], the authors proposed a method to benchmark (both quantitatively and qualitatively) Cloud Security SLAs of one or several Providers with respect to the Customer requirements. Both Providers and Customers SLAs are expressed using the SecLA language. The authors used Quantitative Policy Tress (QPT) as data structure to represent and reason on Security Level Agreements.

In [17], the authors provided a framework for security in SLAs for Cloud computing. The objective of the framework twofold; to help potential Cloud customers to identify necessary protection mechanisms and, in the next step, to facilitate automatic service composition based on a set of predefined security requirements.

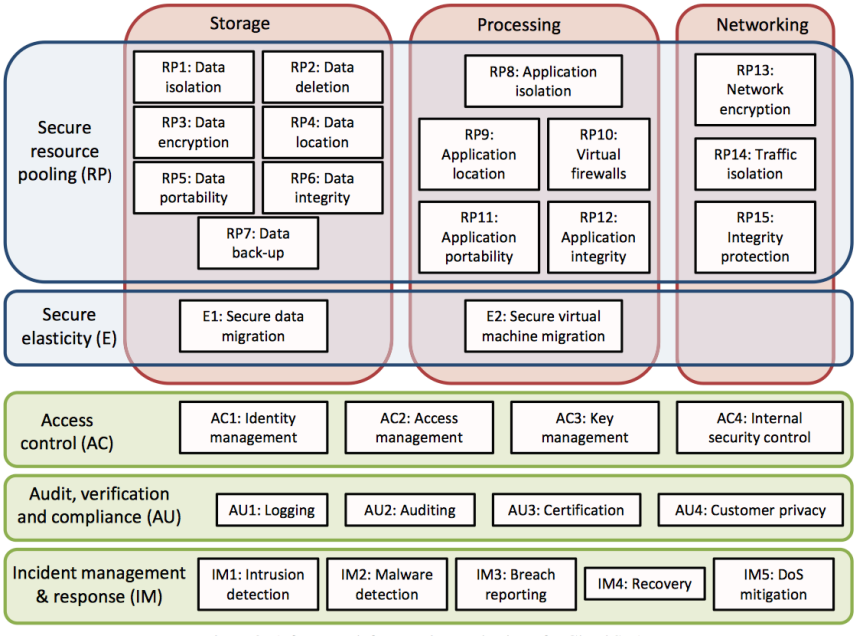

Figure 3.1: SLA metrics with respect to Security Objectives [17, 16]

In [16], the same authors build on this framework and described how to manage the security SLA lifecycle with the aid of a framework for security mechanisms as input to contract requirements. The purpose of their proposal was to facilitate rapid service compositions and agreements for hybrid Clouds based on the necessary security requirements and establish trust between the customer and provider.

In 2014, the European Commission published standardization guidelines for cloud computing service level agreements (SLAs) between cloud providers and cloud service customers [33]. The guidelines provide general recommendations to Cloud Customers and Providers about what they could agree on using SLAs. Representatives from companies including Amazon, Google, IBM, Microsoft, SAP and Salesforce helped develop the 41 pages guidelines within a sub-group of the Cloud Select Industry Group (CSIG). CSIG was set up by the Commission with the aim of developing standardised cloud computing contract terms that could be used by businesses hoping to enter into outsourcing agreements with cloud providers.

Chen-Yu Lee and colleagues described in [79] and Ontology for representing security SLAs (SSLA). The proposed Ontology can be used to understand the security agreements of a provider, to negotiate desired security levels, and to audit the compliance of a provider with respect to federal regulations such as HIPAA standards.

Finally,the International Organization for Standardization (ISO) released in 2015 a standard to offer information security advice for both cloud service customers and cloud service providers, offering guidance on the information security elements of cloud computing, recommending and assisting with the implementation of cloud-specific information security controls.

Challenges

The lack of common standard vocabulary for both security and Cloud concepts has been up to now the principal limit for the adoption of security SLAs in Cloud computing. SLA-Aware Cloud computing is a new architectural design that involves a high degree of self-management. This architectural pattern requires complex and intense performance, security, automation and adaptation mechanisms.

The main challenges to be addressed within the SUPERCLOUD project include the enhancement and/or extension of existing SLA standards and frameworks to leverage Security requirements. A second challenge consist in designing comprehensive and efficient negotiation strategies to enable end-to-end Service Level Agreements. Theses strategies should take into account the critical aspect of security requirements when achieving negotiations. Indeed, negotiation strategies tries often to reach a trade-off that maximizes the satisfaction of both CSP and CSC but making concessions in security requirements should be managed in a different way.

Finally, we need also to build appropriate monitoring and feedback mechanisms to observe the commitments met by an SLA, and the development of adaption strategies to mitigate the effects of possible SLA violation.

Introduction to Authorization and Access Control

Access control, more precisely authorization, is a basic and critical mechanism often used for operating systems. Usually presenting as a software module, access control provides a control solution for some entities (called *subjects*) to access some other entities (called *objects*) through some actions in the system.

In the context of cloud computing, subjects are cloud users, objects are cloud resources and actions are different ways of accessing resources. A cloud user is a digital embodiment of a human, system or service who consumes cloud resources. A cloud resource is a virtual component of limited availability within the cloud infrastructure, such as an instance, a volume, a public IP address, etc.

A cloud platform can include an authorization module to control the access to cloud resources. An authorization module is usually composed of a customizable access control policy/model and a corresponding implementation. Even in a single cloud platform, various authorization mechanisms might be applied for different cloud resources. For example, remote access to a VM is controlled by an SSH server in the VM; the network-level access is configured in various firewalls and gateways. It is an open question whether a single authorization module is needed for a cloud platform to coordinate the access to all cloud resources.

Currently elaborated access control models notably include Access Control Lists (ACL) [100], Role-Based Access Control (RBAC) [101], Organization-Based Access Control (OrBAC) [66], Attribute-Based Access Control (ABAC) [131, 77].

In this section, we will present the different models and language that allow to design and specify authorization and access control policies. The objective of this section is twofold; first, we provide the reader basic concepts that we build upon in the specification of *authorization* management mechanisms within the *Self-Management* framework (cf. Chapter 4). Second, we exacerbate the drawbacks and benefits of each approach in order be able to choose the one that best fits SUPERCLOUD requirements (cf. Chapter 2) with respect to *Authorisation* and *Access Control*.

Mandatory Access Control

Mandatory Access Control (MAC) is a centralized solution. Only the administrator controls access to any object. It provides a security level to the user and for the object (classification). Then, authorization is a relation between these two levels. For example, the common government classifications of their objects are: unclassified, confidential, secret, and top secret []. The subject will grant access to the object only if the security level of the subject is higher than the security level of the object. This model is very used by military and government organizations.

Discretionary Access Control

Discretionary access control (DAC) is one of the most widespread access control models. It is a decentralized solution. Each object is controlled by its owner. An object has an owner and this owner can allow subjects (or users) to access to this object or not. Any rules is written as (S, A, O) which means that the subject S can do the action A on an object O. These two models are static and have a low level of abstraction. As a result, it is difficult to administrate and it needs a lot of memories when we have many users.

Role-Based Access Control

Role-Based Access Control (RBAC) [101] is widely used. A number of concepts are defined in this model such as role, hierarchy, separation of duties and session. Role is a set of users which can be

assigned to the same security rules. For example: Fredric, Robert, Matthieu are doctors in the hospital of EVRY; they will be able to execute same actions on any resources. Consequently, it will be simpler to assign rules to the role doctor and not to each user. Moreover, it is possible to assign several roles (doctor, head of department) to one subject. This model offers a higher level of abstraction than MAC and DAC. RBAC uses also the concept of hierarchy which is a relation between two roles. It offers the possibility of rules derivation. A role A inherits from role B means that permission given to the role B is automatically assigned to the role A. RBAC defines two types of hierarchy:

- Organizational hierarchy is a relationship between an expert and a super-expert in the same branch. The super expert can control and evaluate the tasks of the expert. RBAC offers the possibility to inherit the expert permissions e.g., head of department and employee.
- Specification /Generalization hierarchy is defined between roles A and another role B if this latter can do all actions of A. For instance, role A can be doctor and role B can be a cardiologist. This notion facilitates the design and the management of the policy security.

RBAC also proposes two types of duties separation concept:

- Static Separation of Duties (SSD) ensures the assignment inability of two opposite roles to the same user.
- Dynamic Separation of Duties (DSD) offers the possibility to have two contradictory roles for one user. However, they cannot be activated simultaneously.

Moreover, the concept of session is introduced in RBAC. A session is a unique context associated with a user, within which the user activates a subset of assigned roles. Consequently, every activated role belongs to one session, and each session belongs to a unique user. Moreover, RBAC has an administration model which is ARBAC. The RBAC Model is presented in figure 3.2 where UA is user assignment and PA is permission assignment.

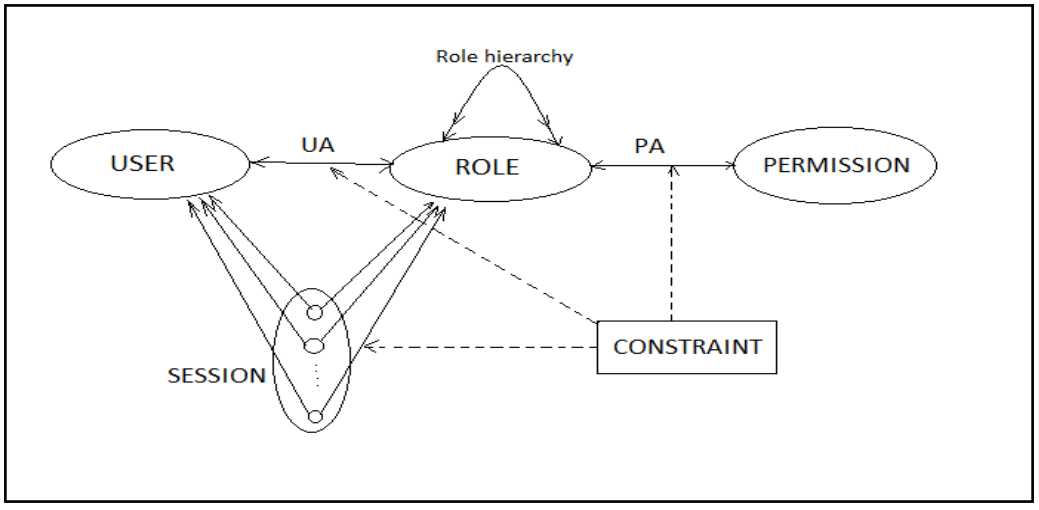

Figure 3.2: RBAC model

To resume, RBAC has a higher level of abstraction than MAC and DAC. Administration also is simpler (it has also an administration model ARBAC). Although, the concept SSD and DSD offer a possibility to create a dynamic policy, RBAC does not introduce solutions to interact with the environment. Finally, it does not address to the problem of collaboration. To ameliorate RBAC, various models are proposed. In the following, we detail some of them.

Temporal Role Based Access Control and Geographic Role Based Access Control

Temporal Role Based Access Control (TRBAC) and Geographic RBAC (Geo-RBAC) aim to ameliorate the dynamism of RBAC. For a same user, access to the resource can be authorized or denied in function of time condition in TRBAC and location condition in Geo-BAC. Note also that another model is designed based on TRBAC which is GTRBAC in order to solve possible conflicts in the policy. However, these solutions have not an administration model.

Team-based Access Control

Team-based Access Control (TMAC) [114] is also based on RBAC. It defines new notions to improve the collaboration criteria of RBAC.It represents a team work. This model defines a way to identify roles which contribute in a team. User will be assigned to a team according to its role. Teams interact to accomplish a mission. However, this model suffers from several problems. First, it deals with the problem of a local collaboration; the team definition can be used in only one system. Secondly, writing rules for a team and a role may create conflicts (permission assigned to a role which cannot be authorized in its team). Resolving this type of conflict is not clear in TMAC. TMAC extension, which is CTMAC, is proposed in order to interact with environment. Time, location and other contexts are used in CTMAC. These models offer a solution to achieve a mission. They do not address the general problem which is the collaboration between different entities. Administration in these models will be harder than RBAC since they have not an administration model.

Coalition Based Access Control

Coalition Based Access Control (CBAC) [32] incorporates aspects of RBAC and TCAC: team, task and role. It also defines the notion of organization to address the issue of collaboration between two or more systems. CBAC defines two new notions Organization; Coalition (OC) and mission. OC is formed by a set of organizations and mission is the task that needs to be done by an organization. A relationship must be defined between OC and missions. However, with these different new concepts, its management is complicated and hard since it has not an administration model.

Attribute Based Access Control (ABAC)

Attribute-Based Access Control (ABAC) [131, 77] is an authorization model that provides dynamic and context-aware access control. Rules specify conditions under which access is granted or denied. ABAC uses attributes as building blocks in a structured language that defines access control rules and describes access requests. In ABAC, access decisions are based on attributes of the requester and resource, and users need not be known by the resource before sending a request. Attributes are sets of labels or properties that can be used to describe all the entities that must be considered for authorization purposes. Usually, each attribute consists of a key-value pair. Each entity has attributes that define the identity and characteristics of the corresponding entity. Some of the most relevant attributes that can be described are:

- Attributes of the resource: may include resource name or the identifier.
- Attributes of the requester: may contain the identifier, the name or the organization.
- Attributes of the service: may include the service name and address
- Attributes of an action: can be an action name.
- Attributes of the environment: may be the current date or time.

This approach might be more flexible than RBAC because it does not require separate roles for relevant sets of subject attributes, and rules can be implemented quickly to accommodate changing

needs. However, although ABAC is easy to set up, analyzing or changing user permissions can be problematic because of the complexity of cases that could be considered in the system. i.e there could be many combinations in the rule conditions.

Semantic-Web based Access Control

The Common Information Model (CIM) [26] created by the DMTF provides a common definition of management information for systems, networks, applications and services designed also for vendor extensions. CIM's common definitions allow to describe information systems in an implementation independent way, enabling vendors to exchange semantically rich management information between systems throughout the network. XML-encoded specifications do not embody the constructs for facilitating tasks like parsing, logical deduction or semantic interpretation. Semantically-rich policies introduce ontology based representation that permits such tasks. Adopting a formal representation of CIM, several reasoning processes become available in order to check, validate, simulate or detect conflicts in information systems. Using OWL [55] as language to represent CIM, the semantic of the Description Logic (DL) can be applied in the reasoning processes. DL is closely related to Semantic Web technologies, in which information is constantly being discovered, modified and updated. The CIM model is independent of any encoding and it can be represented on different languages including, from XML to RDF and from MOF to OWL. Thus, a Semantic-Web based access control approach can use the CIM represented in OWL as ontology, and Semantic Web Rule Language (SWRL) [56] to express the rules which define the behavior of the system. The combination of OWL ontologies and the SWRL to specify policy rules offers users the advantage of allowing automated reasoning. This is carried out by what is called a reasoner, referring to a specific piece of software which performs reasoning processes. Jena or Pellet are example of this kind of reasoners. These processes constitute a remarkable added value of the usage of Semantic Web technologies, since they are able to infer new knowledge, that is, deriving additional information not explicitly specified in the ontology. Moreover, they also perform a formal validation and verification of the domain constraints which are specified in the ontology to assure they are fulfilled. The representation of the policy and domain information used in this authorization system is based on these Semantic Web concepts and technologies, providing the following set of features as added value for the policy language:

- The use of an ontology based representation of CIM model to describe a management system. It provides a solution that facilitates the system management tasks and related reasoning processes about policies.
- Separation between domain description and policy description. This approach separates the concepts that are necessary to describe the domain to be protected and the rules which use such concepts to create policies expressing the desired security for the administered services. The separation of domain description and policy description permits to manage both specifications individually using different techniques.
- Reasoning capabilities about domain descriptions. The management representations are done in the form of an ontology, allowing reasoning processes to check constraints and query the information. This CIM representation and its OWL encoding incorporate semantic expressiveness into the management information specifications to ease the tasks of validating and reasoning about the policies which definitely help in handling the security management complex tasks like conflict detection and resolution. This is due to the fact that OWL is based on description logics. This simple and yet powerful kind of first order-like logic allows to perform reasoning tasks not only on individuals but also on the structure of the base information model which holds the instances.
- Reasoning capabilities about policy description. Policy rules are created in the form of horn like rules, enabling reasoning capabilities for policy conflict analysis techniques. Policies are defined

by managers as if-then rules and they are encoded by Semantic Web Rule Language (SWRL) which extends the set of OWL axioms to include a high-level abstract syntax for Horn like rules that can be combined with an OWL knowledge base. A useful restriction in the form of the rules is to limit antecedent and consequent atoms to be named classes, where the classes are defined purely in OWL.

- Interoperability. The use of standards and ontological representation eases the interpretation and integration of the information. This enables advanced capabilities like concept alignment between heterogeneous domains, allowing the combination of concepts from the different domains of the organizations.

In CIM, the basic components of an authorization decision (or privilege) are subjects, actions, and targets. This access control approach allows RBAC and ABAC models since authorization decisions can be taken based on roles and attributes of any kind. The attribute limitation is on the ontology definition but CIM is able to modeling almost any IT system feature.
A sample authorization decision would be read as follows: it is permitted for subject(s) S to perform action(s) A in the target(s) T. To model the concepts that related to an authorization decision, this solution makes use of the set of CIM concepts depicted in figure 3.3.

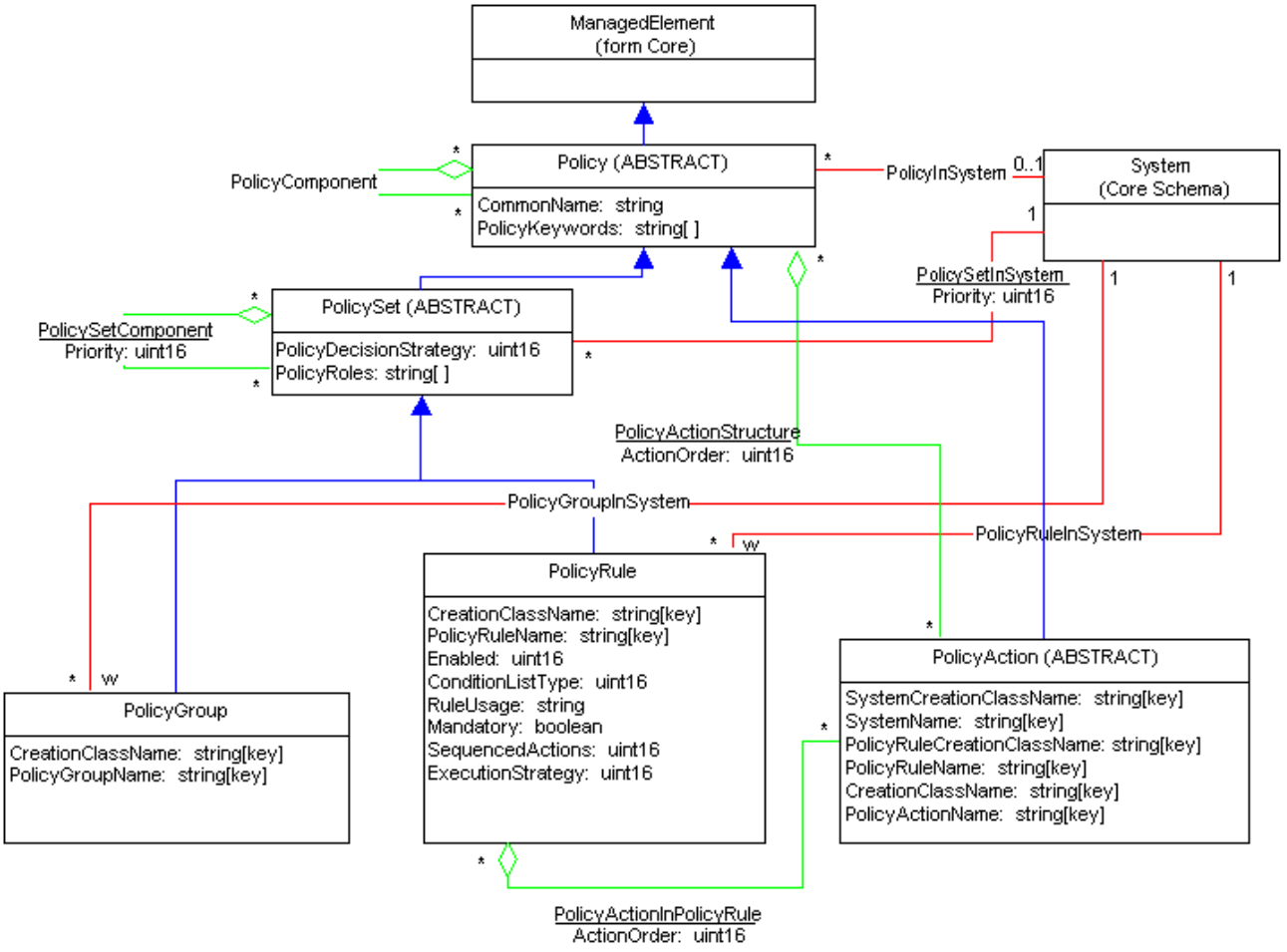

Figure 3.3: CIM Authorization model [26])

Privilege is the base concept for all types of activities which are granted or denied to a subject for a target. AuthorizedPrivilege is the specific subclass defining current privileges which result of applying the authorization policy rules. The association of subjects to AuhorizedPrivilege is accomplished

via the authorizedSubject association. The entities that are protected (i.e. targets) can be similarly defined via the authorizedTarget association.

To illustrate these concepts, let us consider an academic organization in which some exams are done with computers. But these exams can only be done in computers belonging to a controlled network whose computers are inside invigilated zones to avoid students cheating the exams. The domain description for this example can be represented using a CIM based ontology and it may be composed by a role that represent the set of students [Student], the controlled examination network [ExamNetwork], the examination service [ExamService], the privilege to access the exam [AccessExam] and finally the subject [JohnDoe].

This semantic web authorization mechanism has been deployed twice as proof of concept Access Control services. In [113], authors have adopted this approach enabling the dynamic management of authorization by combining the Semantic Web technologies with Grid authorization systems. The authorization service acts as a Policy Decision Point (PDP) which take authorization decisions based on the reasoning over the policies. Likewise, a similar deployment of the semantic-aware access control system has been applied in Cloud Computing environments.

However, when it comes to performance, the semantic-web based approach still lacks of enough efficiency in some use cases. This is mainly because of the reasoning time required by the reasoner engine in a "live" inference scenario like the ones proposed in SUPERCLOUD project. The complexity of the ontology and the size of the knowledge base could lead to an unfeasible reasoning time. Moreover, in some particular scenarios where the computing and consumption requirements are quite limited by the hardware, a semantic-web access control system couldn't be the most advisable solution. Thus, for the SUPERCLOUD project, it could be considered the usage of this access control approach but employing a more lightweight rule-based system instead of a semantic web engine based on OWL and SWRL.

Based on CIM concepts, a security policy language (SPL) [94] has been defined. It is a high-level language defined in XML aimed to define the desired security behavior of the networked systems and applications.

It was devised in the bounds of the POSITIF European project and later extended in DESEREC EU project. It is mainly based on the security policies, concepts and parameters already defined in the Common Information Model (CIM) [] from the DMTF, but in a high level fashion avoiding the complexity of CIM. It is defined in XML due to the ease with which its syntax and semantics can be extended and the widespread support that it enjoys from all the main platform and tool vendors. SPL supports different types of security policies and it allows grouping, priority and classify these policies. The SPL definitions are composed of a set of attributes and three groups of elements. The Common Elements, which represent common concepts, the Policies, which define the set of security policies, and the PolicyGroups that provide a mechanism to group and classify policies.

Organization-Based Access Control

Organization Based Access Control (OrBAC) [67, 65] is becoming largely used for modelling access control policies. It integrates various concepts defined in the previous work such as role, hierarchy, and context. Also OrBAC adds extension to enhance its use in a collaborative system.

The main concept of OrBAC is the entity organization. The policy specification is completely parameterized by the organization. This notion encourages researcher to handle simultaneously several security policies associated with different organizations. It is characterized by a high level of abstraction. Instead of modeling the policy by using the concrete and implementation-related concepts of

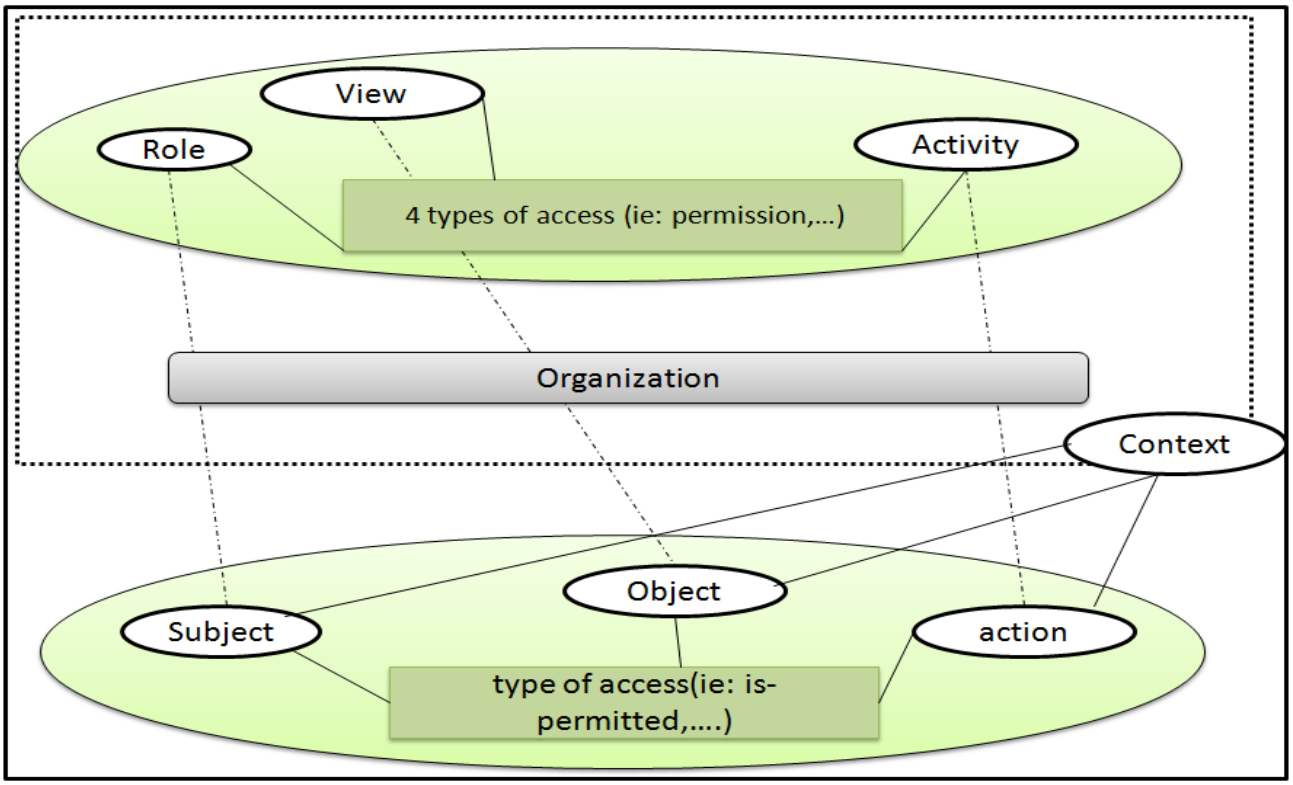

Figure 3.4: OrBAC model

subject, action and object, the OrBAC model suggests reasoning with the roles that subjects, actions or objects are assigned in the organization. Thus, a subject is abstracted into role which is a set of subjects to which the same security rule apply. Similarly, an activity and a view are respectively a set of actions and objects to which the same security rule apply.

Figure 3.4 describes the OrBAC model which introduces two security levels (concrete and abstract). OrBAC defines context concept. It is a condition that must be satisfied to activate a security rule. A mixed policy can be offered in OrBAC which define four types of access: Permission, prohibition, obligation and recommendation. Rules conflicts can appear in this policy. This problem may be resolved by affecting a coefficient to each rule. Several types of contexts can be used as temporal, geographical (physical and logical), pre-request, declared, etc. Also, we may have contexts which depend on the application. The hierarchy notion which facilitates the tasks of the administrator is also used in OrBAC. In the same way as RBAC, two type of hierarchy (specialization / generalization and organizational) are defined. Moreover, this hierarchy can be used between different roles, different views, different activity or different contexts.
The OrBAC model defines four predicates:

- empower : empower(s, r) means that subject s is empowered in role r.
- consider : consider(b, a) means that action b implements the activity a.
- use: use(o, v) means that object o is used in view v.
- hold: hold(s, b, o, c) means that context c is true between subject s, action b and object o

Access control rules are specified in OrBAC by quintuples that have the following form:

- SR(decision, role, activity, view, context)

which specifies that the decision (i.e. permission or prohibition) is applied to a given role when requesting to perform a given activity on a given view in a given context. We call these organizational security rules.
OrBAC has its administration model AdOrBAC. It uses the same logical formalism and the same concepts of ORBAC. As a result, OrBAC is a self-administrated model. Otherwise, a management and policy specification tool MotOrBAC [4] was implemented for OrBAC. It is designed to allow analyzing and simulating a security policy conforming to ORBAC. It also offers a conflict resolution and enables an administration of ORBAC (since it implements the AdOrBAC model). Policies created with MotOrBAC can be integrated into an application to secure it. MotOrBAC may easily extended by adding a specific plug-in.
To resume, ORBAC offers a generic and dynamic security policy with high level of abstraction. It reduces the cost of the administration of access control policies. The work in [] demonstrates that OrBAC leads to significant reduction of the management complexity. All of these features encourage researchers to propose interoperability security policy based on OrBAC. For these reasons, we will present in the next section some derivatives of OrBAC model that are used in similar environments as the SUPERCLOUD project.

Existing standards

eXtensible Access Control Markup Language (XACML)

The eXtensible Access Control Markup Language (XML) [59, 34] is an access control policy language specified by the Organization for the Advancement of Structured Information Standards (OASIS). It is based on XML and defines a common security policy language as well as the processing model that describes how to interpret the policies. It also includes a request/response protocol to express queries about whether a particular access should be allowed and describes answers to those queries.

The policy language provides a common means to express subject-target-action-condition access control policies. The main concept in XACML is the Policy, which represents the basis for an authorization decision. Policies are formed of Rules, which are the fundamental elements that can be evaluated. Moreover, policies can be also grouped in PolicySets. In order to know the policies that apply to a given request, Targets can be explicitly specified for Rules, Policies, and PolicySets. A Target defines the set of resources, subjects, and actions to which these will apply. Moreover, Policies and PolicySets may optionally be associated with Obligations. They contain a FulfillOn attribute that specifies whether they should be applied when the containing policy evaluates to Permit or Deny.

Another essential concept in XACML is Attributes. They are named values that describe properties of subjects, actions, resources, and the environment of the decision request. Attributes are typed and may contain multiple values. Examples of attributes are subject role memberships, subject email addresses and the time of day environment attribute. Attributes can be referenced by attribute designators or attribute selectors. The former specify a name and a type and can refer to subjects, resources, actions, and the environment of the request context, while the latter allow attribute lookup by an XPath query.

Regarding policy evaluation, the Rules of a policy have an associated effect, which defines the consequence of the rule (permit or deny) if it is evaluated to true. Rules may optionally contain a Condition, which consists in a Boolean expression that further limits the rule applicability. To decide the result of a composed element, Policy Combining Algorithms are used. They enable to determine the decision result of a Policy given the evaluation results of its rules or the decision result of a PolicySet given the evaluation results of its policies. XACML includes the following standard combining algorithms, although it allows users to define their own ones: Deny-overrides, Permit-overrides, First applicable and Only-one-applicable. An access control decision finally results in a value that can be Permit, Deny, NotApplicable and Indeterminate. The last two values are returned when an error occurred and

no decision can be made or when the request cannot be answered by the queried service, respectively.

In XACML based access control environments, it is often assumed that the following components are deployed:

- Policy Administration Point (PAP): manages and defines the policies that will apply.
- Policy Decision Point (PDP): evaluates and issues authorization decisions.
- Policy Enforcement Point (PEP): intercepts user access requests to a resource and enforces PDP decisions.
- Policy Information Point (PIP): provides external information to a PDP, such as LDAP attribute information.

The model also defines a Context Handler entity, which is the system entity that converts decision requests in the native request format of the access requester to the XACML format and also converts XACML authorization decisions to the native response format. In a typical XACML usage scenario, a subject (i.e. user or software application) wants to take some action on a particular target (i.e. resource). The subject submits its query to the entity managing the resource (PEP). It forms an XACML request message based on the attributes of the subject, action, target, and any other relevant information, and sends it to the policy decision point (PDP). The PDP analyzes the request and determines whether access should be granted or denied according to the XACML policies that are applicable to this request. Then, an XACML response message is returned to the PEP, which will then enforce the obligations and allow or deny access to the subject.

Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) [27] is an XML-based framework that enables the exchange of authentication, entitlement, and attribute information between entities. It is developed by the Security Services Technical Committee of the Organization for the Advancement of Structured Information Standards (OASIS). The specification defines different kinds of assertions, protocols and bindings to exchange this information. Assertions are statements issued by an entity that declares some authentication information or attributes of a given subject. SAML defines the following three types of assertions:

- Authentication Assertion: an assertion containing information about a user authentication, e.g., type of authentication, timestamp, username. Authentication Assertions are usually sent from an Authentication Authority at an Identity Provider to a Service Provider to prove the user identity.
- Attribute Assertion: this assertion is used to transfer a user attributes. SAML does not specify which attributes may be transferred. Attribute Assertions are usually sent from an Attribute Authority at an Identity Provider to a Service Provider to provide the user attribute information.
- Authorization Decision Assertion: an assertion to transfer authorization decision statements (e.g., permit, deny). Authorization Assertions, which are less used than Authentication and Attribute Assertions, are usually sent from the Policy Decision Point at a Service Provider to the accordant access control entity.

Besides the assertions, SAML also defines a set of protocols that specify the messages used to request and to send these assertions, as well as several bindings using standard communication protocols (e.g., SOAP, HTTP) used to transmit those protocol messages. SAML is currently mostly used for Single Sign-On and for attribute-based authorization. In these cases, authentication assertions are used to transfer authentication information from the authenticating entity (e.g., Identity Provider)

to the requesting entity (e.g., Service Provider) to avoid a new authentication of the user at the Service Provider. For attribute-based authorization, attribute assertions are used to send attribute information from an attribute authority, e.g., being part of an Identity Provider, to a deciding entity, providing the users attributes are stored centrally.

Single Sign-On (SSO)

There are several Identity Management solutions, standards and enabling technologies in the literature that support Single Sign-On (SSO) mechanisms in order to ease and improve the user interaction with the provided services.

Shibboleth [14] is a package of several supporting functions for Web Single Sign-On across or within organizational boundaries. It is based on the Security Assertion Markup Language (SAML) and represents one of the most used federation supporting packages in the academic context. A Shibboleth federation typically consists of Identity Providers (IdP), which host user data and Service Providers (SP), which offer restricted resources to users. Shibboleth supposes that users access the restricted resources by using a web browser and the infrastructure is built on the idea of a browser-based interaction between the federation parties. Although Shibboleth does not specify how the trust within the federation shall be established, it is usually statically established by signing contracts with the federation manager or with every participant. Moreover, participants share X.509 certificate information that is accessible in the federation meta-data. This meta-data should be updated regularly by the federation members to ensure security and its handling is part of the Shibboleth bundle.

The OAuth Authorization Framework [106] is an open framework that enables a user to authorize a client application to get limited access to some resources stored in a server without having to reveal his credentials. In OAuth, an authorization token is issued to the client application by an authorization server. This token is issued upon the approval of the resource owner and it provides limited access to the resource (e.g. specific lifetime). The client application uses this token to access the server storing the resources without presenting any user credential. The current version OAuth 2.0 is a proposed standard defined in IETF RFC 6749 and it is not backward compatible with OAuth 1.0. The importance of the OAuth protocol in protecting user's privacy is also confirmed by the attention it is drawing by the largest web companies like Facebook, Google, Yahoo, LinkedIn or Twitter.

OpenID [92] is a decentralized authentication specification that enables a user to be authenticated through an URL or XRI identifier by any other OpenID supported provider. With OpenID, the user can choose which provider to use and there is no central authority registering OpenID providers or services. The authentication protocol uses plain text key-value pairs and standard HTTP(S) request/responses. When a user wants to log into a service, it is requested for its OpenID identifier. An OpenID provider discovery process takes place, depending on the kind of identifier. Optionally, both the service and the OpenID provider can establish an association with a shared secret. The user is redirected to the provider, which authenticates the user and redirects him back to the original service with an authentication assertion. The OpenID technology is being widely used by companies like Google, Flickr, Yahoo, Facebook or VeriSign.

As part of Identity Management frameworks, supporting standards and enabling technologies are used to provide authentication and entity identification. Electronic Identity (eID) technologies are commonly used to identify users. eID Solutions vary in complexity, in ability to integrate with other systems, and in size of deployments. The primary objective of an eID system is to identify the user, to authorise and authenticate, and to be secure. National eID cards have been deployed in number of European countries, with Belgium eID being considered the most successful deployment to date. Public Key Infrastructures (PKI) [30] is also an enabling technology commonly used in Identity Management systems. A PKI is a system to issue, distribute and validate digital certificates, e.g., X.509 certificates.

These are digital certificates signed by a trusted authority that usually contains information about the identity of the bearer, its public key and further elements, such as identity of the issuer, validity, type of algorithm, etc.

WS-Federation [50] is an Identity Federation specification that allows different security realms to federate, such that authorized access to resources managed in one realm can be provided to security principals whose identities and attributes are managed in other realms. This includes mechanisms for brokering of identity, attribute, authentication and authorization assertions between realms, and privacy of federated claims. The WS-Security and WS-Trust specifications allow for different types of security tokens, infrastructures, and trust topologies. Specifically dealing with the issuing, renewing, and validating of security tokens, as well as with ways to establish, assess the presence of, and broker trust relationships between participants in a secure message exchange. WS-Federation uses these building blocks to define additional federation mechanisms that extend these specifications.

Apart from the aforementioned Identity Management approaches for Single Sign-On, there are commercial solutions available in the market like the Microsoft Windows Identity Foundation [97] or the Oracle Access Manager [12]. There also exist several European projects focused on Identity Management. SEMIRAMIS [] is a EU research project that aims to provide a secure and reliable solution for the management of personal information across multiple stakeholders. The project defines a pilot infrastructure that provides e-services related to identity management and secure personal attribute transfer. STORK [80] is another EU project that aims to establish a European eID Interoperability Platform that will allow citizens to establish new e-relations across borders, just by presenting their national eID.

Access control In cloud computing

In cloud computing, the relationship between *subjects* and *objects/resources* is very dynamic. In addition, *subjects* and *objects* can be in distinct security domains making the specification of *access control* policies a tedious task. Consequently, many of the traditional models that we have presented previously can not be used, or requires a big adaptation effort to be in compliance with the cloud specificity. For instance, models that heavily rely on Identity (e.g. IBAC) for the management of access control reveal to be inappropriate in Cloud as they are unable to scale. Furthermore, these models can be hardly used to manage access control in *open* environments where the identity of *authorized subjects* cannot be know beforehand [126].

In this Section, we briefly review some existing approaches that tackles the challenge of managing access control in cloud environments. Most of these approaches highlight the importance of managing multi-tenant and inter-tenant access control decisions.

For instance, in [76], a flexible attribute-based multi-policy access control (ABMAC) model has been proposed. The main idea of the ABMAC is to use multiple *autonomous* security policies, a policy per tenant/domain. Then *authorization* decisions are derived by combining individual decisions. The main benefit of this model lies in its capability to scale at very large levels.

The authors of [127] proposed Access Control for Cloud Computing (AC3). This model tries to facilitate the concepts of role and task that have been used in other models. In AC3, users are classified according to their actual jobs. Thus, users will be located on a security domain that relates to their role. Every role within the model will be assigned a set of the most relevant and needed tasks for achieving this role. This approach is very similar to the principals carried out in OrBAC with a subtle analogy between tasks and activities.

Calero et al. [6] introduce an authorization system enabling cross-tenant resource access based on

trust. Tang et al. [110, 111] further analyzes possible types of trust relations among tenants and propose a Multi-Tenant RBAC (MTRBAC) model for collaborative access control in a multi-tenant cloud [112].
In [108], the authors proposed a semantic access control model for cloud infrastructures. The system has been designed for healthcare systems and aims at providing an RBAC-like model based on Semantic Web Technologies, basically ontologies.
The above selection of cloud-specific access control models should be considered in complement to several more classical approaches that only use one of the aforementioned models in cloud systems. Unfortunately, none of the above mentioned work do consider cross-tenant interaction in policy administration level.

An Introduction to Trust Management

Trust has been extensively investigated in the last fifteen years which gave rise to numerous trust models. Most of these models have been developed in *distributed artificial intelligence* (DAI) or *security*. In the context of SUPERCLOUD project, there exists many possible configurations involving a *trustor* and a *trustee*. For instance, a provider represent the *trustee* with respect to customers that trust him for providing the service they agreed upon. Similarly, the provider can play the role of *trustor* if we consider the situations in which the activity of a customer can affect the performance of the provider, and hence affecting his reputation. There exists also a *trustor-trustee* relationship linking each component used in the project architecture. For example, the elements of the storage layers has to trust the VMs and other compute layer components (cross-layers trust). Horizontally, components from the same layer has to trust each others to build a chain of trust (CoT) to allow data flow across tenants, users and providers.

The objective of this section is not to review the complete literature on this subject, there exist several surveys that provided more comprehensive studies (c.f. [61, 52, 96, 109, 11, 73, 54]). Instead, we describe representative approaches and discuss there relevance to cloud computing. Further, this Section aims also at defining the concepts related to Trust Management in Computer Science and how it will be understood within the SUPERCLOUD project. For example, we will clarify the meanings of *Trustor* and *Trustee* used above, we will also give a precise definition of what is *trust* and what is *trust management*.

What is Trust?

In the real world as well as in the virtual ones, trust constitutes a fundamental concept for humans, without which they can neither act nor interact. So unsurprisingly, trust received in the last decades much attention from several disciplines. To that aim, it seems essential for the sake of clarification, the definition section by the disambiguation of its meaning.

For Gambetta [48], trust (or, symmetrically, distrust) is "particular level of the subjective probability with which an individual, A, expects that another individual, B, performs a given action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action". First, we notice here that Gambetta, which is a sociologist, conceptualized trust as a mathematical concept, making its definition more concrete. Also, the part "a particular level" of the definition means that for Gambetta, trust can be somehow quantifiable. For Gambetta, 0 means complete distrust and 1 full trust. Further, this definition makes explicit the idea that trust is subjective and introduces the specificity of trust: trust is made with respect to a specific action to be performed. This definition takes into account uncertainty induced by the behavior of the interacting partner, without which there would be no need for trust. Further, Gambetta states that "if we were blessed with an unlimited computational ability to map out all possible contingencies in enforceable contracts, trust would not be a problem". With this statement, Gambetta highlights the fact that trust involves decision in complex situations that are hard to grasp for human minds.

The complex nature of the situations involving trust is further reinforced by the definition provided by *Niklas Luhmann* [82]. For Luhmann "the complexity of the future world is reduced by the act of trust". Luhmann approaches trust from a sociological background as he relates the use of trust to interactions among societies and questions the existence of society without trust [75, 84]. He considers trust as one of the most important "internal mechanisms for the reduction of complexity". However, the authors defined complexity in very abstract terms even if generally he used it in reference to *uncertain* situations.

In computer science, McKnight [86] defined trust as "the extent to which one party is willing to depend

on something or somebody in a given situation with a feeling of relative security, even though negative consequences are possible". For Grandison and Sloman, trust is "the firm belief in the competence of an entity to act dependably, securely, and reliably within a specified context". Finally, for Castelfranchi, and Chaveny as well, trust is used as "the mental counterpart of delegation" [46]. So the authors consider delegation as an action taking this mental state as an input.

Each of the above definitions advanced our understanding of trust and provides important building bricks for the definition of this concept. However, none of them matches perfectly our conceptualization of trust. In the light of the above definitions, we define trust as:

Definition 1 (Trust) *the deliberate decision of an individual/entity A (called* trustor*) to be in a situation of vulnerability towards the behavior of an individual/entity B (called* trustee*) with respect to an issue X (i.e.* trust issue*) and within a context C.*

What is Trust Management?

Trust management has been defined by Blaze and colleagues as "a unified approach to specifying and interpreting security policies, credentials, relationships which allow direct authorisation of security-critical actions." [22, 20]. The main novelty of the approach introduced by Blaze *et al.* is that they unified the concepts of *security policy*, *credentials* and *authorisation* under the concept of *trust management*. However, their definition is too abstract and not very intuitive to explain what trust management really is.
For Jøsang, trust management is "the activity of collecting, codifying, analysing and presenting security relevant evidence with the purpose of making assessments and decisions regarding e-commerce transactions"[61, 62]. Although, broader and more intuitive, this definition was criticised by Grandison in his doctoral thesis to be too domain-specific (i.e. e-commerce) [54]. Nonetheless, Grandison reused it to define trust management as"the activity of collecting, encoding, analysing and presenting evidence relating to competence, honesty, security or dependability with the purpose of making assessments and decisions regarding trust relationships for Internet applications" [53, 54].
The main drawback in Grandison's definition is that the author restricted the nature of the evidences based on which the trust relationship can be established. Further, Grandison used the verb "collecting" for evidence while some of them can not collected but should be requested (e.g. credentials). Therefore, we prefer to adapt the above definitions to provide one that best matches our understanding of trust management.

Definition 2 (Trust Management) *The automated activity of collecting, requesting, providing and analysing information with the purpose of making trust decisions (e.g. access control, delegation, collaboration) based on policies [125].*

The main aspect we stress in this definition is the automated nature of trust management process. It is the automation requirement that makes the trust management issue complex and necessitates so much investigations. Further, we used the term information instead of evidence in order to comply with the models that have been proposed in the *Distributed Artificial Intelligence* domain. Finally, we generalize the purpose of trust management to trust decisions rather than focusing on trust relationships. From our perspective, a relationship implies some kind of continuation in time, while a trust decision better reflects the dynamic nature of trust.

Foundations of Trust Management

A Trust management System is an abstract system that processes a symbolic representation of trust relationship in the perspective of trust decision automation [124].
The "symbolic representation of trust" refers to the concepts of credentials and policies by means of which the issuer states that he trusts the entity to which the statement is applicable. Of course,

this trust is not generic, thus is most of the cases the statement concerns a specific issue (e.g. read a document). The symbolic representation of trust relationships can be best illustrated through the everyday ticket experience [119]. The ticket (let us say a tram ticket) can be considered as a symbol of trust between the tram company and the ticker holder. The ticket acts as a proof that the holder paid for the journey and consequently that he is entitled to get on the tram. Once bought, the ticket can be later given to someone else, thus transferring the trust relationship. In the tram, only the ticket will be verified and not the identity of the holder. Concretely, in the above example, the tram ticket illustrates the importance for credentials while the tram inspector enforces the policy (which is quite simple here).
Thus, a trust management system aims at linking the requester and the requested via a trust relationship based on which a trust decision can be made. To that aim, trust management systems provide a language for the specification of **policies** and **credentials**, and a **trust management engine** (trust engine for short) that evaluates whether the provided credentials satisfy the specified policy. These three components are presented in the following sections.

Credentials

Credentials (or digital credentials) represent the counterpart of the paper credential we use in the real world (e.g. passport, driving licence, student card). They represent digital documents or messages that are certified (i.e. signed) by *certification authorities*. They allow user authentication but can also provide additional information such as the user's attributes (cf. Section 3.2.4), memberships or rights. Blaze, in his trust management jargon, defined credential as "a signed message that allows a principal to delegate part of its own authority to perform actions to other principals". It is this definition that we used as a basis. For instance, a public key "certificate" is an example of a credential. Public key infrastructures (PKI) have been systemically used by trust management systems to create, distribute, validate, store and revoke credentials.

Policies

Policies have been extensively used in the computer science literature (e.g. information systems, security, multi-agent systems). Initially, policies have been introduced in computer science to automate tasks and decision makings (e.g. batch instructions). But nowadays, the main motivation for using policies is to make systems support dynamic and adaptive behaviour. Policies allows a system to change its behaviour without being stopped.
Despite their extensive use in the literature, the concept of policies is still hard to define and the provided definitions are either too generic or domain specific. For instance, Sloman defined policies as "rules governing the choices in behaviour of a system" [104]. While this definition captures the meaning of a general policy, it failed addressing its role which is to specify the circumstances under which the choices are made (in reaction to which conditions). Further, this definition reduces the form of a policy to a set of rules and thus excludes many of the approaches which do not rely on rules (cf. Section 3.4). Recently, De Coi and Olmedilla stated that "policies specify who is allowed to perform which action on which object depending on properties of the requester and of the object as well as parameters of the action and environmental factors" [40]. This definition makes explicit the OrBAC approach, and thus covers *de facto* IBAC, LBAC, RBAC and ABAC policies. However, this definition restricts the scope of a policy to situations in which an access request has to be evaluated. Consequently, this definition could not be used to describe situations in which a decision is not merely an access control decision (e.g. delegation). To avoid misunderstandings, we clarify the meaning of trust policies and define it as follows.

Definition 3 (Policy) *A policy is a statement that specifies under which conditions an entity (human or artificial) can be trusted for a specific issue (e.g. resource action, task delegation)*

A policy represents the expression of the conditions under which the individual A deliberately takes the decision to trust B. Thus from our perspective, the role of a policy is twofold: (i) it serves a means for A to express the policy that its trust management system will rely on, and (ii) it is used as a common language that A and B will use to exchange their respective trust conditions. In the light of that, the role of the policy specification language is paramount. The language provides the basic syntax to express conditions which represent the building blocks of a policy. Specification languages can be more or less verbal and can have solid or weak formal basis. Thus, depending on the nature of the policy specification language, policies fall into three categories: *informal*, *semi-formal*, and *formal*. We limit our attention to *formal* policies that can be understood by both artificial and human agents.

Trust Engine

The objective of the trust engine is to assess if the credentials provided by the requester are valid and whether they satisfy the specified policy. Importantly, trust management systems are not responsible for making trust decisions. It is always the human or the application using the TMS that decides whether to effectively trust the requester or not. So the main advantage in using a TMS is to offload applications of complex and tedious tasks that are *credentials verification* and *policies evaluation*. Figure 3.5 illustrates this principle and shows the basic functioning of a trust management system.

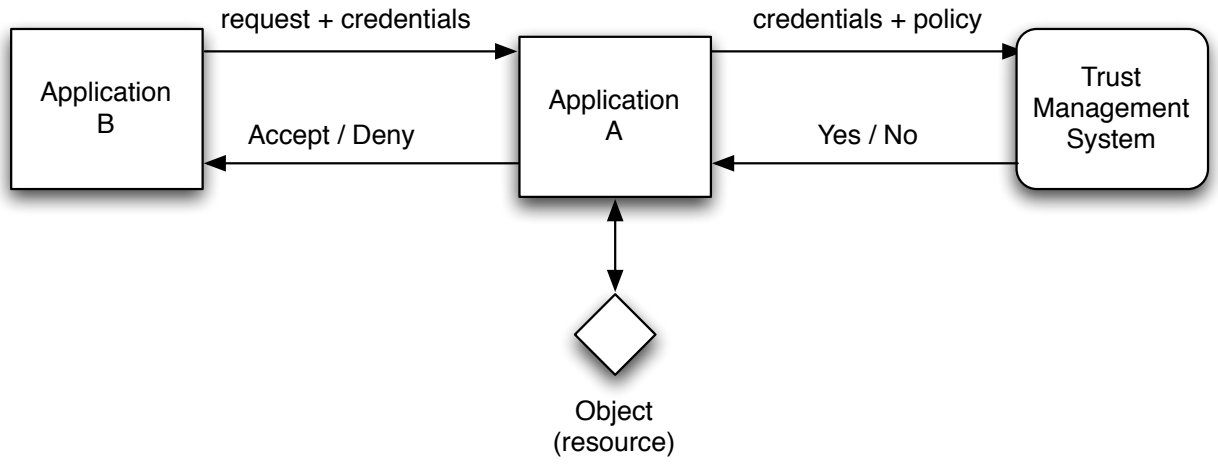

Figure 3.5: Illustration of the functioning of a trust management system

In Figure 3.5, the application A invokes the trust management system to determine whether application B can be allowed to perform an operation on a resource. To that aim, the application A provides the TMS with its local policy for the resource concerned by the request and the credentials provided by application B. The TMS produces an answer based on which the application A decides to allow or deny the request of B.
Depending on their degree of sophistication, trust management systems can provide more or less functionalities. We are particularly interested in the TMS that output detailed answers. Figure 3.6 illustrates the degree of sophistication a TMS can achieve by providing more or less elements in its answers.
Based on the type of information provided by the TMS, Seamons and colleagues identified two functioning modes of trust management systems [102]. In our work, we distinguish four modes that we summarise as follows:

- **Mode 1**: In this mode, the TMS produces a boolean answer (trust/no trust) that states whether the credentials provided satisfy the policy.
- **Mode 2**: In addition to the boolean answer, in this mode the TMS provides a *justification*,

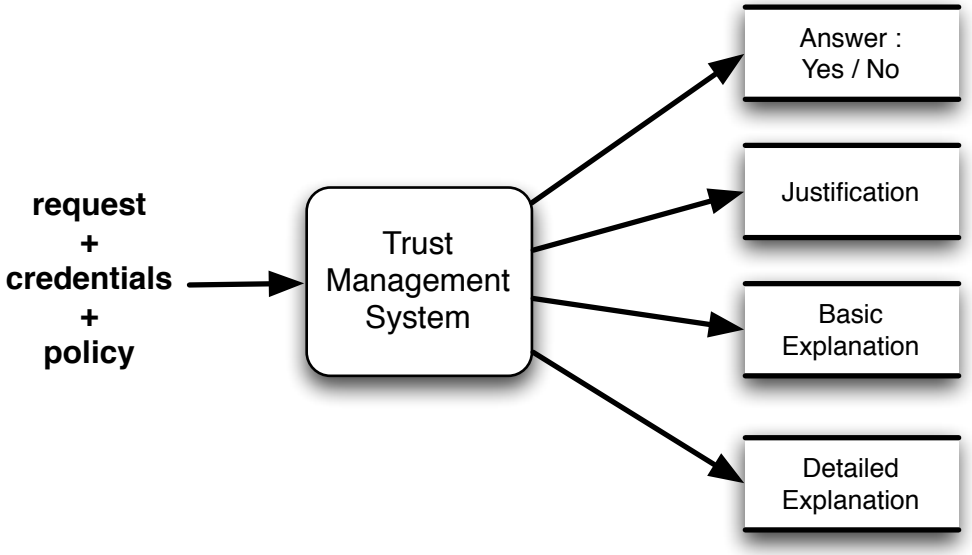

Figure 3.6: Functioning modes of a trust management system

when the request is denied, that states which conditions in the policy the provided credentials were unable to satisfy.

- **Mode 3**: In this mode, the TMS provides an answer, a justification and an *explanation* when the policy is satisfied. The explanation contains all credentials that satisfy the policy.
- **Mode 4**: This last mode extends the third mode as it provides a *detailed explanation*. The detailed explanation is obtained by providing all subsets of credentials that satisfy the policy.

Modes 1 and 2 are often used by the resource owner to verify whether the credentials its interlocutor provided satisfy its policy, while modes 3 and 4 are used by the requester to determine whether the credentials it possesses (and which subset of credentials) satisfy the policy stated by the owner of the resource. These latter modes were particularly introduced in the context of trust negotiation that we will tackle in the next section.

Automated Trust Negotiation

Automated Trust Negotiation (ATN) is an approach to trust management in which trust is established through the gradual, iterative, and mutual disclosure of credentials and access control policies [98, 128]. Unlike the traditional trust management systems that have have been presented in the previous section, automated trust negotiation approach consider credentials as first class resources that should be protected through *release policies* dedicated to them. Consequently, ATN systems provide users with better fine-grained and flexible control over the disclosure of the potentially sensitive data conveyed by their credentials [77, 98, 123].
However, negotiation generally refers to the process by which agents can reach an agreement on matters of common interest [90]. We adapt this well-established definition to the *negotiation-based adaptation* of trust policies.

Definition 4 (Automated Trust Negotiation) *Automated trust negotiation is an iterative process in which two interacting agents reach an agreement on the credentials they are willing to release to gain each other's trust.*

The introduction of trust negotiation has several benefits. First, it better reflects the asymmetric nature of trust. It allows also the establishment of bilateral trust as both participants in an interaction can request credentials from each other. Finally, it allows a more flexible trust management as trust is established gradually and incrementally. Research on trust negotiation has been principally focusing on *how to make trust management systems achieve trust negotiation?* and *how to make trust negotiation successful?*. The first question represents the requirements for trust negotiation, while the latter represents *trust negotiation strategies*. The requirements are further divided into *requirements for trust management systems* and *requirements for policy specification languages*.
Trust negotiation is inherently a strategy-driven process as the choice made by the individual affects the amount of credentials it releases and the time it makes in this task [77]. Therefore, recent research in trust negotiation area has been primarily focusing on proposing efficient and optimised negotiation strategies and protocols. Generally speaking, however, the implemented negotiation strategies fall into three categories: *eager*, *parsimonious* and *prudent* strategies [54].

Eager Strategy: In the eager strategy, participants in the negotiation adopt a naive position in which they disclose almost all credentials they possess. The negotiation is considered to be successful when each participant received enough credentials to be assured about the interaction he is engaged in. The major advantage of this strategy is that it does not require the use of *release policies* and it minimises the time of the negotiation [9]. However, this strategy increases the amount of disclosed credentials and thus the sensitive date they convey.

Parsimonious strategy: With this strategy, participant exchange only credentials requests (no credential is released) and tries to find a possible sequence of credentials disclosure that can lead to a successful negotiation [54]. Also, in parsimonious strategies, only credentials that are explicitly requested are released. Unlike the eager strategy, the parsimonious one minimises the credentials exposure but increases considerably the time of the negotiation without any guarantee of success.

Prudent strategy: is a mix of the previous strategies. An eager strategy is applied to credentials that are not sensitive and a parsimonious strategy is used for the sensitive ones. This strategy has been proved to over-perform the other strategies in situations where the negotiation involves the disclosure of sensitive and non sensitive credentials [129].

Survey on Trust Management Systems and Models

With the advance of cloud computing, the objective of researchers on security is to propose *decentralised* and *fine-grained*access control mechanisms to leverage the distributed nature of these systems. The general idea is to allow resource owners to state *who* they trust and for *which* issue. Trust management systems (TMS) were originally designed to solve the problem of deciding whether a request to perform a potentially harmful action on a sensitive resource comply with the access control policy [22, 20]. Nevertheless, in Distributed systems, such as Distributed Cloud Computing, these system can be used in a broader way to evaluate whether a trust decision complies with the user requirements or not. We believe that these requirement can be expressed as SLAs or trust policies, but any other specification mechanisms can be considered as well.

In this section, we review a selection of trust management systems. These systems are split into *decentralised trust management systems (DTM)* and *automated trust negotiation systems*. DTM systems are further divided into *authorisation-based TMS (ABTMS)* and *Role-Based TMS*. These systems are presented in the chronological order in which they have been published. We think that respecting this chronological causality helps the reader to understand key elements of each system.

Authorisation-Based TMSs

The *authorisation-based TMSs* category relates to systems that pioneered the trust management approach. Whilst most of these systems are considered nowadays as obsolete, the mechanisms they proposed remain valid and can be found at the basis of most of modern trust management systems. They inherit from IBAC and ABAC models. They build a trust chain in order to map a credential holder to the authorisation it can be trusted for.

PolicyMaker [22]

PolicyMaker is the first application stamped as a trust management system. This system introduces the concept of programmable credentials and policies by means of an assertion language. The syntax of an assertion is:

```
Source ASSERTS Subject WHERE Filter
```
(3.1)

The above syntax can be used to specify both credentials and policies. Here, the statement represent a credential by means of which a *source* authorises a *subject* to perform actions that are accepted by the *filter* (i.e. interpreted program). The main difference between a credential and a policy is that in policies the keyword *policy* is always used as the source of the assertion. For instance, we can use the assertion given below to authorise the entity holding the public key ``rsa:123'' to access all resources shared among the community.

```
policy ASSERTS ``rsa:123'' WHERE filter to access all shared resources
```
(3.2)

In PolicyMaker, an assertion (whether credential or policy) is used to state that the source (or the local application in the case of a *policy*) trusts the key holder to perform the actions accepted by the filter. However, the formalism used to encrypt keys is left to the application using PolicyMaker, thus PolicyMaker is generic with respect to the keys encryption scheme. The semantic of the operations and the enforcement of the trust evaluation decisions are also left to the application.
Typically, the application provides to the PolicyMaker trust engine a set of requested actions, a set of credentials and a policy. The objective of PolicyMaker is to check whether the credentials form a delegation chain by means of which the action can be linked to the key of the keys. PolicyMaker then replies with an answer (``Trust'' or ``False'') and it is up to the application to interpret the answer and take the appropriate decision. The functioning of the PolicyMaker engine is similar to the architecture we described in Figure 3.5 (cf. Section 3.3.3). PolicyMaker does not support *negotiation*. Policies are evaluated in a static and context-independent way. Finally, PolicyMaker can not be used to manage resources that are owner by more that one individual.
In PolicyMaker, the choice of the policy specification language is left open which makes policy evaluation undecidable in the most cases [54]. KeyNote [21], its successor, overcomes this drawback and imposes that the policies must be written in a specific language. KeyNote makes also the cryptographic verification which was left to the application in PolicyMaker.

REFEREE [31]

REFEREE (Rule-controlled Environment For Evaluation of Rules and Everything Else) is a W3C and AT&T joint trust management system used for web document access control. The system was developed based on the PolicyMaker architecture, but the functioning of the system is somehow different. Here, it is the resource providers (i.e. authors of web content) that are trying to gain the trust of the resource consumer (i.e. the site visitor). The system was essentially used to prevent minors from accessing illegal content. The system uses PICS (Platform for Internet Content Selection) labels as credentials that the REFEREE trust engine (a browser plug-in) evaluates with respect to a local policy.
Profiles-0.92 is a rule-based policy language that was designed for REFEREE. As illustrated in Listing 3.1, each policy is an "s-expression" that is evaluated in a top-down manner.

```
(((invoke "load-label" STATEMENT-LIST URL "http://www.emse.fr/")
(false-if-unknown
(match
(("load-label" *)
(service "http://www.emse.fr/CA.html") *
(ratings (RESTRICT > trust 2)))))
STATEMENT-LIST))
```

Listing 3.1: Example of a policy specified in Profiles-0.92 (adapted from [31]).

The above policy states that any document having a label (certified by *emse*) with a trust rating greater than 2 can be viewed by the user. The matching between labels and the conditions specified in the policy is purely syntactic. Thus it remains to the application and the labelling institution to define its semantic.
REFEREE trust engine evaluates the above policy in two steps. First, tries to find and download labels provided by the server which URL has been specified in the policy. Then a pattern-matcher is run to find a label with a trust rating. If the rating is greater than 2 the result of the evaluation would be `true`, if not the result would be `false` and if no label was found the result would be `unknown`. Thus, REFEREE trust engine implements a three-valued logic, specially for the management of the meaning of `unknown` [31, 54].

Binder [42]

Binder is the first trust management system which uses a logic-based policy language [42]. The particularity of Binder lies in its explicit specification of right delegation though the extension of Datalog with the `says` construct [29]. In Binder, credentials represent keys which holder use to sign delegation assertions. Then policies are used to filter these assertions and map them to their authors. The specification language proposed in Binder allows the expression of two type of declarations: *beliefs* and *policies*. For instance, the following declaration is used by a individual A to state that another individual B can be trusted for joining his community and for reading his personal files.

```
can(B, read, MyFile).
can(B, join, MyCommunity).

can(X, join, MyCommunity) :- Y says trust(Y,X), can(Y,join,MyCommunity)
```

Listing 3.2: Examples of Binder declarations (beliefs and policies).

The above example illustrated also the declaration of policies in Binder. In this example, the policy states that if A trusts an individual Y to join his community and that Y trusts another individual X, this latter can also be trusted to join the community. Worth noting in this policy is the use of the `says` construct.
The role of the Binder trust engine is to evaluate policies with respect to local assertions (i.e. beliefs) and assertions made by others (i.e. others' beliefs). The main novelty of the system lies in the addition of the `says` construct each time an assertion is received from others. Indeed, each time an individual sends an assertion, the Binder trust engine transforms this assertion into a certificate which is signed with the private key of the issuer. Then these assertions are sent to other Binder engines in order to make trust decisions. When an assertion is received, Binder verifies the validity of the certificate and automatically quotes the assertion with the `says` construct to distinguish them from local assertions.

SULTAN [54]

SULTAN (Simple Universal Logic-based Trust Analysis) is a TMS that has been proposed for the specification and analysis of trust and recommendation relationships [54]. In SULTAN, credentials

represent certified statements about identity, qualification, risk assessment, experience or recommendations. The system is provided with a policy specification language in which policies are used to specify two types of policies: *trust/distrust* policies and *positive/negative* recommendation policies. In fact, trust policies corresponds to classical meaning of policies used in this report, while recommendation policies are statements by means of which individuals make recommendations to each others. In the following, we provide the syntax used to specify both types of policies.

$$PolicyName : \mathbf{trust}(Tr, Te, As, L) \leftarrow Cs; \tag{3.3}$$

The above policy represent a trust policy by means of which a trustor (i.e. Tr) trusts (or does not trust) to some extent (L corresponds to the level of trust) a trustee (i.e. Te) with respect to an action set (i.e. As) and if the conditions hold (i.e. Cs). Similarly, the recommendation policy defined hereafter specifies that the recommender (i.e. Rr) recommends at a recommendation level (i.e. L) the recommended agent (i.e. Re) to perform the action (i.e. As) if the conditions (i.e. Cs) hold.

$$PolicyName : \ \mathbf{recommend}(Rr, Re, As, L) \leftarrow Cs; \tag{3.4}$$

The SULTAN trust engine is responsible of collecting the information required for the policy evaluation, making trust relationship decisions, and monitoring the environment in the perspective of re-evaluating existing trust relationships.

Ponder [36]

Ponder is merely a policy language for which there was no associated trust management system [36]. Ponder is the first object-oriented policy language that adopts a role-based approach. Nevertheless, many of the features proposed by this language inspired other systems which explains our motivation to review it. Ponder is a declarative language which can be used for the specification of four types of policies, namely *authorisation*, *obligation*, *refrain* and *delegation*. Ponder pioneered the use of a deontic approach which was reused later by other languages such as Rei [64] and KAos [116, 115]. Furthermore, the main novel aspect of Ponder lies in the constructs it provides for updating, organising and handling policies on runtime according to the environment context.
For instance, the following example is an instantiation of a positive authorisation policy type called `rights`. The policy specifies that `members` (the subjects of the policy) can *modify* the target objects of type `file` that are stored in the common space of the community `com`.

```
type auth+ rights(member, target <file> com) {action modify(); }
```
(3.5)

Another interesting aspect of Ponder policies lies in the fact that subjects and objects to which a policy applies can be grouped. For instance, in the above example, the policy concerns all members of the community and all files, making factorisation of rights implicit and more efficient that what can be expressed in RBAC models or any other role-based trust management systems. Further, the authors assumed that their language is flexible, scalable and extensible; flexible as it allows the reuse of policies since many instance of the same policy can be created for many different conditions; scalable as it allows the definition of composite policies; and extensible as it accepts the definition of new types of policies that can be considered as sub-classes of existing policies, thanks to the object-oriented approach. However, due to the absence of implementation, none of these properties have been proved to be valid.
Recently, Ponder has been redesigned, as *Ponder2*, to increase the functionality of authorisation policies (e.g. operators for all managed objects have been added). In contrast to the previous version, which was designed for general network and systems management, Ponder2 has been designed as an entirely extensible framework that can be used at different scales: from small embedded devices to complex services and virtual organisations.

Role-Based TMSs

In *authorisation-based TMSs*, the delegation of authority is used in a very restrictive way. For instance, in PolicyMaker, a community member cannot simply specify that "all members of my community can access my resources". In these systems, the solution would be to delegate the right to the members I know and authorise these members to further delegate the right to each member they know. This approach makes trust decision complex and difficult to manage and control (e.g. a member can delegate the right to non members).
In response, a new generation of *role-based* trust management systems that take advantage of the strength of RBAC and trust management approaches have been used. From RBAC, role-based TMS borrow the concept of role and from trust management, they borrow delegation and distributed credentials certification. The combined use of roles and distributed credentials management makes these systems convenient for large scale systems such as virtual communities.

IBM Trust Establishment [57]

IBM Trust Establishment (IBM-TE) is a role-based trust management system developed by IBM for e-commerce applications. The main objective of IBM-TE is to map credential holders to groups. Credentials are specified in a generic language but the system provides transcoding mechanisms to handle X.509. Credentials are used to authenticate users, while policies are used to express restrictions on how a credential holder could belong to a group. Groups are used in the sense of roles which are mapped to authorisations (cf. Section 3.2.3). IBM-TE comes with a dedicated XML-based policy specification language called *Trust Policy Language* (TPL). An example that illustrates the TPL syntax is given in Listing 3.3.

```
<GROUP NAME="Community">
<RULE>
<INCLUSION ID="reco"
           TYPE="Recommendation"
           FROM="members"
           REPEAT="2">
</INCLUSION>
<FUNCTION>
           <GT>
           <FIELD ID="reco" NAME="Level"></FIELD>
           <CONST>1</CONST>
           </GT>
           </FUNCTION>
</RULE>
</GROUP>
```

Listing 3.3: A fragment of a TPL Policy specifying a membership rule.

The above policy is used to add a new member to the group `Community` (role). The policy states that a new member can be admitted if he can provide two recommendations of existing members. For that, two XML tags are used: *inclusion* and *function*. `Inclusion` tag defines the credentials that the requester must provide (e.g. two recommendation credentials), while the `function` tag allows the definition of additional conditions over requested credentials (e.g. recommendations must be greater than, `GT`, 1). The trust engine processes credentials in a traditional way. Along with his request, the requester provides the set of credentials he possesses. These credentials are then evaluated by the engine to determine to which group the requester can be mapped [57].

Role-Based Trust Management Framework [81]

The *Role-Based Trust Management Framework* (*RT*) was initially designed to support trust decision making in collaborative environments [81].
In *RT*, roles are represented as attributes, so an individual is said to belong to a role if it possesses a credential in which the role identifier is specified. Unlike IBM-TE, *RT* uses an extension of Datalog [29] to represent both credentials and policies. In *RT*, the terms credential and policy are used interchangeably. However, *RT* uses the term *certificate* in reference to the meaning of credential we use here. The formalism used to specify certificates have not been defined, but some approaches proved the compatibility of *RT* with both X.509, PGP and any certification mechanisms using PKI. The syntax used to specify policies is defined as follows:

$$Com.evaluator(?X) \leftarrow Com.recommender(?X) \wedge Com.Member \tag{3.6}$$

The above policy states that any member of the community that recommended another member is allowed to evaluate it. The *RT* policy specification language comes in five flavours RT_0, RT_1, RT_2, RT^T and RT^D [81]. RT_0 is the basic language that supports roles hierarchies, delegation of authorities over roles, roles intersection and attribute-based delegation of authority. RT_1 adds to RT_0 the possibility to add parameters to roles (e.g. in the previous policy, the evaluator and the recommender roles are endowed with parameters), RT_2 adds to RT_1 logical objects. They represent a way to group (logically) resources of access modes in order to ease the specification of policies. RT^T adds thresholds to roles, while RT^D supports delegation of roles activation (i.e. roles that are active only within a specific context).

Cassandra [15]

Cassandra is a TMS that aims at enabling individuals involved in potentially large scale systems (e.g. P2P systems) to share their respective resources under the restriction of local policies [15]. The system has been principally deployed in the context of electronic health records access control. Cassandra is role-based and supports X.509 credentials. Policies in Cassandra are expressed in a language based on Datalog with constraints [29]. For instance, a policy can state that an individual A can delegate the role of *moderator* as long as he commits to the role *admin*. This policy is specified using the constructor `canActivate()` as follows:

$$\texttt{canActivate}(B, moderator) \leftarrow \texttt{hasActivated}(A, admin) \tag{3.7}$$

Cassandra provides flexible delegation mechanisms which allow the explicit specification of delegation lengths [15, 40]. In addition, Cassandra proposes a mechanism for roles revocation, including cascade roles revocation (e.g. if an individual is revoked from its role, all individuals to which he delegated the role are revoked too). Finally, the Cassandra policy language allows the specification of roles hierarchies. Cassandra trust engine is only available as a proof of concept implementation in OCaml. The main feature of the engine is the implementation of the semantic of Cassandra policies for their evaluation.

Automated Trust Negotiation Systems

In trust management, a trust decision comes after a complex interaction process, where parties exchange policies and credentials. Traditionally, in early TMS, trust is established in an unidirectional way: the resource owner is assumed to be trusted by the requester. Consequently, before manipulating the resource, the requester must provide its credentials to know whether its request is accepted or not. So if the request was not accepted, the requester would have uselessly released its credentials (which contains sensitive data such as its id, its age or its address). Therefore, due to privacy considerations, such an approach is not acceptable. To that aim, several *negotiation-based TMS* have been proposed.

TrustBuilder [130]

TrustBuilder was the first TMS to introduce the concept of trust negotiation. TrustBuilder uses X.509 certificates and TPL policies (cf. Section 3.4.1). The authors reused also the IBM-TE engine for the evaluation of policies. So TrustBuilder can be considered as an extension of IBM-TE to include negotiation features. The main novelty of this system lies in its rational management of credentials disclosure. To that aim, the trust engine is endowed with a negotiation module in which strategies are used to determine safe credentials disclosure for both parties involved in the interaction.

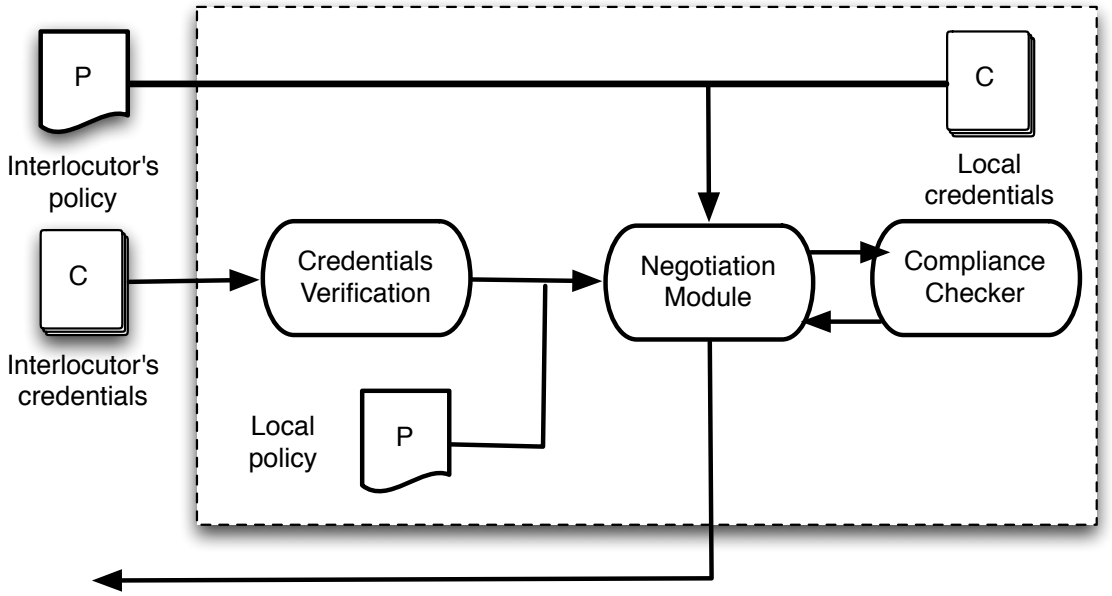

Figure 3.7: Architecture of the TrustBuilder TMS

As illustrated in Figure3.7, TrustBuilder engine is split into three sub-modules: *credentials verification module*, *negotiation module* and *compliance checker module*. The core element of this architecture is the negotiation module which is responsible of enforcing negotiation strategies. The objective of a negotiation strategy is to minimise credentials disclosure. To that aim, TrustBuilder evaluates iteratively the policy of the interlocutor and the set of local credentials to compute the minimal set of credentials that satisfy the policy (as depicted in Figure 3.7).
Recently, Lee and colleagues proposed an extension of TrustBuilder that they called TrustBuilder2 [78]. This extension aims at endowing the TMS with four main functionalities: support of arbitrary policy languages, support of arbitrary credentials format, integration of interchangeable negotiation strategies and flexible policies and credentials management.

Fidelis [126]

Fidelis is a TMS that originates from the OASIS (Open Architecture for Secure, Interworking Services) distributed authorisation architecture project. Fidelis makes use of keys, X.509 and PGP as credential, and policies and credentials are systematically specified by distinct entities. Fidelis distinguishes two types of entities: *simple* and *composite* principals. Simple principals are in fact public keys while composite principals are groups of public keys (e.g. groups or communities).
The *Fidelis Policy Language* (FPL) is the core element of the system. The language is able to express recommendations and trust statements. The syntax of the language is presented in the following example.

```
any-statement: ind -> statement
```

```
asserts any-statement: self -> statement
where ind == 0x14ba9b925 || ind == 0x5918b01a || ...
```

Listing 3.4: A blind delegation policy in Fidelis.

This policy represents a special type of delegation policies, called *blind delegation*. It is used to make an individual "blindly" trust and assert all assertions made by other individuals. In this example, the group of trusted individuals is constrained by the variable `ind`.
On the top of the FPL, the authors developed a trust negotiation framework in which meta-policies are used to specify negotiation strategies. Meta-policies are designed to express four types of conditions about when a credential could be disclosed: designated principal disclosure, context-specific disclosure, trust-directed disclosure, and mutual exclusion [126]. For instance, the following meta-policy is used to disclose the trust statement `T2(a, b): self->p` (which is used as a credential here) when negotiating with the `0xb258d29f` key holder.

```
negotiator(): self -> 0xb258d29f
grants disclose(T2(a, b): self->p)
```

Listing 3.5: A credential disclosure meta-policy in Fidelis.

Fidelis does not support standard negotiation strategies. Thus termination property is not guaranteed making the evaluation of a policy not decidable in many situations. Finally, Fidelis distinguishes between *static* policies and *live* policies. Static policies do not depend on environment variables (e.g. date, time) to be evaluated, while live polices must be queried dynamically and tailored to each request. Nevertheless, live policies were only used in the context of negotiation as presented above and no adaptation mechanisms have been proposed as the description may suggest.

Trust-$\mathcal{X}$ [18]

Trust-$\mathcal{X}$ is a TMS that was designed for trust negotiation in peer-to-peer systems [18, 19]. Trust-$\mathcal{X}$ is built upon two bricks: $\mathcal{X}$-profiles and $\mathcal{X}$-TNL. $\mathcal{X}$-profiles are data structures used to store user's credentials along with uncertified declarations containing information about them (e.g. age, mail, address). $\mathcal{X}$-TNL stands for *XML-based Trust Negotiation Language*. $\mathcal{X}$-TNL has been developed for the specification of Trust-$\mathcal{X}$ certificates and disclosure policies.

```
<policySpec>
<properties>
        <certificate targetCertType= Corrier_employee>
        <certCond>
        //employee number[@code=Rental Car.requestCode]
        </certCond>
        <certCond> /.../[position=driver]
        </certCond>
        </certificate>
</properties>
        <resource target="Rental_Car"/>
        <type value="SERVICE"/>
</policySpec>
```

Listing 3.6: Example of $\mathcal{X}$-TNL policy specification.

The code in Listing 3.6 shows an example of an $\mathcal{X}$-TNL policy defined by a rental car agency. The agency allows drivers of the Corrier society, which is part of the agency to rent cars without paying. This policy can be satisfied by providing a credential which is specified using the $\mathcal{X}$-TNL too. The syntax of a credential is described in the example provided in Listing 3.7.

```
<Corrier Employee credID='12ab', SENS= 'NORMAL' >
<Issuer HREF='http://www.Corrier.com' Title=Corrier Employees Repository/>
<name>
<Fname> Olivia </Fname>
<lname > White </lname>
</name>
<address> Grange Wood 69 Dublin </address>
<employee number code=34ABN/>
<position> Driver </position>
</Corrier Employee>
```

Listing 3.7: Example of $\mathcal{X}$-TNL profile

The trust-$\mathcal{X}$ engine provides a mechanism for negotiation management. The main strategy used in Trust-$\mathcal{X}$ consists in releasing policies to minimise the disclosure of credentials. So, only credentials that are necessary for the success of a negotiation are effectively disclosed [107]. Thus Trust-$\mathcal{X}$ makes use of a prudent strategy (cf. Section 3.3.4).
The primary novel aspect proposed in Trust-$\mathcal{X}$ consists in the use of *trust tickets*. Trust tickets are issued upon successful completion of a negotiation. These tickets can later be used in subsequent negotiations to speed up the process in case the negotiation concerns the same resource. Additionally, Trust-$\mathcal{X}$ provides also a mechanism to protect sensitive policies. This is achieved using *policy-precondition*; policies are sorted logically so that the satisfaction of a policy is the precondition of the disclosure of the subsequent policies.
Recently, Braghin [25] proposed an interesting extension in which the framework is used to handle negotiations between groups of individuals instead of only between two individuals.

ATNAC [98]

Adaptive Trust Negotiation and Access Control (ATNAC) is an integrated TMS that combines two existing systems: GAA-API and TrustBuilder (already presented in Section 3.4.2). GAA-API is a generic authorisation and access control system that captures dynamically changing system security requirements. The system uses X.509 credentials to convey information between negotiating partners. ATNAC uses TPL to express trust policies that, in addition to the partner's properties, explicitly refer to the context suspicion level (SL). In ATNAC, several policies are specified and used to protect the same resource. So the main novelty of the system lies in the trust engine that monitors the environment suspicion level (thanks to GAAI-API) and adapts the selected policy based on the suspicion level. This mechanisms is used in ATNAC to provide adaptive trust negotiation in order to counter malicious attacks.

	Suspicion Level		
	low	medium	high
R_1	freely	freely	freely
R_2	freely	freely	C_1
R_3	freely	C_1	C_1 and C_2

Table 3.1: Example of policies used in ATNAC (adapted from [98])

The above table illustrates the adaptive approach advocated by ATNAC. In this example, the resource R_1 is non sensitive and thus it is disclosed independently from the suspicion level. In contrast, R_3 can be freely disclosed in the context of *low* LS, requires a credential C_1 when SL is *medium* while both C_1 and C_2 are required when SL is *high*.

PROTUNE [23]

The *PRovisional TrUst NEgotiation framework* (PROTUNE) is a system that provides distributed trust management and negotiation [24, 23] features to web services. The PROTUNE framework provides: (a) a trust management language, (b) a declarative meta-language for driving decisions about information disclosure, (c) a negotiation mechanism that enforces the semantics of the meta-language, (d) a general ontology-based approach to support the policy language extension, and (e) an advanced policy explanation mechanism that is able to output *why*, *why not* and *how to* answers that are important during a negotiation process. One of the main advances made by PROTUNE lies in the use of *declarations* along with credentials during the policy evaluation process. Declarations are the unsigned equivalent of credentials. They can also be considered as statements that are not signed by a certification authority. However, the most novel part of the project remains the policy specification language which combines access control and provisional-style business rules.
In PROTUNE, policies are specified using the rule language defined in [24]. The language is based on logic program rules extended with an object-oriented syntax[3].

```
allow(X, access(Resource)) :-
    goodReputation(X), validID(C).
validID(C) :-
        credential(identity, C[subject:X]).
goodReputation(X) :-
        declaration(Y,X,reputation(R)), Y!= X,  R > 50.
```

Listing 3.8: Example of a PROTUNE access policy.

The above PROTUNE policy states that an individual must provide a valid credential proving its identity and that he must have a good reputation to be allowed to access a resource. This kind of policy is called access policy. Similarly PROTUNE implements negotiation strategies through *release* policies which states under which conditions a credential can be released. An example of such resource is provided hereafter.

```
allow(release(credential(C[type:identity]))) :-
     credential(ta, Cred[issuer:'Trusted Authories']).
```

Listing 3.9: Example of a PROTUNE release policy.

This policy states that identity credentials can be released only to individuals providing a credential that proves that they belong to the *'Trusted Authorities'* group.

XeNa [3]

Abi Haidar and colleagues [3] proposed an XACML Negotiation of Access which brings together negotiation for trust establishment and access control management within the same architecture. XeNA trust engine ropose full support of XACML access control and negotiation policies.
In XeNA, the negotiation process based on resource classification methodology occurs before the access control management. A negotiation module at the core of this negotiation process is in charge of collecting resources required to establish a level of trust and to insure a successful evaluation of access. The access control management is based on an extended Role-Based Access Control (RBAC) profile of XACML. This extended profile responds to advanced access control requirements and allows the expression of several access control models within XACML.

[3] $A.at : v$ means that the individual A has the attribute at and that this attribute has the value v. This expression is in fact an abbreviation of the logic predicate $at(A, v)$

A Preliminary review of Trust Models in Cloud Computing

Trust in cloud computing has been addressed in the literature from two different perspectives; the cloud service consumer (CSP) and cloud service provider (CSC). In this section, we classify existing approaches from both perspectives:

Credentials based Trust Models

These models draw their inspiration from the approach advocated in *decentralised Trust Management Systems* presented previously (cf. Section 3.4.1). In theses models, trust is established between CSP and CSC using credentials (i.e. certificates and/or keys) issued by trusted third parties or certification authorities. An entity A trusts another entity B if and only if it receives the adequate credentials to build a chain of trust that links B to A. This models reproduces the Web of Trust (WoT) approach in which each member vouch for each others [1, 2].

SLA based Trust Models

In [88], the authors proposed the "Trust as a Servics" (TaaS) framework in which the authors introduced and *adaptive credibility model* that distinguishes between credible trust feedbacks and malicious feedbacks by considering cloud service consumers' capability and majority consensus of their feedbacks. In this work, trust has been addressed from the perspective of users.

In [55], the authors proposed a multi-face model to manage trust in Cloud Computing marketplaces. The model collects several attributes to assess the trustworthiness of a Cloud Service Provider. These attributes corresponds to Service Level Objectives (cf.s Section 3.1.1) defined within active SLAs. Feedback information is also collected from different sources and used alongside SLA metrics to derive a trust score for each CSO. The authors refers to the CAIQ (cf. Section 3.1.4) as a way to extract SLA compliance information.

In the Joint Risk and Trust Model (JRTM) developed in the context of the A4Cloud (Accountability for Cloud) [28], statistical data (i.e. proofs and indicators) collected from third party services (i.e., a Trust as a Service Provider) are accumulated and computed to estimate the trust that a Cloud Customer put on a specific Cloud Service Provider. The model relies on the assessment of the cloud service security and pricacy risk to derive a trust metric. The information used include for instance statistics on the number of security and privacy incidents that the CSP was subject to.

Feedback based Trust Models

In these models, trust is built upon feedbacks, ratings and opinions that CSP and CSC express based on their past experience with each others. In these models, each entity (CSP and CSC) is responsible of collecting, processing and sharing a reputation measure. In general, feedbacks are expressed with respect to business (e.g., QoS) and security parameters. Thus the object of a feedback can be any property that CSP and CSC can rates each others on it. Feedbacks are more general compared to reputation. But the models used in reputation based-trust models are comparable to the one used for feedback management. Reputation is the social evaluation of a group, a community or a society of agents towards the trustworthiness of an individual [99].

In DAI, and more particularly in multi-agent systems, reputation has been considered as a substantial dimension of trust [63]. In the following, we review some predominant reputation models.

ReGreT [99] is a well known decentralised trust and reputation model for e-commerce. Proposed by Sabater and Sierra in 2001, the main objective of *ReGreT* was to make more accurate trust evaluations. To that aim, the authors used three factors based on which trust was computed: *the direct*

experience, *the global reputation* and an ontological *fine-grained reputation* which defines reputation values for each trait of the individual using the ontology. In ReGreT, the network to which the agent belongs is used to assess the credibility of the information provided by each agent. Social relationship are presented in the form of fuzzy rules which are later used to determine whether the witness information provided by an agent should be considered or not.

Jøsang [63] proposed a reputation model (called the Beta Reputation System) for the decision making in the context of e-commerce transactions. The authors used the concept of reliability along with the probability of success to determine the trustworthiness of a partner. The reliability of an individual is assessed in a direct and indirect way. The direct reliability is computed based on previous knowledge about the partner, while the indirect one is given by recommendation from other trust third party. The indirect reliability is then computed by making the average of all recommendation weighted by the recommender trust degree. Then this value is combined with the direct reliability in order to derive a trust degree. Once this trust degree obtained, it forms a belief that is described as set of fuzzy propositions such as "A believes that B is very trustworthy".

FIRE [60] is another important model which has been designed by Huynh and colleagues for multi-agent systems. The authors compute trust based on past experiences, the role of the agent, its reputation and a kind of certified reputation. Roles are used to determine to which degree an agent that have a certain position in the society could be trusted. The main idea is that trust depends on the fulfilment of the role ascribed to the agent. Also, the authors make a distinction between witness reputation and certified reputation. Certified reputation is a reputation that comes from certified presumably trusted witness, while normal reputation comes from every agent of the society.

Predictions based Trust Models

In [122] the authors defined a similarity based prediction model. Entities (i.e., cloud users and cloud providers) are represented using a vector of capabilities and interests. The more theses vectors are similar the more likely trust can be established between them.

In [55] the authors presented a behavior-based trust model in which the trust value depends on the expected behavior of the cloud provider. The behavior of the provider is assessed with respect to specific attributes such as security measures, compliance and customer support. Here again, the authors focus on the perspective of the user that tries to select the best provider.

"Hardware" Trust Management

The cloud computing trust model is based on the notion of transitive trust. It means that if entity A trusts entity B, and entity B trusts entity C, then entity A trusts entity C. Using this property enables building a chain of trust (CoT) from a single root of trust (RoT). The trust of the host platform lies in the platform's hardware trust. An outside entity can determine the trustworthiness of the platform via attestation from TPM, which is the Root of Trust Reporting (RTR) on the hosting platform. The TPM performs integrity measurement on the platform in two ways:

- **Static Root of Trust for Measurements (SRTM):** as the part of TCG 1.1b specification, the platform is assumed to be in a secure state and starts in an immutable environment at the boot time. A set of instructions called the Core Root of Trust for Measurement (CRTM) is then executed on the platform to measure the BIOS. After computing a hash value, it is sent to the TPM that keeps it in a Platform Configurations Register (PCR). Once the trust is established in the system by measuring the BIOS, CRTM passes control to the measured BIOS, where it measures the next component in the boot chain and again, will store the value in a PCR of the

TPM. This process is executed for each component in the boot sequence and the rest of the Trusted Computing Base (TCB) of the platform. The TCB includes all measured components that provide the foundation of trust in the platform.

- **Dynamic Root of Trust for Measurements (DRTM):** DRTM intends to provide the capability of executing code from a clean slate while the system is running, without any rebooting (like SRTM). The major chip vendors AMD and Intel implemented technologies that allowed the DRTM. Although AMDs Secure Virtual Machine (SVM) and Intels Trusted Execution Technology (TXT) (former code name LT LaGrande Technology) are more than just DRTM implementation, DRTM is the core concept on which they are built. In order to execute a custom code (aka Measured Launched Environment - MLE), Intel TXT creates a trusted environment from an untrusted state by invoking a set of security instructions (SMX) on the processor to perform a very specific set of tasks (GETSEC). GETSEC ensures that a very special code, i.e. SINIT Authenticated Code Module (ACM) can be executed safely. During this process, all but one CPU are disabled and all current running processes, interrupts and I/O (via IOMMU, e.g. to avoid DMA attacks) are blocked/stopped. Afterward, CPU rejoins in a clean state and anything executed before is discarded. At this point, the signature of the special code (SINIT ACM) gets validated and its hash measurement is sent to the TPM (in the PCR 17). Next, execution is passed to the ACM which then measures MLE and sends the measurement to the TPM (in the PCR 18). Finally, execution is passed to the MLE.

The security of those mechanisms relies on the fact that PCRs values cannot be set (or forged) but only extended. This means whenever a measurement is sent to a TPM, the hash of the concatenation of the current value of a PCR and the new measurement is stored. Obviously, there's a beginning to all of this: - With SRTM, only the CRTM can reset PCRs 0 to 15 at boot - With DRTM, only the TXT instructions can reset PCRs 17 to 20 (when in locality 4 (SMX operations)).

Secure Boot

The goal of the Secure Boot is to make sure the system boot loader is trusted and signed by the publisher or the manufacturer of the system. Through signature verification in the next-stage boot loader(s), kernel, and, potentially, user space, the execution of unsigned code can be avoided. During the boot process, TPM-enabled devices can validate the integrity of the machine, enabling protection and detection mechanisms to function in hardware, at pre-boot and in the secure boot process. The secure boot process implements a chain of trust. Begining with an implicitly trusted component, all other components can be authenticated before being executed.
The Unified Extensible Firmware Interface (UEFI) secure boot was created as a BIOS 'replacement' to enhance security in the pre-boot environment, by sitting between hardware and OS at the BIOS level. UEFI Secure Boot only requires a non-volatile (flash) storage which can be switched from read-write mode to read-only mode during system boot. This storage is used to store the UEFI implementation and UEFI protected variables (e.g. the trusted root certificate). Through using a complementary TPM, UEFI secure boot and the operating system can take measurements of all phases of the booting process. It can be used to detect modified boot code, settings and boot paths while providing sufficient information to an outside entity to attest to the platform state after the boot process has been completed. Although it does not prevent an insecure boot from occurring, the measurements are available to detect any modification to the system.
In a different approach, ARM's TrustZone enables secure services to run in the secure world of the processor. TrustZone is a set of security extensions added to ARM processors, so that an ARM processor can run a secure operating system (secure OS) and a normal operating system (normal OS) at the same time from a single core. Because security is designed into the hardware, TrustZone avoids security vulnerabilities caused by proprietary, non-portable solutions outside the core. ARM does not specify the root of trust for TrustZone. It is usually assumed that a unique device key which is

accessible only inside the secure world of TrustZone is available, and this device key serves as the root of trust for TrustTone.

Trusted Execution Environment

A Trusted Execution Environment (TEE) is a secure area that resides in the application processor that guarantees that sensitive data is processed and protected in an isolated, trusted environment with respect to confidentiality and integrity. The TEE offers isolated safe execution of authorized security software, and protects the integrity and confidentiality of key resources, such as the user interface and service provider assets. It provides an end-to-end security by enforcing protected execution of authenticated code, confidentiality, authenticity, system integrity, and privacy.
Intel has designed a set of architecture extensions, called Intel Software Guard Extensions (Intel SGX), which aim to increase the security of software through an inverse sandbox mechanism. In this technology, a legitimate software can be sealed inside a secure enclave and protected from attacks, regardless of the privilege level of the attacker. Before the enclave is constructed, the enclave code and data are accessible for inspection and analysis. Once the applications protected portion of the code is loaded into an enclave, its code and data is measured and becomes protected against external software access. In Intel SGX model a CPU which can have many secure enclaves, where as in ARM TrustZone, the CPU works in two halves: the insecure world and the secure world.

Preliminary Self-Management Architecture

The objective of the security self-management approach is to build new generation of SUPERCLOUD systems that can manage themselves (i.e., Self-Managed) to reach and maintain high-level security objectives, previously defined by the cloud customer by means of SLAs (i.e., User-Centric). This approach calls for a security system that is highly independent of human intervention for the management of security properties. To address such a complex security management problem, we leverage in this preliminary architecture mechanisms used in *autonomic computing* (AC) [70]. Autonomic systems offer effective mechanisms that allow them to *monitor*, *control*, *regulate* and *recover* themselves from problems without external intervention [70].

In this Chapter, we describe this preliminary architecture and introduce basic building blocks for self-management.

Overview of Self-Management Architecture

In the this section, we provide an abstract overview of the Cloud Security Self-Management architecture depicted in Figure 4.2 below. In this architecture, *Cloud Service Customers* and *Cloud Service Providers* interact with a *SUPERCLOUD Front-end* (i.e., the Frontal) to express their requirements and constraints. Based on that, and eventually after a negotiation phase, a Service Level Agreement is established to define the expected quality of service and level of protection (cf. Section 3.1). This process is described in details in Section 4.2.1. Based on this SLA, a U-Cloud (Customer Specific Cloud) is created over a single or multiple providers as illustrated in the left part of Figure 4.1. The management and the control of this U-Cloud is then mandated to the Security Self-Management of the user.

The *Security Self-Management* is structured in three layers, namely the ***Orchestrations Layer***, ***The Aggregation Layer***, and the ***Resources Management Layer***.

- In the ***Resources Management Layer***, we can find *Self-Management Agents* that are responsible of delivering security atomic services such as *enforcement*, *detection*, *reaction* and *monitoring*. We assume that they are part of the SUPERCLOUD Hypervisor or the distributed visualization infrastructure (developed in SUPERCLOUD Workpackages 2, 3 and 4). As illustrated in the overview, these components operate on a particular layer of the visualization infrastructure (i.e., storage, compute and network) and are dedicated to a specific security service (e.g., Intrusion detection, Authorization enforcement, Trust Management). We assume also that some security services require multiple Self-Management Agents (SMAs) overs different layers. For instance, enforcement of *authorization* policies encompasses data, compute and storage layers as described in Section 4.2.3. A more detailed description of these *Self-Management Agents* is provided in Section 4.8.
- The ***Aggregation Layer*** provides an unified and uniform view of multiple SMAs to the orchestrator and abstracts the heterogeneity of lower layers. Thus this layer ensures the satisfaction of *heteregeneity* and *interoperability* requirements identified in Chapter 2.

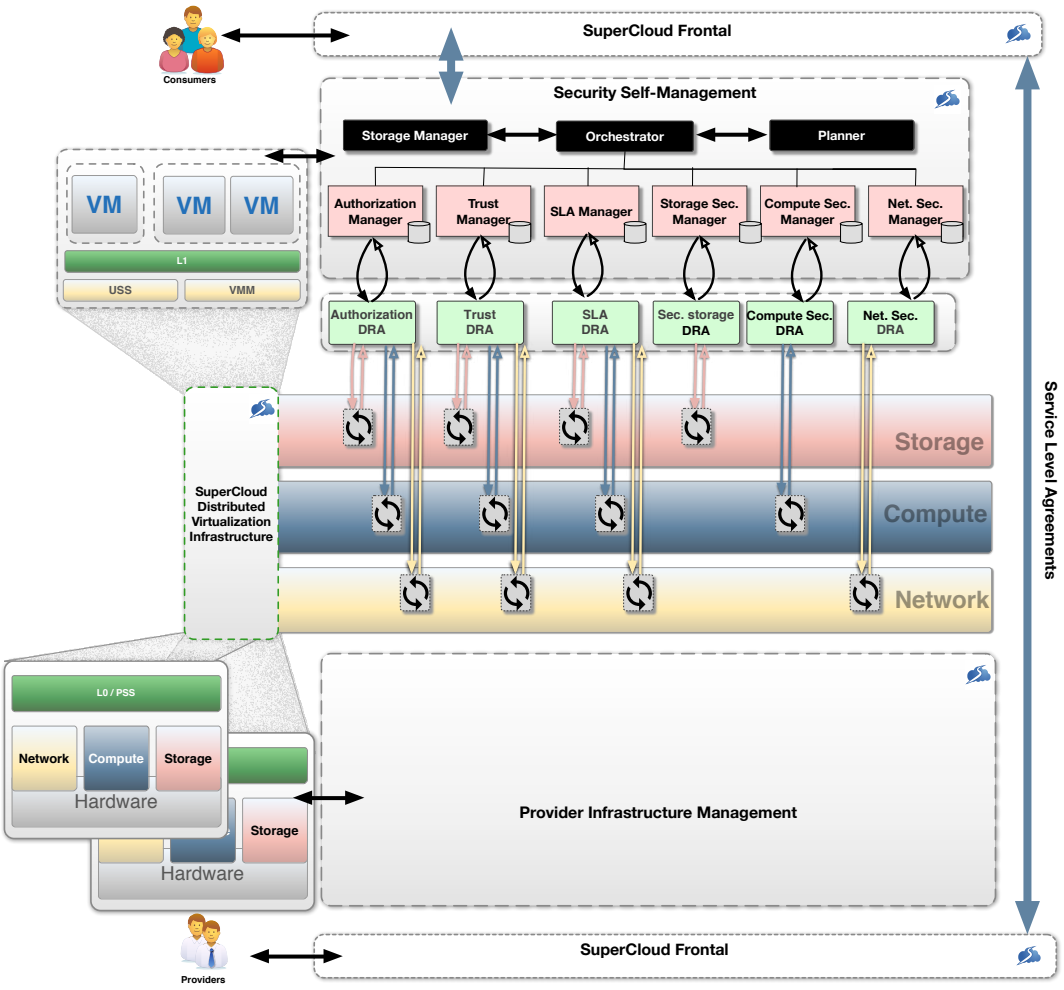

Figure 4.1: Self-Management of Security

- The ***Orchestration Layer*** encapsulates the decision making components that are in charge of providing security services. It is composed of several managers, each responsible of a specific security service such as *Authorization and Access Control* (cf. Section 4.2.3), Intrusion detection and prevision (cf. Sections 4.2.4 and 4.2.6)) or trust management (cf. Section 4.2.7). It contains also, an overall orchestrator that coordinates the actions of all security managers, a planner that generates plans to reach and/or maintain security objectives, and a storage manager that is in charge of the persistence and delivery of the knowledge that is necessary to achieve the self-management of security.

Self-Management Building Blocks

In this Section, we describe the principal building blocks we identified from the proposed architecture in order to provide user-centric control of cloud security. Based on the analysis of security requirements expressed in the uses cases, we identified four functional features that are essential to achieve the self-management of security, namely *Management of Security Service Level Agreement*, including *Security Service Level Agreements* (cf. Section 4.2.1), *Management of Access control and Authorizations* (cf. Section 4.2.3), *Management of Trust* (cf. Section4.2.7), and finally the *Management of Security Self-Services* (cf. Sections 3.2) at different levels of the virtualisation infrastructure, namely compute (cf. Section 4.2.5), storage (cf. Section 4.2.4) and network (cf. Section 4.2.6).

Security Service Level Agreement Management

Ensuring Service Levels is a key issue for successful deployment of cloud infrastructures. To that aim, Service-Level Agreements are nowadays a common way to formally specify the expected level of (functional and non-functional) quality under which cloud services shall be delivered. No matter if theses agreements are legally binding contracts or not, SLAs provide a high level of confidence that a commitment can be met (cf. Section 3.1). In this section, we provide a preliminary design specification for the management of Service Level Agreements with respect to, not only Quality of Service, but also and mostly on Quality of Protection.

As discussed in Section 3.1, Quality of protection is used in the literature (cf. [49]) in reference to security mechanisms that a provider is able to implement for the clouds that runs on its infrastructures. In SUPERCLOUD, we propose a joint management of Quality of Service and Quality of Protection within a unique and uniform Security Service Level Agreement (SSLA).

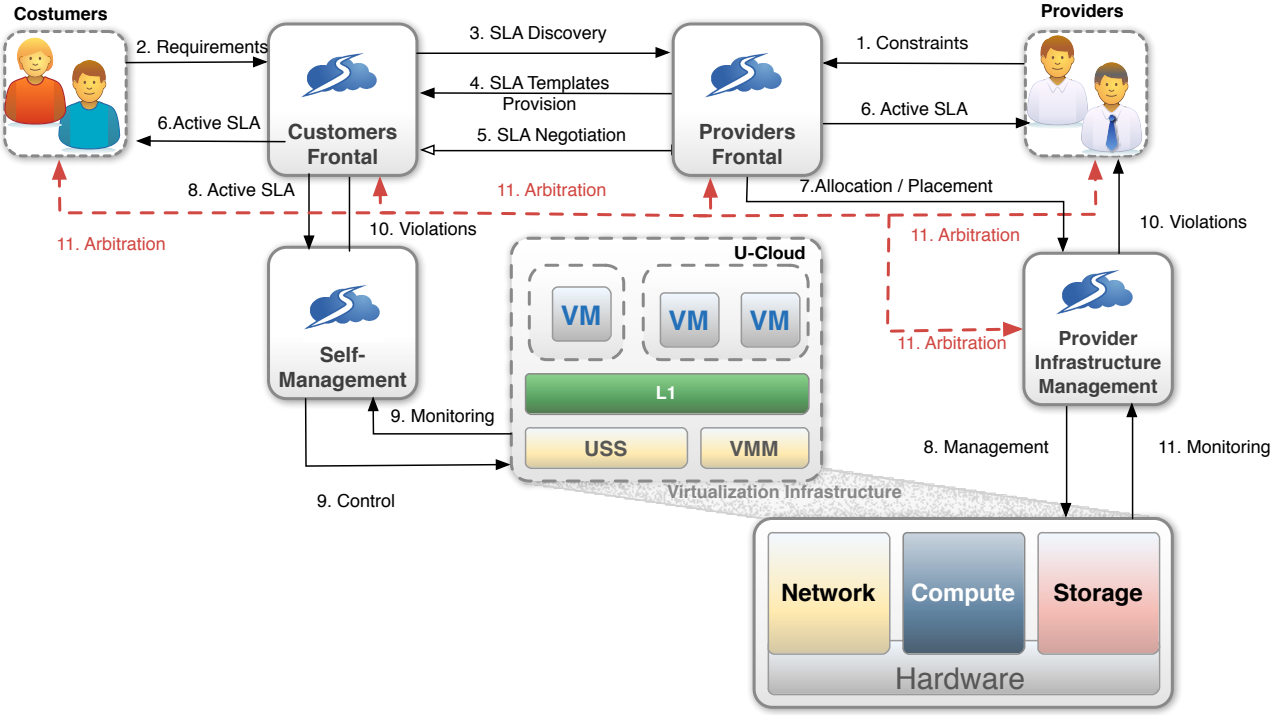

Figure 4.2: The Security Service Level Agreement Management Process

Figure 4.2 illustrates the overall vision of how the Security Service Level Agreement is established between Customer and Providers. In this process, we identify five principal actors that we present hereafter:

- **A Cloud Service Customer (CSC)** is any person, organization or entity that uses Cloud Services.
- **Cloud Service Providers** are persons, organizations or any entity responsible for making Cloud Services (Compute, Network and/or Storage) available to CSCs.
- **SUPERCLOUD Frontal** is the entity that helps both CSCs and CSPs lay out their requirements and constraints. It negotiates relationships between them trying to find the best agreement. With respect to classical Cloud Architecture, the frontal plays also the role of Broker in

advertising offers and managing cloud services discovery[1]. In the Figure 4.2, we make a distinction between *Providers* and *Customers Frontal*, this distinction is not always necessary as we can imagine configuration with a unique *Frontal* managing both *CSC* and *CSP*.

- **Customer Self-Management** allows customers to have a complete control over their U-clouds towards the fulfilment of performance and security objectives. This component has been presented in the previous section (cf. Section 4.1) and will be described in detail in the next Sections.
- **Provider Infrastructure Management** is the entity responsible of managing the Provider's Cloud infrastructure. This component operates at, both, the virtualisation level and the hardware level. The description of this components is out of the scope of this deliverable.

SSLA Process

In SUPERCLOUD, Providers offer their resources (i.e., network, compute and storage) as a service (i.e., Infrastructure as a Service). The process described hereafter presents part of the *end-to-end* management of *user-centric* security requirements trough SSLAs.

1. Providers register at the broker, embodied in the *SUPERCLOUD Frontal*, in order to make their services available to CSC. To that aim, each provider will propose an SLA template for each configuration they can provide.
2. Customers express their requirements by describing the service they desire, not only in terms of quality of service but also in terms of security requirements (i.e., Quality of Protection). The Costumers can have already their SLA template/offer ready but SUPERCLOUD can also provide assistance mechanisms to help them formalizing their needs (e.g. pre-filled SLA templates and a GUI to personalize these templates).
3. Based on the requirements expressed by the CSC, the SUPERCLOUD Frontal will launch a discovery process in order to find compatible SLA templates proposed by registered CSP. This phases may imply a composition of several offers.
4. The SUPERCLOUD creates then an SLA offer based on the best SLA template candidate(s). The SLA offer adapts the Service Level Objectives in order to meet the Customers needs. Often, a negotiation is conducted in order to reach an agreement that reduces that gap between the requirements of the CSC and the constraints of CSP(s).
5. If an agreement is reached, both CSP(s) and the CSC are notified with the terms of the SLA. Otherwise, another SLA template is selected and the process repeats until an agreement is reached[2].

Preliminary Security SLA architecture

In this section, we present a preliminary architecture of the Security Service Level Agreement framework that we derived from the SSLA establishment process presented in the previous section.

The management of SSLAs comprises multiple components that are scattered among four of the elements of the architecture presented in Figure 4.1. Theses components are *SUPERCLOUD Frontal* (Provider and Customer sides), the *Security Self-Management* and the *Provider Infrastructure Management*. Due to the preliminary stage of this architecture, we will describe the functioning of each

[1] We assume that the Cloud Broker is part of the frontal, we can imagine complex configuration in which several Brokers are involved in the process

[2] We can also imagine that after several attempts, Customers and Providers are asked to relax their requirements/ constraints.

component rather than focusing on specification details that will be refined later in the project.

We identified six different components in order to comply with the SLA life-cycle phases identified and discussed in Section 3.1. In what follows, we present each component starting from the ones integrated at *SUPERCLOUD Frontal* level and finishing by the assumed components that should be present at the *Provider Infrastructure Management*[3].

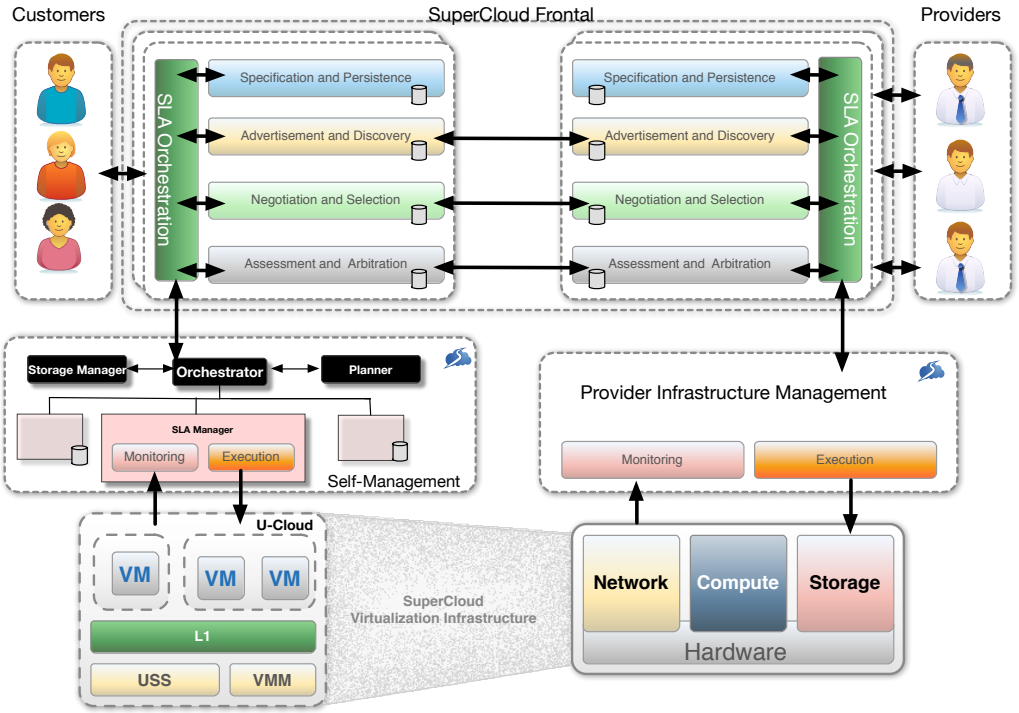

Figure 4.3: Preliminary Security Service Level Agreement Architecture

SLA orchestration

With respect to the SLA life-cycle, *orchestration* is an additional step that allows the coordination of the other phases. The SLA orchestrator manages the overall SLA lifecycle and serves as a central intermediaire between the CSCs/CSPs and the other SSLA components.

Specification and Persistence

Specification may include a graphical Web based interface that allow both Service Providers and Service Customers to interact with the SUPERCLOUD frontal. The main focus of these interactions consists in the definition of SLA templates/offers that encapsulate their requirements and/or constraints. The specification can also assist CSC and CSP defining the negotiation and/or arbitration strategies that will be used by the *SUPERCLOUD Frontal* on their behalf.

In addition, we consider that this component is also in charge of the *persistence*. It abstracts the repositories that are used by each components by providing the basic functionality to store, retrieve and delete SLAs[4] and any data that is necessary for the good execution of SLAs.

[3]As the specification of this later elements did not start yet, we can only assume that such components could be integrated, even if it appears that there is no burden to such integration in private clouds. In presence of public clouds, we imagine that we can still achieve this objective using virtualized monitoring components.

[4]We use interchangeably SLA and SSLA as they refer to the same concept in this Delivrable.

Advertisement and Discovery

The advertisement and discovery of SLAs are complementary phases. CSC providers advertise their SLA templates to propose their services to CSCs. The advertisement can be achieved using a *Broker* or via syndication to a registry. Similarly, CSC that looks for Cloud Service offers can search for compatible templates using registry or via Brokers. We assume also that in some scenarios, *SUPERCLOUD* can play the role of virtual *cloud services marketplace* in which CSC and CSP can advertise and discover cloud services.

An interesting and important feature of the *Advertisement and Discovery* component would be the management of cloud service compositions based on multiple-providers' offers. With respect to that, QoS-ware and QoP-aware composition of cloud services (e.g., [37, 118]) should be carried out. This makes the discovery process more complex, but also more efficient for the fulfilment of both CSC and CSP requirements/constraints.

Negotiation and Selection

The negotiation module is an important component of the SLA management. It executes state-of-the-art methodologies and protocols to reach the optimal agreement for both CSCs and CSPs. We assume that one instance of the SLA negotiation module is settled on each instance of SUPERCLOUD Frontal. We can also consider a centralized configuration in which a unique SUPERCLOUD Frontal manages all interactions in endogenous way.

At this stage of the project, we intend to extend XeNA [4] Language and Framework to manage SLA negotiation. The objective ot the extension will focus on allow XeNA to negotiate Service Level Agreement instead of Access Control decision and policies as it is the case in the current version of the prototype. XeNA [4] have proved to be flexible and extensible enough to express a large kind of negotiation strategies.

The Selection is the logical consequence of a successful negotiation. Thus the outcome of a negotiation should be an SLA. This SLA is thereafter executed.

Execution

At this stage, the SLA is executed and mechanisms to maintain it are deployed. The execution of an SLA is generally the responsibility of the Cloud Service Provider. However, in multi-cloud scenarios in which the agreement involves one CSC and multiple CSP, we need that the *Brokering* functionalities proposed by the *Frontal* include dispatching of configurations among the selected *CSPs*.

Furthermore, with the *User-Centric* management advocated in SUPERCLOUD, we assume that part of the execution should be carried out by the *Security Self-Management* component. For instance, the configuration of *Access control* policies and the specification and evaluation of trust requirements are managed by the Customer via the this Security Self-Management element.

Concretely, at the level of Service Management, the execution of an SLA imply the extraction of Service Level Objectives from the agreed SLA. These objectives are then converted into concrete operations to be enforced by management components (e.g., Self-Management). This step involve the planner and is coordinated by the *security Orchestrator* (cf. Section 4.2.10.

Monitoring

Monitoring SLAs is a key element for the assessment phase of the SLA life-cycle (cf. Section 3.1). This element is responsible for collecting data from different proves (cf. description of Self-Management

Agents in Section 4.2.8) to allow both CSP and CSC to supervise the status and performance of Cloud resources. It provides QoS and QoP measurements by abstracting a unified management interface for different types of external proves provided by Self-Management Agents (i.e., at compute, storage and network level).

Authorization management

As discussed in Deliverable 2.1, authorization in SUPERCLOUD is performed by several complementary visions and levels of authorization. Central is the notion of *tenant*, which may range from a usage profile over resources, to an access control group including several users, or to simply a user. Authorization may be:

- *Usage control oriented (A):* the user requests a security SLA to the SUPERCLOUD which will instantiate the U-Cloud accordingly after negotiating with underlying providers. A tenant is then instantiated, giving usage rights to resources/VMs running within the tenant. This is addressed by the OrBAC authorization framework [66, 35, 13].
- *Access control oriented (B):* the authorization component controls access to different VMs running in the tenant according to an access control policy. This is addressed by the MOON authorization framework [89].
- *Application-level (C):* the authorization component controls access to resources between different applications. This is addressed by a third authorization framework.

More detailed descriptions of such frameworks may be found in Deliverable D2.1. To illustrate, we give a brief overview of the OrBAC framework, sketching how it could be adapted and integrated in the SUPERCLOUD Self-Management Architecture. We then show how authorization management at the application-level is performed based on role-based access control, for instance as provided by the OrBAC framework.

OrBAC Authorization Framework

Organization Based Access Control (OrBAC) aims at modelling security policies that are centered on organizations. Intuitively, an organization is any entity responsible for defining and/or managing a security policy. Hence a company is an organization but security components such as firewalls or virtual machines managers can also be modeled as organizations. An OrBAC policy specification is done at the organizational level, also called the abstract level, and is implementation-independent. The abstract level defines the concepts of organization, role, activity, view, context and security rule to express the abstract policy. The policy to enforce, called the concrete policy, is inferred from the abstract policy. Through the assignments of subjects to roles, actions to activities and objects to views, concrete security rules are derived. This approach makes all the policies expressed in the OrBAC model reproducible and scalable. Actually once the concrete policy is inferred, no modification or tuning has to be done on the inferred policy since it would possibly introduce inconsistencies. Everything is done at the abstract policy specification level. The abstract policy, specified at the organizational level, is specified using *roles*, *activities* and *views* which respectively abstract the concrete subject, actions and objects.
The OrBAC model uses a first order logic formalism with negation. However since first order logic is generally undecidable, OrBAC is restricted in order to be compatible with a stratified Datalog program [43].
The OrBAC framework architecture is composed of three elements; (i) the OrBAC API, (ii) the *MotorBAC Tool* and (iii) the OrBAC Plug-ins, as show in Figure 4.4 below. This is mainly a conceptual view of the OrBAC infrastructure, as the three components do not necessary require to be on one single system. In the following a short description of the OrBAC components is given.

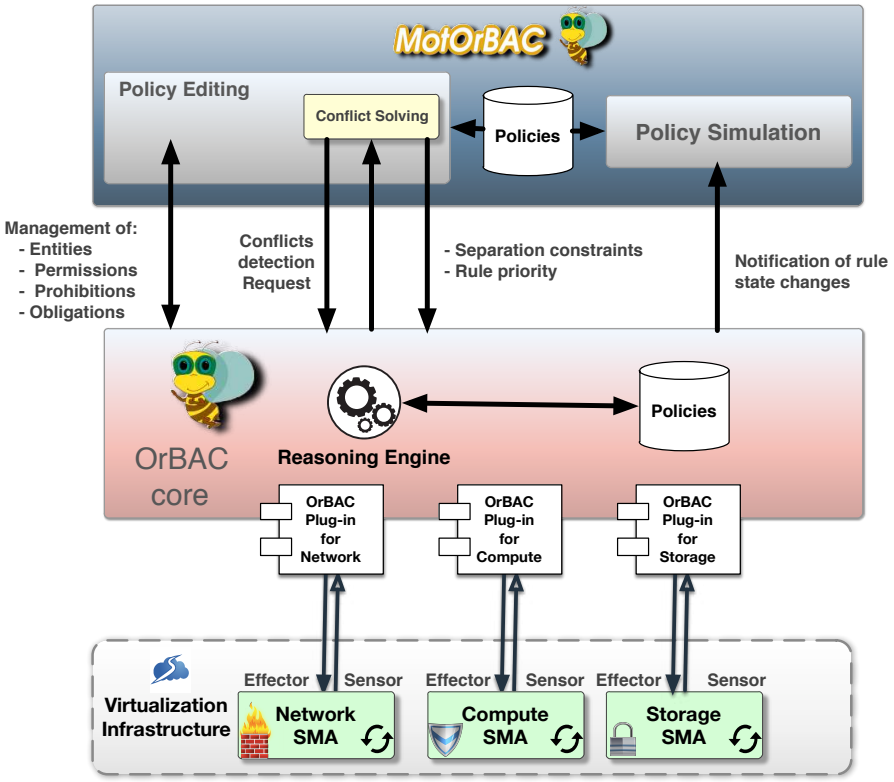

Figure 4.4: Architecture of the OrBAC framework

MotOrBAC

MotOrBAC is a tool that allows users to specify and simulate their policies. The tool is built on top of the OrBAC API which can be integrated to applications or cloud layers to interpret OrBAC policies. MotOrBAC can be run in normal or administrator mode. In normal mode, the users can create, manage and simulate policies. In addition to these features, the Administration mode allows users with higher privileges to assign and delegate roles. Consider for instance a task that allows the affectation of subjects to role or subject is permitted to affect subjects to roles.

OrBAC API

OrBAC Application Programmer Interface (OrBAC API) is the core component of the OrBAC framework. It provides also a Java library which can be used to programmatically create and manipulate OrBAC policies within Java Applications. The API features the following OrBAC policy editing capabilities:

- abstract policy specification: organizations, roles, activities, views, contexts, and abstract rules (permissions, obligations, prohibitions) can be manipulated. This includes organizations, roles, activities, and views hierarchies
- separation constraints and rules priorities can be specified to solve conflicts between abstract rules
- several languages can be used to express contexts and entity definitions. Simple ad-hoc languages have been defined to express temporal conditions or simple conditions on concrete entities (subject, action or object) attributes. Two more powerful languages can be used, Java and Prolog, to be able to express a wide range of conditions
- the administration policy, or AdOrBAC policy, associated to an OrBAC policy can be specified using the same concepts and API methods

The current version of the API provides three implementations:

- a Jena based implementation which stores the abstract policy and the associated concrete entities in a Resource Description Framework (RDF) file. The Jena reasoning engine is used to infer the concrete policy
- an implementation which stores the abstract policy and the associated concrete entities in a XML file. A custom multi-threaded backward chaining algorithm is used to infer the concrete policy. The fork/join framework from the JDK7 is used to implement multi-threading
- the last implementation derives from the second implementation with a focus on the persistence of concrete entities. In this implementation, the abstract policy is still represented in XML whereas the concrete entities are stored in a MySQL database.

OrBAC Plug-ins

Neither the MotOrBAC nor the OrBAC API do include any functionality to deploy and/or enforce OrBAC policies. To that aim, the plug-ins have been proposed to extend the OrBAC infrastructure with deployment mechanisms. Currently a short list of publicly available OrBAC plug-ins exists but this list should be extended with SUPERCLOUD specific plug-ins. Plug-ins are implemented as OSGi[5] bundles. The Equinox[6] implementation of the OSGi R4 core framework specification[7] is used to manage the bundles. An alternative solution would be the integration of the API into an ad-hoc deployment tool.
In the context of SUPERCLOUD, both plug-ins and API integration could be considered for the enforcement of authorization decisions.
As illustrated in Figure 4.4, plug-ins can be developed for each layer of the virtualization infrastructure. For instance, at the network layer, a concrete use of *Orbac* in conjunction of Software defined Network (SDN) controller is described in Section 4.2.6. The use of plug-ins at the Storage and/or compute level can be analogously described.

Application-Level Authorization

In Deliverable D2.1, a conceptual model of the SUPERCLOUD application-level authorization framework is given, in which *users*, through their *role* in an *organization*, get access to *applications* that can access *APIs* on their behalf. We now describe the implementation of this authorization framework using the SUPERCLOUD self-management architecture.
For concreteness, in Figure 4.5 we show a possible application-level authorization flow. In this diagram, the "IAM (identity and access management) Server" acts as authorization self-management manager responsible for evaluating the authorization decision of whether a given user has access to a certain application (specifically, version of the application). To this end, the Application Version asks the IAM server to authorize the use of a certain API scope on behalf of the user. The IAM server asks the user to log in, and verifies the correctness of the given credentials, and whether they allow access to the given app scope. This latter decision is based on role-based access control and can for instance be implemented as an OrBAC plugin (see previous section).
When applicable, the user is requested to give explicit consent for the application to access the API scope. The IAM returns the session information to the Application Version, which acts as an enforcement agent (specifically, in "forward mode", cf. Section 2) by checking the session information.
The Application version authorizes to the API using a shared secret (not shown in the figure), while also provides information about the authenticated user in the form of a token. The API acts as a "forward mode" enforcement agent by contacting the IAM server to obtain the session related to the

[5] http://www.osgi.org
[6] http://www.eclipse.org/equinox/
[7] http://www.osgi.org/Specifications/HomePage?section=2#Release4

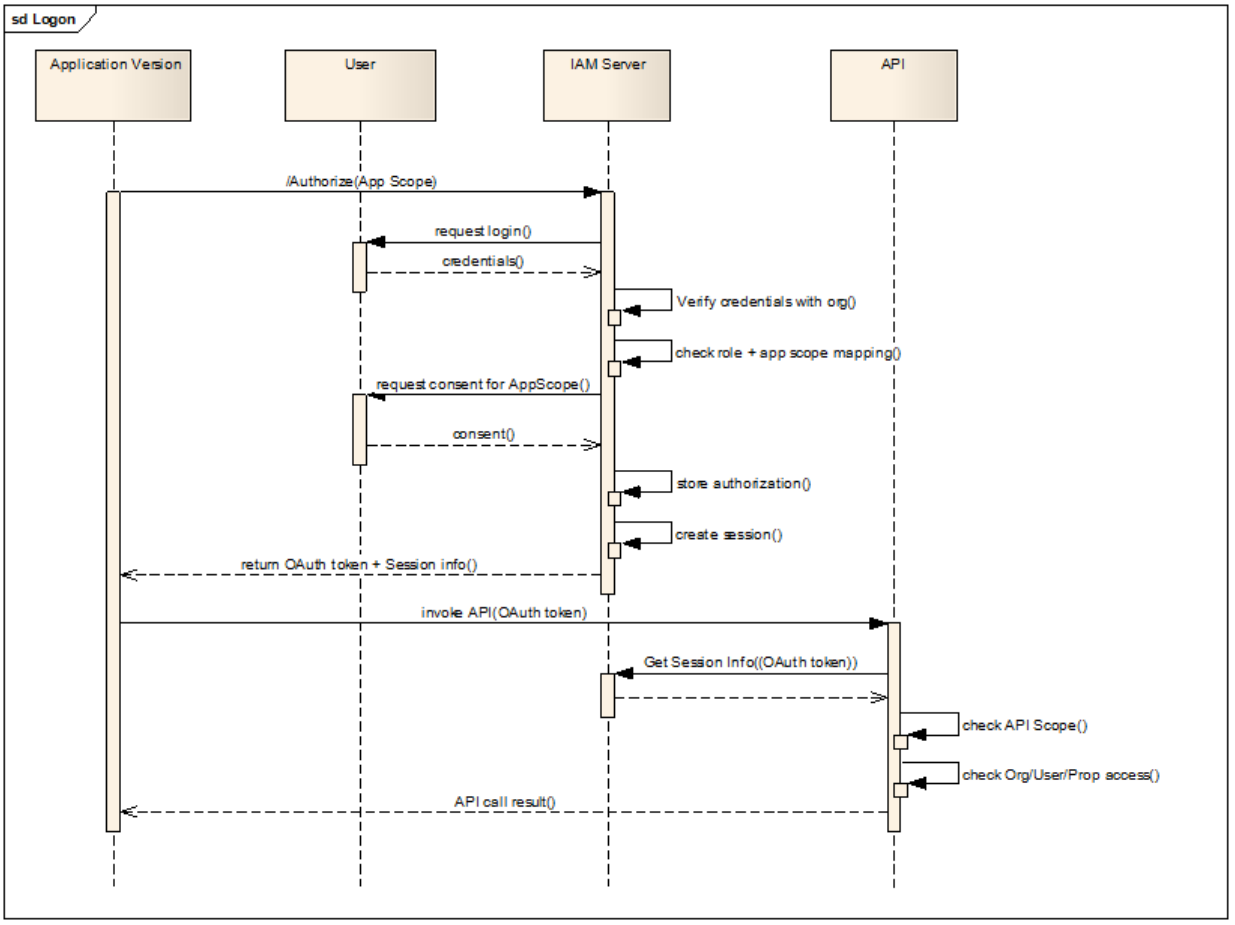

Figure 4.5: Flow diagram for application-level authorization

token, and checking if the token was actually correctly issued by the IAM server. Additionally, it makes an API-dependent decision of whether to allow the particular API call in an API-dependent way. (This is out of the scope of the authorization framework, but could again be implemented as an OrBAC plugin if the API uses role-based access control to protect the resources it exposes.)
Summarizing, from a self-management point of view, the IAM server acts as self-management manager, and the application version and API act as "forward mode" enforcement agents. Note that, as discussed in Chapter 2, one import requirement in authorization is "location self-management", meaning that location is an important factor in authorization decisions. For such decisions, an additional "location monitoring sensor" would provide input to the self-managent manager.

Compute Security Manager

For computation, self-management is performed along two dimensions, according to the multi-layer and multi-provider nature of the SUPERCLOUD infrastructure:

- *Vertical self-management* addresses autonomic management of security within and *across layers* of the distributed cloud. Security automation may apply to different levels of the SUPERCLOUD computation infrastructure defined in Deliverable D2.1 (e.g., VMs, user-level virtualization, user-controlled modules in provider infrastructure). Control is captured through orchestrated autonomic security loops, with probes for monitoring and effectors for modify layer security states. Autonomic security managers called *Layer Orchestrators* (LO) capture the decision intelligence. A *Vertical Orchestrator* (VO) performs cross-layer self-protection through LO coordination. Different VOs may in turn be orchestrated vertically through an *Arbiter* component, playing the

role of a root VO, to capture and enforce trade-offs between user and provider control over security, or between security and other concerns (e.g., autonomous management for energy efficiency).

- *Horizontal self-management* addresses *multi-provider* autonomic management of security. Security automation is considered independently from the underlying providers, using as touch points the VOs defined previously (typically one per provider), to provide a homogeneous level of security across clouds. Different alternatives may be considered in terms of control. A unique *Horizontal Orchestrator* (HO) with a centralized autonomic security loop for all providers for maximized control, but poor scalability and dependability. HOs may also *collaborate* using different patterns: peer-to-peer interconnection for scalability, broker for fine-grained control over interoperability, or hierarchical control for higher level of optimality for the security response.

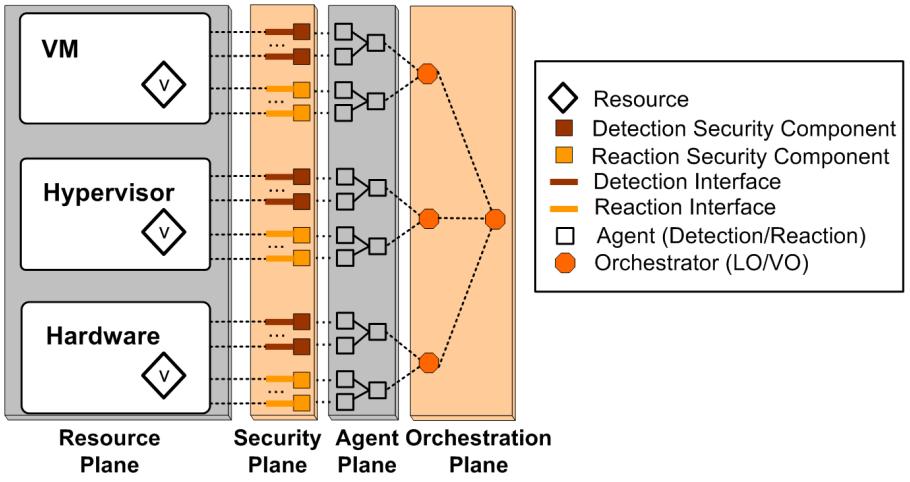

Figure 4.6: VESPA Architecture

More details on the self-management architecture for computation may be found in Deliverable D2.1. To implement this two dimensional self-management architecture within a Compute Security Manager, we use the VESPA (Virtual Environments Self-Protecting Architecture) framework [117, 74]. VESPA regulates protection of resources through several coordinated autonomic security loops which monitor the different infrastructure layers. The result is a very flexible approach for self-protection. Its main features are: (1) policy-based security adaptation based on a self-protection model capturing symmetrically and flexibly both detection and reaction phases; (2) two-level tuning of security policies according to security contexts both inside a software layer and across layers; (3) flexible orchestration of layer-level self-protection loops using system-wide knowledge to allow a rich spectrum of overall infrastructure self-protection strategies; and (4) a layered, extensible architecture allowing simple integration of commodity detection and reaction components thanks to an agent-based mediation plane abstracting security component heterogeneity.
The VESPA architecture contains: (1) a *resource plane*, with the resources to protect; (2) a *security plane*, with components delivering security services, for detection and reaction; (3) an *agent plane*, performing mediation between security services and decision-making elements; and (4) an *orchestration plane*, with autonomic management components and administrator-defined policies. Security detection components monitor protected resources, generating alerts. A hierarchy of *Detection agents (DA)* collects, filters, correlates such alerts, forwarding aggregated security information to the orchestration plane. Based on administration-defined strategies, high-level reaction policies are then generated to respond to detected threats. Another hierarchy of *Reaction agents* (RA) refines such policies, so that they may be applied to low-level security reaction components. In the end, such components modify protected resource state and behavior accordingly.

VESPA is virtualization-aware, considering resources in different infrastructure layers, for straightforward mapping to cloud architectures. Two types of autonomic managers called *Orchestrators* define the decision-making logic, in and between layers. *Layer Orchestrators* (LO) perform layer-level security adaptations. A *Vertical Orchestrator* (VO) is in charge of cross-layer security management: security events detected in one or several layers may trigger reactions in one or several other layers. With VESPA, vertical self-management may be delegated to the control of a single VO. Through LOs, the VO is also aware of all layer information in the U-Cloud. This design allows both intra-layer and cross-layer automated security supervision of a U-Cloud. Horizontal self-management may be implemented by deploying several VOs working in close cooperation, with several possible VO-to-VO communication patterns, master-slave, or fully peer-to-peer. A VO interprets and enforces security policies on resources in its domain, propagating to neighboring domains changes in reaction policies and or security state. Ongoing work is to extend VESPA to enable embedding in the SUPERCLOUD architecture and to implement cross-layer trust management (see Deliverable D2.1).

Storage Security Manager

SUPERCLOUD will allow Cloud Services Customers (CSC) to specify security requirements with respect to state-of-the-art data protection mechanisms such as cryptography and fault tolerance. The policy based approach advocated in the project will allow the automatic extraction of theses requirements and their conversion into security policies to be automatically executed on behalf of the CSC. This last step shall be the role of the *Storage Security Manager*. For instance, CSC can specify the desired fault tolerance mechanisms, then the Storage Security Manager will ensure the compliance of the running configuration with his needs. We envision also to use Attribute-based encryption to emulate self-protected Data. For more details on Fault Tolerance and Attribute-based Encryption mechanism, we refer the reader to Section 3.3.2 of Deliverable D3.1.

Network Security Manager

This section describes preliminary Self-Management mechanisms at the network level.

To achieve a user-centric management of network security, the SUPERCLOUD needs to satisfy two design specifications. First of all, the network abstraction layer will provide appropriate SDN control applications that allow users to compose their own security services in the data plane, and to tune these services on a per-flow (e.g. web traffic) or per-destination (e.g. towards a database service) basis. Second, users will be able to define their own security management procedures that will be automatically triggered by the SUPERCLOUD in response to attacks and security incidents.

In order to meet the first design specification, we will explore available SDN controller APIs that make possible the implementation of an SDN security management application on top of the SUPERCLOUD network abstraction layer. This application provides an abstraction model that enables to represent all security services available in the data plane (e.g. IDS, Proxies, honeypots, anti-DDoS). It further enables users to associate, on-demand, flows in the data plane with their appropriate security services. Last of all, this application will dynamically manage routing policies with respect to the security management procedures and the requirements that are introduced by the SUPERCLOUD users.

Regarding the second design specification, we will explore possible techniques to provide the SDN security management application with the ability to collect and process security incidents (e.g. standard alert formats such as IDMEF), to correlate them with other information related to the network topology, and then to trigger appropriate user-defined reaction strategies.

In what follows, we briefly present the how at the SND level we can execute DDOS attack mitigation strategy.

When attacks, intrusions or misconfigurations are detected, remediation actions should be rapidly and automatically performed. The enforcement of remediation strategies is performed by the reaction security service provided by the Self-Management Agent. The reaction action depends on the target SUPERCLOUD plane as it relies on the reaction API provided by this plane. One of the approaches that will be proposed in SUPERCLOUD is an SDN based attack mitigation strategy. Indeed, the recent emergence and rapid development of SDN offer to us an opportunity to re-examine and improve attack mitigation solution. Meanwhile, the computational overhead at the routers and switches can be significantly reduced, as large connection states or flow tables can be migrated and handled by the SDN. We envision that attack mitigation schemes can be effectively implemented and deployed at SDN controllers, paving a way for SUPERCLOUD cloud providers and network users to defend against attacks together by correlating and sharing the tasks of monitoring, analysis, detection and mitigation.

With this approach, OrBAC [66] based security policies will be designed for SDN controllers. These attack mitigation policies aim to react against the malicious behaviors. They will be triggered by alerts detected from the SUPERCLOUD monitoring module and therefore, new policies should be applied to react against the attack. These policies will be developed as applications for the SDN controllers that make reactions on an event basis and dynamically adapt security policies to handle suspicious and malicious flows. Then the policy changes will eventually lead to the automated configuration of the OpenFlow switches. Also, the end-to-end visibility yielded at the controller allows optimizing the deployment of middleboxes and the computation of flow paths with different QoS levels.

Finally, we note a more detailed description of how this remediation is triggered and manager could be found in the sections dedicated to Self-Management of Security in Deliverables D2.1, D3.1 and D4.1.

Trust Management

This module aims at the definition of a model to manage trust relationships across users and across providers. The formal specification of a trust model will be conducted in the future within a dedicated task. However, we have already identified core elements of the trust manager and its eventual interactions with the other self-management building blocks.

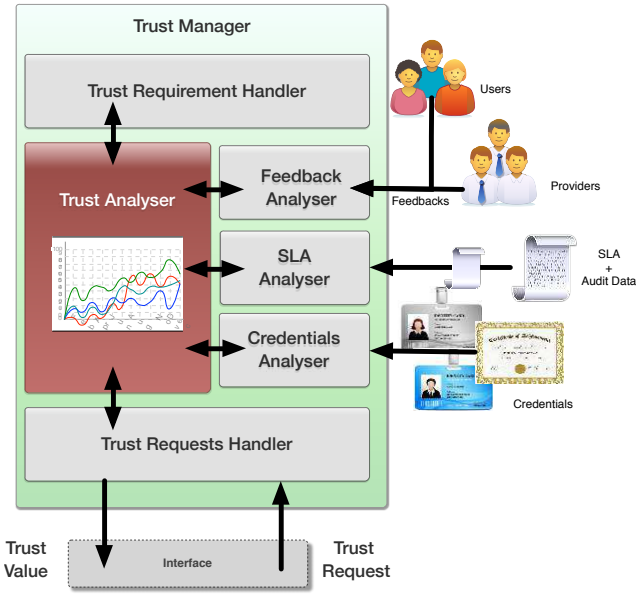

Figure 4.7: Specification of the Trust Manager

As depicted in the figure above, the trust manager is composed of six components that we describe

hereafter.

- **Feedback analyzer** is responsible of the collection and analysis of the feedbacks. Each feedback represent an opinion of a CSC or a CSP about another CSC/CSP. These opinions are then grouped and aggregated (e.g., using schemes reviewed in 3.3) to build a single "reputation" value. The way this reputation value is computed and the information needed to compute it will be specified in a dedicated task.

- **SLA analyzer** implements the SLA-based trust models presented previously (cf. Chapter 3). To that aim, this component is in charge of the extraction the active SLA's. It performs also local assessment of Service Level Objectives fulfilment. Alternatively, this component can rely on/collaborate with the *SLA manager* (cf Section 4.2.1) to perform this task. The result of this assessment reflects the degree to which a provider should be trusted. Trust here reflects the reliability of the provider.

- **Credentials analyzer** reproduces the policy-based or credential-based trust models. In theses models, trust is granted if the credentials that allows the satisfaction of a trust policy are provided. Trust policies generally express delegation of rights based on properties that should be provided using credentials (cf. Section 3.3 for more details). With respect to this scheme, this in charge of verifying the provided credentials and the derivation of a trust value based on these credentials. The exchange of credentials can also involve a trust negotiation process in which interacting parties try to reach an agreement about the minimal credentials that each party should release (cf. Section 3.3.4). If so, we assume that an *automated trust negotiation* module is required somewhere within the manager to allow such negotiation.

- **Trust requirements analyzer** is in charge of extracting and analysing trust requirements. Trust requirements are statements that specify under which conditions/constraints an entity (Provider or User) can be trusted for a particular issue. For instance, in the context of SUPERCLOUD we can suppose that the requirements used to trust a provider to guarantee service availability are not the same as the ones used for privacy preservation. To that aim, we propose to use *Trust Policies* [125] to express trust requirements. Trust policies are used to state what properties an interacting entity shall possess in order to be considered as trustworthy (cf. Section 3.3.3).

- **Trust analyzer** encapsulates the schemes used to derive trust based on *trust policies* and *trust information*. These schemes depend on the nature of the information to be processed and the specification language used by Cloud Customers to express their trust requirements.

- **Trust requests manager** orchestrates and coordinates the collaboration of the aforementioned components. It receives trust requests, triggers information collection, and queries the *trust analyzer* for trust computation. The result of trust evaluation is then transmitted to the initial requester.

Self-Management Agents

Self-Management agents are components that are responsible of delivering security atomic services such as *enforcement*, *detection*, *reaction* and *monitoring*. Theses "Touchpoint" agents can for instance encapsulate or interact with provider (or vendor) specific APIs to deliver the manageability interface to the self-management system. These agents are also the interfaces between Self-Management components and the other SUPERCLOUD components. To design theses agents, we draw inspiration from the IBM autonomic computing design pattern [69].
As depicted in the Figure 4.8 above, the manageability interface of Self-Management Agents is organized into *sensors* and *effectors* that we describe hereafter.

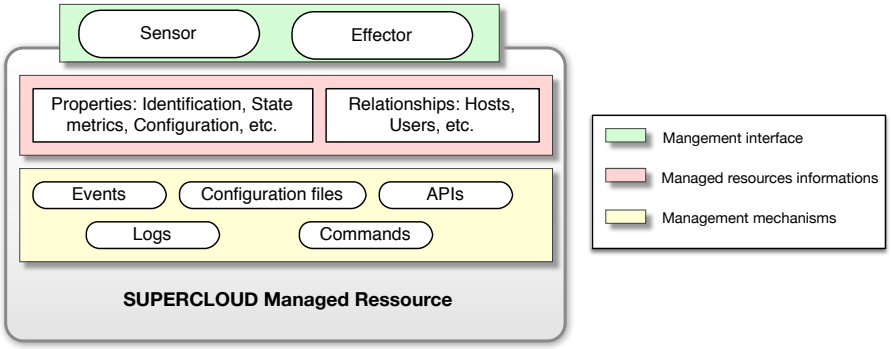

Figure 4.8: Abstract specification of Self-Management Agents (Touchpoints)

Effector

The effector is used to bring changes to the resource[8]. It makes use of the *manageability mechanisms* provided by the managed resources or their controller to execute the decisions made by higher level components such as the *manager* and/or the *orchestrator*. The *Effector* functioning is subsequently organized into *Enforcement* and *Reaction*.

Enforcement

Generally speaking, enforcement refers to the execution of security policies. In the context of SUPERCLOUD, enforcement consist in applying security configurations such as access control decisions, firewall rules or data encryption schemes. The enforcement role of the Self-Management Agent consist in providing configuration scripts to the security control API provided by each of the SUPERCLOUD planes (i.e., network, compute and storage).
The enforcement mechanisms will depend on the security services that are provided in each plane. The quality of a security service and its properties could be defined and negotiated within an SLA. For example, with respect to authorization management. Enforcement consist in applying the decision of the PDP encapsulated in the Authorization manager.
We identify two enforcement modes; *autonomous* and *forward*.

1. **Autonomous mode** refers to the ability of Self-Management Agents to build and enforce decisions made by their own. This means for instance that the Self-Management Agent embeds an additional reasoning and decision components with respect to the reference architecture presented in Figure 4.8. This component could be an *OrBAC API*, a MOON HOOK or an identity and access management (IAM) server (cf. 4.2.3).

2. **Forward mode** means that the Self-Management Agent will systematically forward all requests it receives to the upward relevant manager which is responsible of making decisions. The enforcement consist then in executing decisions made by the *authorization manager*.

In SUPERCLOUD, we consider that *forward mode* is the default enforcement implementation for *Self-Management Agents*. So the SMA will actively identify the manager concerned by the decision and query it for decisions making.

Reaction

When attacks, intrusions or mis-configurations are detected, remediation actions should be rapidly and automatically performed. The execution of remediation strategies is performed by the *reaction* security service provided by the Self-Management Agent. The reaction action depends on the target

[8]In SUPERCLOUD, we consider a broad notion of resource that inlcude any artefact at the level of network, compute and storage

SUPERCLOUD plane as it relies on the reaction API provided by this plane. In SUPERCLOUD, we adopt a policy-based reaction process. In other words, once an attack or an intrusion has been detected, the Security Self-Management will trigger the enforcement of new security rules. This means the activation of a *prohibition*, an *obligations*, a *permissions*, or any combination of them. A more detailed description of how this process along with a concrete scenario of DNS-based reaction could be found in Section 4.2.6 and in dedicated Self-Management sections in Deliverables D2.1, D3.1, and D4.1.

Sensor

The *Sensor* goal is to transmit states, events and properties of the managed resource to the Security Self-Management component such as the *manager* and the *orchestrator*. As illustrated in Figure 4.8, it also relies on the *manageability mechanisms* to extract information and transmit it to relevant recipients. A *Sensor* can also provide on-demand information to other components. For instance, activities *logs* can be processed to detect intrusions and attacks, then *commands* can afterward be used to execute the reactions. The *Sensor* is also subsequently organized into two sub-functions, namely *Monitoring* and *Detection*.

Monitoring

The role of this function is to report faults, anomalies or service disturbance, but also any other sensed information that can be obtained as well. It performs also measurements and performance models to monitor Service Level Objectives. For instance, from the location self-management requirement (Chapter 2), the need arises for a "location monitoring sensor" that checks if the requester of a particular resource is in a particular geographical location. If the SLA includes location awareness or location control, the location sensor is used to extract accurate geolocation to be forwarded to SLA manager in order to evaluate compliance with the active SLA. Many other types of sensors can be considered as any context information, functional and non functional measure can be obtained from the cloud infrastructure.

Detection

The detection activity allows the analysis of activities' logs to detect intrusions and/or attacks. Depending on the natures of the analysed logs, the detection can be classified into *Host Intrusion detection* (HID) and *Network Intrusion Detection* (NID) [41].

In *HID*, the analysis focuses on system, applications and services activities. For instance, Virtual machines activities can be obtained from the VMM and analysed. In the *NID*, the focus of the analysis in on the network packets that are exchanged between the SUPERCLOUD components (e.g., VMs, containers, applications, etc.). In both *HID* and *NID*, the analysis of these activities can be achieved in static and dynamic modes. The static analysis consist in comparing theses activities with the signature of well known attacks. In the dynamic analysis, the analysis is independent from any attacks base and relies on learning algorithms that tries to mine *normal* and *abnormal* activities.

Security Orchestration

In this Section, we briefly present how the security orchestrator coordinates the self-management process involving the different managers presented above. The orchestration involves three sub-components, namely the *Orchestrator*, the *Planner* and *Storage manager*. These components are described hereafter.

The orchestration approach we propose tries to mimic the MAPE-K model [69]. While *Monitoring* and *execution* are delegated to security-specific services managers such as *authorization manager*, the

Analysis is performed by the *Orchestrator*, the planning by the *planner* (cf. Section 4.2.11) and the knowledge is managed by the *Storage Manager* (cf. Section 4.2.12).

Orchestrator

The *Orchestrator* centralizes the decision making logic and coordinates the activities of Security Self-Management. At the initial stage, it retrieves and analyses the active SLA and requests the *Planner* to generate corresponding plans. Once these plans received, the orchestrator dispatches theses plans to the concerned managers to be executed. The executor (i.e., a manager) translates theses plans into formal descriptions of actions that Self-Management Agent will convert into target specific operations.

Planner

As highlighted above, the planner is responsible of extracting the user's description of the expected cloud configuration in terms of required components (and inter-connectivity), as well as the desired security and performance level. To that aim, the Planner will parse the active SLA to identify the configuration that need to be deployed by the provider in order to meet Security and Performance Service Level Objectives

Parsing the SLA will allow the Planner to extract parameters that will be used to generate deployment templates. These templates are used to allocate, place and configure the instantiated user cloud. The format of such configuration pattern is not decided yet. To that aim, we only rely on the description of parameters that could be used as *Service Level Objective*. Theses parameters has been described in Chapter 3.1.

Storage Manager

As its name suggests, this component is responsible of the knowledge base persistence. It stores rules, policies, plans, SLAs, and any data or information that is used in the Self-Management of Security. This component can be a database manager for instance or any efficient storage management mechanism.

Summary and Conclusion

Summary

This deliverable presented a preliminary architecture for SUPERCLOUD Security Self-Management infrastructure in multi-clouds.

In Chapter 2, we presented and analysed security requirements that should adhered to during the specification of Self-Management architecture. We first extracted Healthcare requirements from use-cases provided by SUPERCLOUD partners **Philips Healthcare** and **MAXDATA**. Then we described more generic requirements that may apply to any Multi-Cloud scenario. Based on these requirements analysis, we highlighted security mechanisms and services that should be implemented in order to bring *user-centric control* and *self-management* to current cloud architectures. These mechanisms include, but are not limited to, *fine-grained access control*, *flexible authorization management*, *efficient trust management* (that we split into two aspect *hard* and *soft* with orthogonal execution *horizontally* (across-providers) and *vertically* (across-layers). We identified also Security Service Level Agreements as a fundamental mechanism to enable *end-to-end* security customization and control.

In Chapter 3, we introduced basic concepts and existing approaches related to security services that have been identified previously (e.g., authorization, trust, SLA management, etc.). In this Chapter, we reviewed also different models, languages, and frameworks that have been proposed in the literature in *Access Control and Authorizaton*, *Service Level Agreements* and *Trust Management*. The objective of this Chapter was twofold; first we provided the reader basic concepts that we build upon in the specification of the *Self-Management* architecture. Second, we exacerbated the drawbacks and benefits of each approach in order be able to choose the one that best fits SUPERCLOUD requirements (cf. Chapter 2).

In Chapter 4, we presented a high-level architectural overview of the Self-Management architecture. Guided by the requirements, the specification of security self-management derive from the objective to design cloud systems that can manage themselves to reach and maintain high-level security objectives, previously defined by the cloud customer by means of SSLAs. We identified principal building blocks to reach this objective, then we delve in the description of each component. We showed also how each component, while implementing the techniques we reviewed Chapter 3 participates in the fulfilment for the requirements identified in Chapter 2.

Next steps

In the next months, the first steps will be to refine the proposed architecture to present the interfaces between the different building blocks and how this self-management architecture could be integrated to the SUPERCLOUD overall architecture, as well as the definition of the interfaces and necessary components for the *compute* (WP2), *storage* (WP3) and *network* (WP4) sub-architectures.

A challenging task would be the definition/extension of existing SLA specification language to express

security requirements and build on these languages to provide an automated SLA management framework.

Finally, a last important step will consist in implementing part of the self-management architecture to evaluate the validity of this approach. This step may lead to a re-evaluation, refinement and adaptation of some components of the architecture, making this deliverable a living document.

List of Abbreviations

ABAC	Attributes Based Access Control
API	Application Programmer Interface
CSC	Cloud Service Customer
CSP	Cloud Service Provider
EC	European Commission
HIDS	Host-based Intrusion Detection System
IAM	identity and access management
NIDS	Network Intrusion Detection System
OrBAC	Organization Based Access Control
RBAC	Role Based Access Control
RDF	Resource Description Framework
SLA	Service Level Agreement
SSLA	Security Service Level Agreement
SMA	Self Management Agent
SLO	Service Level Objective
TBD	to be determined
VESPA	Virtual Environments Self-Protecting Architecture
VM	Virtual Machine
WSLA	Web Service Level Agreement
XML	Extensible Markup Language

Bibliography

[1] Alfarez Abdul-rahman. The PGP Trust Model. *Architecture*, pages 1–6, 1997.

[2] Alfarez Abdul-Rahman and Stephen Hailes. Supporting trust in virtual communities. In *Proceedings of the 33rd Hawaii International Conference on System Sciences-Volume 6 - Volume 6*, HICSS '00, pages 6007–, Washington, DC, USA, 2000. IEEE Computer Society.

[3] Diala Abi Haidar, Nora Cuppens - Boulahia, Frédéric Cuppens, and Hervé Debar. XeNA: an access negotiation framework using XACML. *Annals of telecommunications - annales des télécommunications*, 64(1-2):155–169, October 2008.

[4] Diala Abi Haidar, Nora Cuppens Boulahia, Frdric Cuppens, and Herv Debar. Xena: an access negotiation framework using xacml. *annals of telecommunications - annales des tlcommunications*, 64(1-2):155–169, 2009.

[5] Joseph A. Akinyele, Matthew W. Pagano, Matthew D. Green, Christoph U. Lehmann, Zachary N.J. Peterson, and Aviel D. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '11, pages 75–86, New York, NY, USA, 2011. ACM.

[6] J.M. Alcaraz Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multi-tenancy authorization system for cloud services. *Security Privacy, IEEE*, 8(6):48–55, Nov 2010.

[7] Cloud Security Alliance. http://cloudsecurityalliance.org, 2011.

[8] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.

[9] Claudio A. Ardagna, Ernesto Damiani, Sabrina Capitani di Vimercati, Sara Foresti, and Pierangela Samarati. Trust management. In Milan Petković and Willem Jonker, editors, *Security, Privacy, and Trust in Modern Data Management*, Data-Centric Systems and Applications, pages 103–117. Springer Berlin Heidelberg, 2007.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[11] D. Artz and Y. Gil. A survey of trust in computer science and the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):58–71, June 2010.

[12] Michael Ault. *Oracle Administration and Management*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition, 2002.

[13] F. Autrel, F. Cuppens, N. Cuppens-Boulahia, and C. Coma. Motorbac 2: a security policy tool, 2008.

[14] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Monte Goode, and Kate Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, Gridshib, and MyProxy. In *5th Annual PKI R&D Workshop*, April 2006.

[15] M.Y. Becker and P. Sewell. Cassandra: Distributed access control policies with tunable expressiveness. In *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '04, pages 159–, Washington, DC, USA, 2004. IEEE Computer Society.

[16] K. Bernsmed, M.G. Jaatun, P.H. Meland, and A. Undheim. Security slas for federated cloud services. In *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, pages 202–209, Aug 2011.

[17] Karin Bernsmed, Martin Gilje Jaatun, and Astrid Undheim. Security in service level agreements for cloud computing. In Leymann et al. [16], pages 636–642.

[18] E. Bertino, E. Ferrari, and A. Squicciarini. X -tnl: An xml-based language for trust negotiations. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 81–, Washington, DC, USA, 2003. IEEE Computer Society.

[19] E. Bertino, E. Ferrari, and A. C. Squicciarini. Trust-x: A peer-to-peer framework for trust establishment. *IEEE Trans. on Knowl. and Data Eng.*, 16(7):827–842, July 2004.

[20] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The role of trust management in distributed systems security. In Jan Vitek and Christian D. Jensen, editors, *Secure Internet programming*, pages 185–210. Springer-Verlag, London, UK, 1999.

[21] Matt Blaze, Joan Feigenbaum, and AngelosD. Keromytis. Keynote: Trust management for public-key infrastructures. In Bruce Christianson, Bruno Crispo, WilliamS. Harbison, and Michael Roe, editors, *Security Protocols*, volume 1550 of *Lecture Notes in Computer Science*, pages 59–63. Springer Berlin Heidelberg, 1999.

[22] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *Proceedings of the 1996 IEEE Symposium on Security and Privacy*, SP '96, pages 164–, Washington, DC, USA, 1996. IEEE Computer Society.

[23] P.A. Bonatti, L Juri, Daniel Olmedilla, and Luigi Sauro. Policy-driven negotiations and explanations: Exploiting logic-programming for trust management, privacy & security. *Logic Programming*, pages 779–784, 2008.

[24] Piero Bonatti and Daniel Olmedilla. Driving and monitoring provisional trust negotiation with metapolicies. In *Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '05, pages 14–23, Washington, DC, USA, 2005. IEEE Computer Society.

[25] Stefano Braghin. *Advanced languages and techniques for trust negotiation.* PhD thesis, University degli Studi dell'Insubria, 2011.

[26] Winston Bumpus, John W. Sweitzer, Patrick Thompson, Andrea R. Westerinen, and Raymond C. Williams. *Common Information Model: Implementing the Object Model for Enterprise Management.* John Wiley & Sons, Inc., New York, NY, USA, 2000.

[27] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Technical report, March 2005.

[28] E. Cayirci. A joint trust and risk model for msaas mashups. In *Simulation Conference (WSC), 2013 Winter*, pages 1347–1358, Dec 2013.

[29] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. on Knowl. and Data Eng.*, 1(1):146–166, March 1989.

[30] Suranjan Choudhury, Kartik Bhatnagar, and Wasim Haque. *Public Key Infrastructure Implementation and Design*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2002.

[31] Yang-Hua Chu, Joan Feigenbaum, Brian LaMacchia, Paul Resnick, and Martin Strauss. Referee: trust management for web applications. *World Wide Web J.*, 2(3):127–139, June 1997.

[32] Eve Cohen, Roshan K. Thomas, William Winsborough, and Deborah Shands. Models for coalition-based access control (cbac). In *Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies*, SACMAT '02, pages 97–106, New York, NY, USA, 2002. ACM.

[33] European Commission. The cloud service level agreement standardisation guidelines.

[34] Robin Cover. Extensible Access Control Markup Language (XACML), 2007.

[35] F. Cuppens, N. Cuppens-Boulahia, and C. Coma. MotOrBAC : un outil d'administration et de simulation de politiques de sécurité, 2006.

[36] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK, UK, 2001. Springer-Verlag.

[37] A.V. Dastjerdi and R. Buyya. Compatibility-aware cloud service composition under fuzzy preferences of users. *Cloud Computing, IEEE Transactions on*, 2(1):1–13, Jan 2014.

[38] D. Davide Lamanna, J. Skene, and W. Emmerich. Slang: a language for defining service level agreements. In *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*, pages 100–106, May 2003.

[39] S.A. de Chaves, C.B. Westphall, and F.R. Lamin. Sla perspective in security management for cloud computing. In *Networking and Services (ICNS), 2010 Sixth International Conference on*, pages 212–217, March 2010.

[40] Juri Luca De Coi and Daniel Olmedilla. A review of trust management, security and privacy policy languages. In *SECRYPT 2008, Proceedings of the International Conference on Security and Cryptography, Porto, Portugal, July 26-29, 2008, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*, pages 483–490. INSTICC Press, 2008.

[41] Herve Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems, 1998.

[42] John DeTreville. Binder, a logic-based security language. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 105–, Washington, DC, USA, 2002. IEEE Computer Society.

[43] Jeffrey D.Ullman. Principles of database and knowledge-base systems. In *Computer Science Press*, 1989.

[44] A. Duncan, S. Creese, M. Goldsmith, and J.S. Quinton. Cloud computing: Insider attacks on virtual machines during migration. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 493–500, July 2013.

[45] A.J. Duncan, S. Creese, and M. Goldsmith. Insider attacks in cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*, pages 857–862, June 2012.

[46] Rino Falcone and Cristiano Castelfranchi. Social trust: a cognitive approach. In Christiano Castelfranchi and Yao-Hua Tan, editors, *Trust and deception in virtual societies*, pages 55–90. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[47] Telemanagement (TM) Forum. Sla management handbook, volume 2: Concepts and principles release 2.5.

[48] Diego Gambetta. Can We Trust Trust? In Diego Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, chapter 13, pages 213–237. Department of Sociology, University of Oxford, 2000.

[49] Dieter Gollmann, Fabio Massacci, and Artsiom Yautsiukhin, editors. *Quality of Protection - Security Measurements and Metrics*, volume 23 of *Advances in Information Security*. Springer, 2006.

[50] Marc Goodner, Maryann Hondo, Anthony Nadalin, Michael McIntosh, and Don Schmidt. Understanding WS-Federation. Technical report, IBM, May 2007.

[51] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA, 2006. ACM.

[52] Tyrone Grandison and Morris Sloman. A survey of trust in internet applications. *Commun. Surveys Tuts.*, 3(4):2–16, October 2000.

[53] Tyrone Grandison and Morris Sloman. Trust management tools for internet applications. In *Proceedings of the 1st international conference on Trust management*, iTrust'03, pages 91–107, Berlin, Heidelberg, 2003. Springer-Verlag.

[54] Tyrone W. A. Grandison. *Trust Management for Internet Applications*. PhD thesis, Imperial College of Science, Technology and Medicine, University of London, 2003.

[55] S.M. Habib, S. Ries, and M. Muhlhauser. Towards a trust management system for cloud computing. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, pages 933–939, Nov 2011.

[56] Brian Hayes. Cloud computing. *Commun. ACM*, 51(7):9–11, July 2008.

[57] Amir Herzberg, Yosi Mass, Joris Michaeli, Yiftach Ravid, and Dalit Naor. Access control meets public key infrastructure, or: Assigning roles to strangers. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, SP '00, pages 2–, Washington, DC, USA, 2000. IEEE Computer Society.

[58] Andrew N. Hiles. Service level agreements: Panacea or pain? *The TQM Magazine*, 6(2):14–16, 1994.

[59] Polar Humenn. The formal semantics of XACML. Technical report, Syracuse University, 2003.

[60] Trung Dong Huynh, Nicholas R. Jennings, and Nigel R. Shadbolt. An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, September 2006.

[61] A. Jøsang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007.

[62] Audun Jøsang. Trust and reputation systems. In Alessandro Aldini and Roberto Gorrieri, editors, *Foundations of security analysis and design IV*, pages 209–245. Springer-Verlag, Berlin, Heidelberg, 2007.

[63] Audun Jøsang and Roslan Ismail. The Beta Reputation System. In *Proceedings of the 15th Bled Electronic Commerce Conference*, volume 160, pages 324–337, 2002.

[64] Lalana Kagal, Tim Finin, and Anupam Joshi. A policy based approach to security for the semantic web. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, volume 2870 of *Lecture Notes in Computer Science*, pages 402–418. Springer Berlin Heidelberg, 2003.

[65] A. A. El Kalam and Y. Deswarte. Multi-OrBAC : a New Access Control Model for Distributed, Heterogeneous and Collaborative Systems. In *8th International Symposium on System and Information Security (SSI'2006), Sao Paulo (Brésil), 8-10 Novembre 2006*, 2006.

[66] A.A.E. Kalam, R.E. Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, and G. Trouessin. Organization based access control. In *Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on*, pages 120–131, June 2003.

[67] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 120–, Washington, DC, USA, 2003. IEEE Computer Society.

[68] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[69] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, Jan 2003.

[70] Jana Koehler and C Giblin. On autonomic computing architectures. Technical report, IBM Research, Zurich, 2003.

[71] Y. Kouki, F.A. de Oliveira, S. Dupont, and T. Ledoux. A language support for cloud elasticity management. In *Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on*, pages 206–215, May 2014.

[72] Yousri Kouki and Thomas Ledoux. CSLA : a Language for improving Cloud SLA Management. In *International Conference on Cloud Computing and Services Science, CLOSER 2012*, pages 586–591, Porto, Portugal, April 2012.

[73] Karl Krukow, Mogens Nielsen, and Vladimiro Sassone. Trust models in ubiquitous computing. *Philosophical transactions of the Royal Society*, 366(1881):3781–3793, 2008.

[74] Marc Lacoste, Aurélien Wailly, Aymeric Tabourin, Loïc Habermacher, Xavier Le Guillou, and Jean-Philippe Wary. Flying over Mobile Clouds with Security Planes: Select Your Class of SLA for End-to-End Security. In *6th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2013.

[75] Pradip Lamsal. Understanding trust and security. Technical report, Department of Computer Science University of Helsinki, Finland, 2001.

[76] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169–180, 2009.

[77] Adam J. Lee. *Towards Practical and Secure Decentraliz Attribute-Based Authorisation Systems*. PhD thesis, University of Illinois, 2008.

[78] Adam J. Lee, Marianne Winslett, and Kenneth J. Perano. Trustbuilder2: A reconfigurable framework for trust negotiation. In Elena Ferrari, Ninghui Li, Elisa Bertino, and Yuecel Karabulut, editors, *Trust Management III*, volume 300 of *IFIP Advances in Information and Communication Technology*, pages 176–195. Springer Berlin Heidelberg, 2009.

[79] Chen-Yu Lee, K.M. Kavi, R.A. Paul, and M. Gomathisankaran. Ontology of secure service level agreement. In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, pages 166–172, Jan 2015.

[80] Herbert Leitold and Bernd Zwattendorfer. Stork: Architecture, implementation and pilots. In Norbert Pohlmann, Helmut Reimer, and Wolfgang Schneider, editors, *ISSE 2010 Securing Electronic Business Processes*, pages 131–142. Vieweg+Teubner, 2011.

[81] Ninghui Li, John C. Mitchell, and William H. Winsborough. Design of a role-based trust-management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, SP '02, pages 114–, Washington, DC, USA, 2002. IEEE Computer Society.

[82] Niklas Luhmann. Familiarity, confidence, trust: Problems and alternatives. In *Trust: Making and breaking cooperative relations*, pages 15–35. Basil Blackwell, 1990.

[83] Jesus Luna Garcia, Robert Langenberg, and Neeraj Suri. Benchmarking cloud security level agreements using quantitative policy trees. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 103–112, New York, NY, USA, 2012. ACM.

[84] Stephen Paul Marsh. *Formalising trust as a computational concept*. PhD thesis, Department of Computing Science and Mathematics, University of Stirling, 1994.

[85] Tim Mather, Subra Kumaraswamy, and Shahed Latif. *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. O'Reilly Media, Inc., 2009.

[86] D. Harrison Mcknight and Norman L. Chervany. The Meanings of trust. Technical Report 612, University of Minnesota, 1996.

[87] Marc Mosch, Stephan Gro, and Alexander Schill. User-controlled resource management in federated clouds. *Journal of Cloud Computing*, 3(1), 2014.

[88] Talal H. Noor and Quan Z. Sheng. Trust as a service: A framework for trust management in cloud environments. In *Proceedings of the 12th International Conference on Web Information System Engineering*, WISE'11, pages 314–321, Berlin, Heidelberg, 2011. Springer-Verlag.

[89] OPNFV Consortium. Moon Security Management Module, 2015. https://wiki.opnfv.org/moon.

[90] Simon Parsons and M. Wooldridge. Game theory and decision theory in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, pages 1–14, 2002.

[91] Adrian Paschke. Rbsla a declarative rule-based service level agreement language based on ruleml. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 308–314, Nov 2005.

[92] David Recordon and Drummond Reed. Openid 2.0: A platform for user-centric identity management. In *Proceedings of the Second ACM Workshop on Digital Identity Management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.

[93] N. Repp, A. Miede, M. Niemann, and R. Steinmetz. Ws-re2policy: A policy language for distributed sla monitoring and enforcement. In *Systems and Networks Communications, 2008. ICSNC '08. 3rd International Conference on*, pages 256–261, Oct 2008.

[94] Carlos Ribeiro, Andr Zquete, Paulo Ferreira, and Paulo Guedes. Spl: An access control language for security policies with complex constraints. In *In Proceedings of the Network and Distributed System Security Symposium*, pages 89–107, 1999.

[95] S. Ron and P. Aliko. Level agreement (sla) in utility computing systems.

[96] Sini Ruohomaa and Lea Kutvonen. Trust management survey. In Peter Herrmann, Valérie Issarny, and Simon Shiu, editors, *Trust Management*, volume 3477 of *Lecture Notes in Computer Science*, pages 77–92. Springer Berlin Heidelberg, 2005.

[97] J. Russell and R. Cohn. *Windows Identity Foundation*. Book on Demand, 2012.

[98] Tatyana Ryutov, Li Zhou, Clifford Neuman, Travis Leithead, and Kent E. Seamons. Adaptive trust negotiation and access control. In *Proceedings of the tenth ACM symposium on Access control models and technologies*, SACMAT '05, pages 139–146, New York, NY, USA, 2005. ACM.

[99] Jordi Sabater and Carles Sierra. Regret: reputation in gregarious societies. In *Proceedings of the fifth international conference on Autonomous agents*, AGENTS '01, pages 194–195, New York, NY, USA, 2001. ACM.

[100] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 26(11):9–19, November 1993.

[101] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. *Computer*, 29(2):38–47, 1996.

[102] K. Seamons, M. Winslett, T. Yu, B. Smith, E. Child, J. Jacobson, H. Mills, and L. Yu. Requirements for policy languages for trust negotiation. In *Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, POLICY '02, pages 68–, Washington, DC, USA, 2002. IEEE Computer Society.

[103] J. Skene, D. Davide Lamanna, and W. Emmerich. Precise service level agreements. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 179–188, May 2004.

[104] M Sloman. Policy Driven Management For Distributed Systems. *Journal of Network and System Management, Vol.*, 2(4), 1994.

[105] Joel Sommers, Paul Barford, Nick Duffield, and Amos Ron. Accurate and efficient sla compliance monitoring. *SIGCOMM Comput. Commun. Rev.*, 37(4):109–120, August 2007.

[106] Martin Spasovski. *OAuth 2.0 Identity and Access Management Patterns*. Packt Publishing, 2013.

[107] A. Squicciarini, E. Bertino, Elena Ferrari, F. Paci, and B. Thuraisingham. Pp-trust-x: A system for privacy preserving trust negotiations. *ACM Trans. Inf. Syst. Secur.*, 10(3), July 2007.

[108] Lili Sun, Hua Wang, Jianming Yong, and Guoxin Wu. Semantic access control for cloud computing based on e-healthcare. In *Computer Supported Cooperative Work in Design (CSCWD), 2012 IEEE 16th International Conference on*, pages 512–518, May 2012.

[109] G. Suryanarayana and R.N. Taylor. A Survey of Trust Management and Resource Discovery Technologies in Peer-to-Peer Applications. Technical Report UCI-ISR-04-6, ISR, 2004.

[110] Bo Tang, Qi Li, and R. Sandhu. A multi-tenant rbac model for collaborative cloud services. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 229–238, July 2013.

[111] Bo Tang and R. Sandhu. Cross-tenant trust models in cloud computing. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 129–136, Aug 2013.

[112] Bo Tang, R. Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. In *Collaboration Technologies and Systems (CTS), 2013 International Conference on*, pages 132–138, May 2013.

[113] The Standfod Center for Biomedical Informatics Research (BMIR). Protege: open source ontology editor and knowledge-base framework, 2000.

[114] Roshan K. Thomas. Team-based access control (tmac): A primitive for applying role-based access controls in collaborative environments. In *Proceedings of the Second ACM Workshop on Role-based Access Control*, RBAC '97, pages 13–19, New York, NY, USA, 1997. ACM.

[115] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 93–, Washington, DC, USA, 2003. IEEE Computer Society.

[116] Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Stuart Aitken. Kaos policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, July 2004.

[117] Aurélien Wailly, Marc Lacoste, and Hervé Debar. VESPA: Multi-Layered Self-Protection for Cloud Resources. In *International Conference on Autonomic Computing (ICAC)*, 2012.

[118] Olga Wenge, Dieter Schuller, Ulrich Lampe, Melanie Siebenhaar, and Ralf Steinmetz. Composition of cloud collaborations under consideration of non-functional attributes. In Xavier Franch, AdityaK. Ghose, GraceA. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 462–469. Springer Berlin Heidelberg, 2014.

[119] Wikipedia. Trust management (information system), 2013.

[120] Linlin Wu and Rajkumar Buyya. Service level agreement (SLA) in utility computing systems. *CoRR*, abs/1010.2881, 2010.

[121] Edward Wustenhoff. Service level agreement in the data center. Sun Blueprints, 2002.

[122] F. Xie, Z. Chen, H. Xu, X. Feng, and Q. Hou. Tst: Threshold based similarity transitivity method in collaborative filtering with cloud computing. *Tsinghua Science and Technology*, 18(3):318–327, June 2013.

[123] M Yagüe. Survey on xml-based policy languages for open environments. *Journal of Information Assurance and Security*, 1:11–20, 2006.

[124] R. Yaich, O. Boissier, P. Jaillon, and G. Picard. An agent based trust management system for multi-agent based virtual communities. In Yves Demazeau, Jörg P. Müller, Juan M. Corchado Rodríguez, and Javier Bajo Pérez, editors, *Advances on Practical Applications of Agents and Multiagent Systems, Proc. of the 10th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 12)*, volume 155 of *Advances in Soft Computing Series*, pages 217–223. Springer, 2012.

[125] Reda Yaich, Olivier Boissier, Gauthier Picard, and Philippe Jaillon. Adaptiveness and Social-Compliance in Trust Management within Virtual Communities. *Web Intelligence and Agent Systems (WIAS), Special Issue: Web Intelligence and Communities*, 11(4), 2013.

[126] Walt Yao. Trust management for widely distributed systems. Technical Report UCAM-CL-TR-608, University of Cambridge Computer Laboratory, 2004.

[127] Younis A. Younis, Kashif Kifayat, and Madjid Merabti. An access control model for cloud computing. *Journal of Information Security and Applications*, 19(1):45 – 60, 2014.

[128] Ting Yu. *Automated trust establishment in open systems*. PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2003. AAI3102006.

[129] Ting Yu, Xiaosong Ma, and Marianne Winslett. Prunes: an efficient and complete strategy for automated trust negotiation over the internet. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 210–219, New York, NY, USA, 2000. ACM.

[130] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security*, 6(1):1–42, February 2003.

[131] Eric Yuan and Jin Tong. Attributed based access control (ABAC) for Web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[132] Dimitrios Zissis and Dimitrios Lekkas. Addressing cloud computing security issues. *Future Gener. Comput. Syst.*, 28(3):583–592, March 2012.