A MODEL-DRIVEN APPROACH TOWARDS DESIGNING AND ANALYSING SECURE SYSTEMS FOR MULTI-CLOUDS

SHAUN SHEI

PhD 2018

A MODEL-DRIVEN APPROACH TOWARDS DESIGNING AND ANALYSING SECURE SYSTEMS FOR MULTI-CLOUDS

SHAUN SHEI

A thesis submitted in partial fulfilment of the
requirements of the University of Brighton
for the degree of Doctor of Philosophy

April 2018

The University of Brighton

Abstract

Cloud computing is a paradigm that is utilised by cloud service providers to offer computational resources as procurable services to consumers. Whereby security is a primary concern to both consumers and cloud service providers, as confidential data and processes are offloaded to third party providers when utilising cloud services. Addressing security is therefore a non-trivial task due to the importance of the enabling technology, such as virtualisation, coupled with social factors considering consumer needs and physical properties such as geo-location influencing jurisdiction on resources and processes. Consumers have security needs to ensure their data and processes are kept secure, where cloud service providers are responsible for identifying security requirements in cloud computing environments and providing assurance that security needs are satisfied through transparency. In order to provide transparency of security requirements in cloud computing environments, concepts unique to the cloud computing paradigm such as multi-tenancy, dissemination of resources and responsibility needs to be captured. By adapting a security requirements engineering approach, security issues are identified early in the software development life-cycle. This ensures the security needs within a system is identified and well-understood before committing to implementation and deployment, where it is exponentially more difficult to address underlying security issues.

In this thesis we present the Secure Cloud Environment Framework(SCEF), a decision support framework enabling developers to elicit cloud security requirements from cloud computing environments. The SCEF consists of a cloud modelling language, a secure cloud process and a set of cloud security analysis techniques. In the SCEF we make explicit, the exact definition and relationship of cloud security requirements within the scope of a cloud computing environment, encompassed and expressed at an organisational, application and infrastructure level of abstraction. The overall contribution to knowledge is the SCEF, which allows developers to model cloud computing environments in order to understand and address cloud security issues. The cloud modelling language is a contribution as it extends the Secure Tropos methodology, which enhances the existing security paradigm through domain-specific cloud computing concepts. The secure cloud process is a contribution as it provides developers with a methodological approach when applying the

SCEF to elicit cloud security requirements. The cloud security analysis techniques is a contribution as it facilitates semi-automated reasoning and enrichment of knowledge. Visual modelling and reasoning capabilities is supported by the SectroCloud module in the Apparatus Software Tool.

Contents

List of Figures

Acknowledgments

First and foremost I would like to thank my family for their continued support throughout my studies. My dad for his advice from an academics perspective and my mom and brother for their fairly unbiased opinions. I would like to give special thanks to the supervisory team for their mentoring and advice over the years. This work would not have been possible without their guidance. Professor Haralambos Mouratidis for the opportunity to work on this project, his expert knowledge and wealth of ideas. Dr Aidan Delaney for his prompt and applicable advice. Dr Stelios Kapetanakis who provided advice outside of the academic box. A big shout-out and thanks to all my friends and colleagues from CEM, it has been a pleasure to share this journey with all of you. I will miss the caffeine-fueled discussions we've had over these years. Finally last but not least, extra thanks to my friend David Ren for proof-reading this thesis and brainstorming ideas from the beginning.

Declaration

I declare that the research contained in this thesis, unless otherwise formally indicated within the text, is the original work of the author. The thesis has not been previously submitted to this or any other university for a degree, and does not incorporate any material already submitted for a degree.

Signed: _____

Dated: _____

Chapter 1

Introduction

The premise of the cloud computing paradigm is that computing resources are offered by third party providers as a form of commodity accessed through network connections (Armbrust et al., 2010; Mell, Grance et al., 2011). In comparison to traditional IT solutions, this lowers the capital needed and abstracts away implementation and infrastructure details by allowing cloud users to select from pre-configured computing services. Cloud computing systems are represented as a pool of resources, which consists of one or more physical servers. But because the configuration and delivery of resources in a cloud computing system is vendor specific, the cloud infrastructure can be distributed across multiple geographical locations with disparate jurisdictions. This poses several security challenges from a technical standpoint due to the outsourcing of data and processes to third party providers. Factors such as the co-tenancy of data and mutually distrusting users sharing the same physical servers also raises security issues with legal and jurisdictional implications. This is compounded by the complexity of determining the security requirements and properties of cloud computing systems involving multiple cloud services, service providers and consumers.

In this chapter we examine the security issues found in cloud computing systems, which combines the existing problems of traditional software systems with the new challenges introduced when outsourcing business operations and data to third party providers. We assess the challenges from an early requirements perspective, specifically focusing on goal-oriented security requirements engineering approaches

with a visual language component. The aim of the work presented is to provide semi-automated reasoning support for developers, throughout the decision making process, in order to understand and address cloud security issues. In this thesis the term developer refers to a practitioner with speciality in security requirements engineering or cloud computing, where their role is to analysis the security requirements of cloud computing systems. We discuss the limitations of existing security requirements engineering approaches in the literature when addressing the security challenges in cloud computing. In Section 1.1 we provide motivation for adopting cloud computing, describing the security challenges limiting the cloud paradigm. In Section 1.2 we describe the advantages of adapting a security requirements engineering approach when addressing cloud security challenges from the early requirements stage. Section 1.3 presents a motivating example demonstrating the benefits of cloud computing systems and the security issues discussed in this chapter. In Section 1.4 we derive three research questions in order to address the identified cloud security challenges. We propose a framework, which is further refined through research aims and we conclude with the anticipated contributions to knowledge. Section 1.5 outlines the structure of the thesis. Finally Section 1.6 provides a list of publications supporting the thesis.

1.1 Cloud Computing and Security Challenges

The National Institute of Standards and Technology (NIST) defines cloud computing as: "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.", where the cloud model is composed of "five essential characteristics, three service models, and four deployment models." (Mell, Grance et al., 2011). The NIST definition is used throughout this thesis to provide a consistent reference for cloud computing terminology.

Cloud computing resources are provisioned through an "utility model" (Buyya, Yeo, Venugopal, Broberg & Brandic, 2009) and offers numerous benefits to end users in comparison to traditional IT solutions; such as rapid deployment, self-servicing,

high availability and competitive costs. In a traditional IT approach, new infrastructure need to be purchased, delivered and installed. Software licences need to be secured and renewed. New personnel also need to be hired and trained to configure, use and maintain systems. Rapid deployment in cloud computing offers cloud users automated provisioning and de-provisioning of computing resources, where the technical implementations are managed by the cloud providers. This allows the cloud users to focus on business operations without worrying about how it is achieved. Self-servicing refers to the selection of ready-to-use computing and business solutions offered by cloud providers as cloud services. As an example a cloud user is able to select business intelligence and data backup services, in order to begin the process of integrating their business data into a cloud computing environment with minimal start-up time. High availability is achieved by distributing computational processes across the resource pool in a cloud computing infrastructure, thereby reducing risks incurred from a single point of failure where processes are hosted on individual terminals. The distribution of the workload across the resource pool is primary managed by the cloud service provider, where the computational processes and data may span across different geographical zones belonging to multiple third party cloud service providers. Therefore in comparison to traditional IT approaches, the major advantages for small to medium enterprises (SMEs) is the reduction of up-front capital costs, increased availability of business processes and a self-service approach when adapting cloud computing systems (Adam & Musah, 2014).

However one of the prerequisites for cloud computing, the outsourcing of data and processes to third parties, raises several security (Subashini & Kavitha, 2011; Sengupta, Kaulgud & Sharma, 2011) and legal questions (Almorsy, Grundy & Müller, 2016). To the cloud user, cloud computing is a black box where the user has little to no control over how or where their data is processed. Multi-tenancy in cloud computing refers to multiple cloud users running independent logical processes but sharing the same physical components, such as CPU, RAM and storage. Virtualisation is the enabling technology for virtual machines(VM), which emulates a physical server and is managed through software known as hypervisors. Therefore a single physical server can host a hypervisor, which manages one or more instances of virtual machines. Each instance of a virtual machine is allocated to a cloud user, which provides the cloud services required by a cloud user. However from a cloud comput-

ing context, the mutual distrust in multi-tenancy environments brings up questions about the security of user data when sharing physical infrastructure (Lombardi & Di Pietro, 2011; Li, Cuppens-Boulahia, Crom, Cuppens & Frey, 2016). Consider the scenario where two rival companies are using virtual machines hosted on the same physical server. A VM escape vulnerability (Luo, Lin, Chen, Yang & Chen, 2011) would enable one company to access the sensitive data of their rival company. This attack has been demonstrated by Ristenpart et al. (Ristenpart, Tromer, Shacham & Savage, 2009), using the Amazon EC2 service as a case study, where they explore the practicality of mounting a cross-VM attack in third-party compute clouds in order to extract information from a co-residing virtual machine.

In the case of self-servicing, the cloud user depends on the cloud service provider to manage service configurations, enforce security policies and ensure compliance to security standards. When computing resources are automatically provisioned and de-provisioned, the cloud user has no control over where their data and processes are stored (Kandukuri, Rakshit et al., 2009). How are they able to ensure that processes running in the cloud system meet their jurisdictional requirements? These are some of the security issues which need to be addressed before a cloud user can fully commit their data and processes to the cloud (Jamshidi, Ahmad & Pahl, 2013; Ahmad & Babar, 2014).

1.2 Security Requirements Engineering Approaches

Security is a factor that is most effective when integrated as early as possible in the software development life-cycle (Kissel et al., 2008). In order to understand how security issues affect user data and processes in cloud systems, an understanding of the users needs is required. More importantly an understanding of the system-to-be and the relationships between the concepts need to be captured. These needs are known as the user requirements (Pohl, 2010). Requirements engineering is a process towards precisely describing the problems a system should solve in a given environment (Cheng & Atlee, 2007). One method for tackling this area is by adopting an agent-oriented requirements engineering approach (Wooldridge, 1997), where the focus is placed on characterising active elements in the environment such as humans or machines to elicit requirements for the target system (E. Yu & Mylopoulos,

1998). Another approach is to adapt a goal-oriented requirements engineering approach to capture, at different levels of abstraction, the objectives the system should achieve (Van Lamsweerde, 2001). The key motivation for considering approaches at the requirements level is to support a developers understanding of system processes, objectives and relations. Stakeholder needs are described through textual and visual representations, where developers create models representing the system-under-design. An example of an agent-oriented approach is Tropos (Bresciani, Perini, Giorgini, Giunchiglia & Mylopoulos, 2004), which is based on i* (E. S. Yu, 1997), a goal-oriented requirements engineering approach.

By adapting a goal-oriented requirements engineering approach to cloud systems, we are able to capture high level concepts such as stakeholders, goals and resources. In model-driven approaches, a modelling language expresses stakeholder needs and system requirements through generalised models (Belaunde et al., 2003). Models can be visualised using graphical notation to help developers understand the concepts, processes and rationale behind the system design (Lapouchnian, 2005; Ellis-Braithwaite, Lock, Dawson & Haque, 2012; Faily, 2011). These models allow developers to examine and analyse stakeholder requirements by visualising components and interactions through an abstract birds-eye view of the cloud system. Thus model-driven approaches with a consistent visual language element can assist developers in visualising cloud-specific characteristics in systems, such as the security needs and responsibilities of third parties. Goal-oriented Requirements Language (GRL) (Amyot et al., 2010), i* (Eric, 2009) and KAOS (Darimont, Delor, Massonet & van Lamsweerde, 1997) are all examples of goal-oriented requirements engineering approaches with a visual language.

The cloud computing paradigm builds upon existing technologies and paradigms, such as distributed systems, virtualisation and Service-Oriented architecture (SOA). This creates a complex scenario where we need to consider security from multiple perspectives. Thus an understanding of existing security issues in these areas is essential for understanding security in cloud computing (Dillon, Wu & Chang, 2010; Jadeja & Modi, 2012). Many of the security challenges found in cloud computing are acute examples of existing issues, which arise as a result of adapting the cloud computing paradigm in traditional IT environments. As an example there are various existing security challenges concerning virtualisation and web service technology

in the literature (Barry, 2003; Chow et al., 2009; Luo et al., 2011; Lombardi & Di Pietro, 2011). However at the cloud computing level, many solutions to existing security issues are no longer applicable due to numerous factors such as involvement of multiple third party providers, multi-tenancy processes, geographically distributed user assets and data centres (Behl, 2011; Fernandes, Soares, Gomes, Freire & Inácio, 2014). While there are established research efforts in integrating security considerations into the software development life-cycle, the recent trend in migrating towards cloud computing solutions has revealed the need for techniques and methodologies to ensure the security and transparency of systems and services (Kanday, 2012; Ramgovind, Eloff & Smith, 2010; Sengupta et al., 2011). This is possible through security requirements engineering approaches such as Secure i* (Elahi & Yu, 2007), Secure Tropos (Mouratidis et al.) (Mouratidis & Giorgini, 2007), Secure Tropos (Massacci et al.) (Massacci, Mylopoulos & Zannone, 2010), SecureUML (Lodderstedt, Basin & Doser, 2002) and UMLSec (Jürjens, 2002). However these approaches lack the specific language requirements to describe cloud computing characteristics, in order to model and perform security reasoning on cloud systems.

Another obstacle limiting the uptake of cloud computing in the industry and public sector is the perceived lack of security when migrating towards cloud environments. Negative publicity include data-breaches (Depot, 2014; Alpeyev, Galante & Yasu, 2011), security leaks (Alliance, 2011), interoperability and compatibility issues (Bergmayr et al., 2013; Ferry, Rossini, Chauvel, Morin & Solberg, 2013; Frey & Hasselbring, 2011).

The Dropbox breach in 2012[1] is an example of a high-profile attack on a public cloud service provider, where leaked email addresses were targeted with spam emails. In this case the credentials of an employee at Dropbox was compromised on a third party website where attackers subsequent reused the same credentials to gain access to the employees' data on Dropbox; including a plain-text document containing email addresses registered with Dropbox accounts[2]. By identifying the security requirements of stakeholders in cloud-based systems, it is possible to model vulnerabilities exploited via threats such as social engineering. Subsequently developers are able to understand the impact of an attack in cloud systems through

[1]http://www.theregister.co.uk/2012/08/01/dropbox_breach/

[2]https://blogs.dropbox.com/dropbox/2012/07/security-update-new-features/

cloud-specific characteristics such multi-tenancy and service models, where multiple users sharing the same physical infrastructure are affected through a single attack. Developers are then able to devise measures taken to enforce security policies on confidential assets; thus understanding the security needs and implication of users in a cloud system. By adapting a security requirements engineering approach, developers can abstractly describe cloud system components and stakeholder needs from a goal-oriented perspective. Developers are then able to integrate high level security features without worrying about implementation-level mechanisms during the early requirement stage.

1.3 Motivating Cloud Adoption Scenario

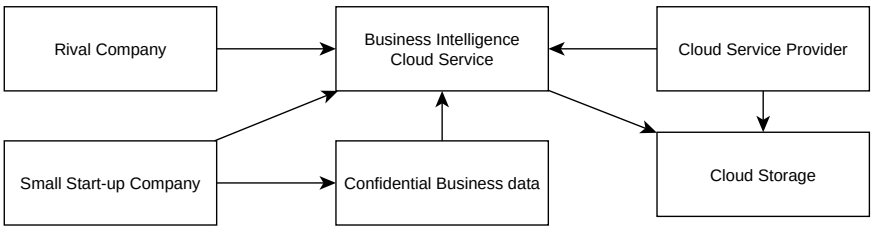

Figure 1.1: Example of a company outsourcing business intelligence processes to a cloud provider.

Figure 1.1 illustrates a scenario where a small start-up company deploys a business intelligence cloud service provided by a cloud service provider. The small start-up company offloads confidential business data to the cloud service, which is stored in a cloud storage owned by the cloud service provider. A rival company is also a customer of the cloud service provider and makes use of the same business intelligence cloud service. This scenario provides a highly abstract view of the stakeholders in the cloud system, the cloud service consumed by the stakeholders and the cloud concepts involved when consuming the cloud service.

Given this scenario, it is possible that the cloud storage is affected by a VM escape vulnerability as they share the same physical server. In this case the rival company is able to exploit the vulnerability and access data found in the cloud storage, including confidential business data owned by the small start-up company.

However we lack the means to fully capture the required information in order to model this scenario. In this scenario the stakeholders include the rival company, the small start-up company and the cloud service provider. However this only provides a high level view of the actors involved in a system. We need to define actors with malicious intent, in order to trace the potential attacks that are carried out. In this case the rival company is an example of an actor with malicious intent, as they intend to access the business data of the small start-up company. Thus a security requirements engineering approach would allow us to capture concepts such as malicious actors, vulnerabilities such as VM escape and security needs of stakeholders such as keeping the business data confidential.

We also need to support different *categories* or *types* of actors from a cloud computing perspective. For example a cloud service provider has different goals and relationships in comparison to a cloud consumer. A cloud service provider will manage one or more cloud services and possess ownership over some physical infrastructure required to deliver cloud services. A cloud consumer however possess ownership data, which is required by cloud services when a cloud consumer uses a cloud service. Therefore one constraint between the cloud actors is that a cloud consumer does not own physical infrastructure. The security requirements engineering approaches discussed so far do not provide support for cloud-specific concepts, such as multi-tenancy, service models or cloud services. To address this, we now define areas of investigation:

- **I1 Cloud Computing Properties**: The first area of investigation is to determine what cloud computing concepts are required to model a cloud computing system. More specifically we are interested in cloud computing concepts with the level of abstraction allowing developers to model cloud computing systems based on stakeholder requirements.

- **I2 Modelling Secure Cloud Systems**: The second area of investigation is to determine what activities are required to support developers throughout the course of modelling and designing secure cloud computing systems. In particular we want to ensure that developers are able to consistently apply our concepts.

- **I3 Cloud Security Analysis**: The last area of investigation is to determine what can be done to enhance the security of the system-under-design. More specifically, what sort of analysis can the developer perform to identify additional information on the security issues and solutions from models of the cloud system?

1.4 Research Questions and Outcomes

In this section we formulate research questions in order to address the identified gaps between cloud computing and security requirements engineering. We then present the anticipated contributions of the research after addressing the research questions.

1.4.1 Research Questions

- RQ1: How do we describe cloud computing concepts to capture cloud systems from a security requirements engineering perspective? This research question addresses the first area of investigation (I1).

- RQ2: How do we define a systematic process to guide security requirements engineers in modelling cloud computing systems? This research question addresses the second area of investigation (I2).

- RQ3: What types of analysis are useful in order to identify cloud security requirements in cloud systems? This research question addresses the third area of investigation (I3).

1.4.2 Research Aims and Objectives

In order to answer the research questions, the research aim is to provide a modelling language to describe cloud computing concepts, a process supporting developers through modelling and designing secure cloud systems and a set of analysis rules to reason about security needs. Thus we propose a framework consisting of a modelling language, process and supports analysis techniques. The purpose of the framework is to provide an integrated set of domain-specific concepts and functionality, which

enables the systematic application of a rigorously defined process. Thus the framework guides developers of cloud computing systems through the process of modelling cloud systems and performing analysis to obtain cloud security requirements. We now define a set of research objectives refining the research aim and describe deliverables of the framework.

- RO1: We will extend existing security requirements approaches in a way which captures the security and cloud needs of stakeholders from the early requirements stage. This contributes towards addressing RQ1.
- RO2: We will define a modelling language capable of modelling cloud computing concepts and relationships from a security requirements engineering perspective. This contributes towards addressing RQ1.
- RO3: We will systematically guide the developer through the process of mapping our modelling language to models of secure cloud systems. This contributes towards addressing RQ2.
- RO4: We will provide guidelines for modelling cloud systems in order to perform semi-automated analysis. This contributes towards addressing RQ3.
- RO4.1: The analysis will support threat and vulnerability identification to help security requirements engineers understand security issues in cloud systems. This contributes towards addressing RQ3.
- RO4.2: The analysis will propose mitigation techniques to help security requirements engineers understand how to address security issues in cloud systems. This contributes towards addressing RQ3.

1.4.3 Research Contributions

The core contribution of this work is **a framework providing decision support for developers to elicit cloud security requirements from a cloud computing environment**. The developer creates a model of the cloud computing system in order to identify cloud services and assets in the system-under-design. Decision support is provided through the identification of security issues in the model, and

the proposed mitigation strategy. The cloud security requirements elicited include the concepts required in order to address identified security issues. This consists of the security mechanisms, security objectives and security constraints identified to mitigate the cloud services and assets at risk in the system-under-design. This work contributes to addressing the lack of support in the Secure Tropos methodology for expressing security needs in the cloud computing domain.

The framework consists of; a modelling language, a process, and a set of analysis techniques. **The first contribution of the work is the extension of the Secure Tropos modelling language for cloud security needs**. This is a contribution to the body of knowledge in the field of security requirements engineering, specifically by extending the Secure Tropos modelling language in order to describe cloud security needs across both hardware and software levels from an organisational perspective. Our extensions to the Secure Tropos modelling language captures the domain-specific cloud computing characteristics, properties and relationships. Specifically we define the concepts which enables the modelling language to describe cloud-specific needs, including managing the security needs of multiple tenants in a virtualised environment, enumerating the configuration of cloud services, physical and virtual components and creating links between the cloud computing domain and security requirements engineering domain.

Cloud Security Example 1 - Compromising cloud user data through exploitation of virtual machine weaknesses:

In Figure 1.2 we illustrate an example using our graphical notation and cloud concepts, where the discrete data of unassociated cloud users sharing a public cloud service is compromised as a result of weaknesses in a virtual machine image.

From this model, the developer is able to identify a cloud threat impacting the virtual machine image required by *CS1*, and take measures to ensure the impact of the threat does not propagate beyond *CS1* to *CS2*. Specifically the *User backup data* in *CS2* is compromised as a result of a virtual machine weakness from *CS1*.

Thus from the perspective of a developer under the employment of the *cloud service provider* cloud actor, they are able to identify their responsibility in ensuring their cloud service *CS1* has the appropriate security measures in place to address and mitigate the threat of virtual machine image weakness.

Thus our work supports the developers understanding of their cloud system,

where our approach of modelling the physical, virtual and organisational components of cloud computing systems highlights the relationships and security needs.

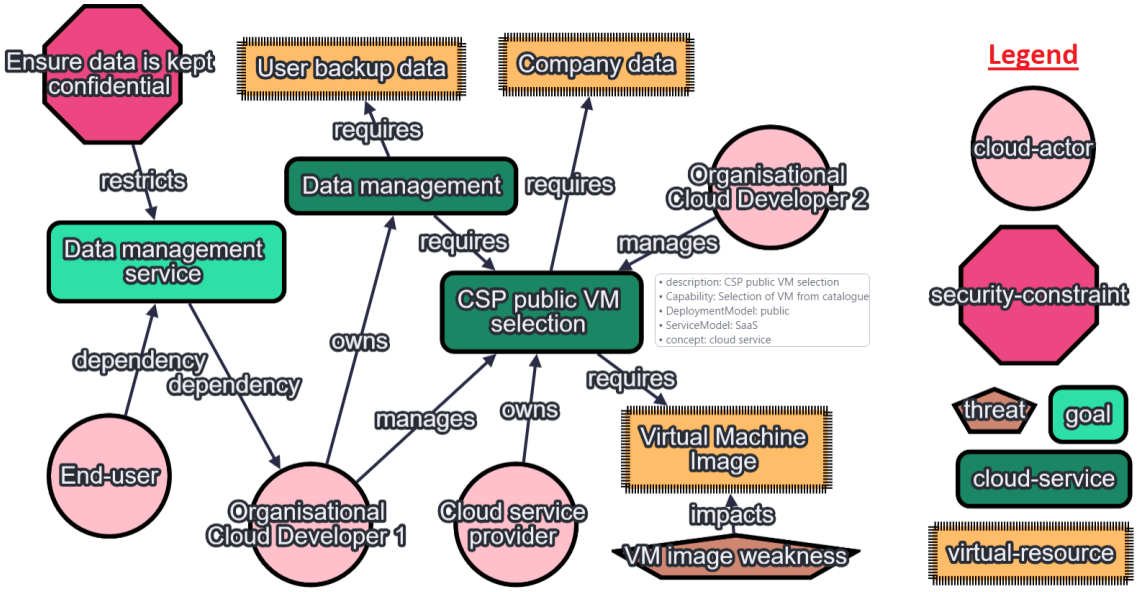

Figure 1.2: Example of a virtual machine image weakness resulting in compromised data on dependent cloud services.

The second contribution is our description of a process guiding developers in modelling the security needs of cloud computing systems. The process describes the procedure for modelling cloud computing systems, applying analysis techniques and enumerating cloud security needs. Users of our framework, envisioned to be organisational developers, follow a well-defined approach to model cloud and systems components at the organisational, application and infrastructure layers. This iterative approach refines a cloud environment model, capturing how concepts and relationships between different conceptual layers within a cloud system are created and associated, to facilitate the developers understanding of cloud security needs. This addresses the need for an approach which provides guidance, specifically to organisational cloud system developers, in addressing cloud security needs from a security requirements engineering perspective. In particular throughout this process, the security needs of cloud stakeholders in a system-to-be and the cloud characteristics of components are examined at multiple levels of abstraction.

Cloud Security Example 2 - Gaining access to cloud data through ac-

count hijacking: In Figure 1.3 we illustrate an example whereby a cloud users credentials can be exploited in order to gain access to assets located in discrete physical locations. Specifically this example is amplified if the cloud user credentials, *"User data"*, is shared with infrastructure managed or owned by 3rd party cloud service providers, namely the *"Datacentre 2"* with *EU* jurisdiction owned by the cloud service provider *"3rd party service user"* where *"Datacentre 2"* composes of *"Storage 2"* and *"User data"* also permeates to *"Storage 2"*. In which case a malicious actor is able to gain access to assets residing on 3rd party infrastructure using the same set of credentials, namely gaining access to *"Company data"*. Thus the scope of the exploit in a cloud environment expands beyond the traditional boundaries of a single organisational system.

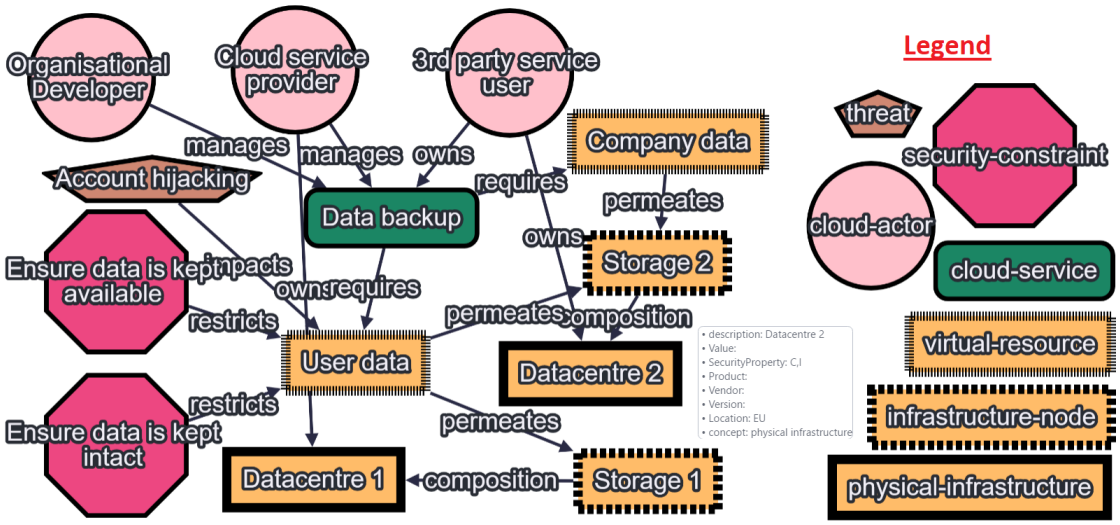

Figure 1.3: Example showing how compromised user credential can be used to gain access to cloud data located in different physical locations.

The third contribution of the work is a set of analysis techniques to perform security analysis, develop mitigation strategies and validate cloud relationships to generate a model with cloud security requirements. This allows the developer to perform semi-automated security and cloud analysis in order to validate cloud security needs in a cloud model. The **cloud security analysis** identifies cloud threats and vulnerabilities given a cloud model. The **mitigation strategy analysis** enhances existing cloud models, by integrating security

knowledge from both traditional and cloud-specific domains describing mitigation objectives and mechanisms. The **transparency analysis** resolves the cloud security needs, describing parties responsible for the mitigation strategy and the cloud security issues addressed. This is a contribution because the analysis provides semi-automated guidance for the user to identify, determine impact of and realise cloud security needs. Thus the analysis techniques enriches the Secure Tropos methodology and provides semi-automated decision support for developers designing secure cloud computing systems.

1.5 Thesis Structure

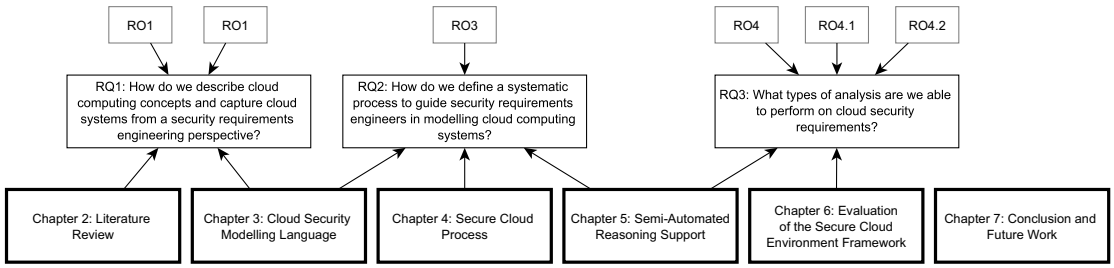

Figure 1.4: Model of the thesis structure showing the research questions, chapters and research objectives.

The thesis organised as follows:

- Chapter 2 examines the current literature in order to derive concepts for the cloud computing paradigm through a security requirements engineering perspective. We compare and contrast goal-oriented security requirements engineering approaches at the early requirements level. We focus on approaches representing concepts through a visual language.

- Chapter 3 describes the concepts, relationships, concrete syntax and instance syntax of the cloud security modelling language required to model secure cloud systems. We then describe models derived through instantiating subsets of the modelling language, focusing on conceptual layers within a cloud system to capture specific security requirements.

- Chapter 4 outlines the activities in order to systematically apply concepts from the secure cloud environment framework through a practitioners perspective.
- Chapter 5 presents the formalised concepts of the modelling language. We then describe three cloud analysis techniques supported through our formal work;(i) *cloud security analysis* to identify threats and security vulnerabilities, (ii) *security mitigation analysis*, to identify and validate security constraints, objectives and mechanisms and (iii) *transparency analysis* to identify security responsibilities of cloud actors and understand security needs.
- Chapter 6 demonstrates the validity of our approach through three use-cases and a case study based on the VisiOn European project.
- Chapter 7 summarises our contributions, discussing limitations to the work and proposes areas for future work.

1.6 Publications

Parts of the research presented in the thesis has been peer-reviewed and published in journals, conferences and workshops.

1. Shei, S., Mouratidis, H., & Delaney, A. (2017). A Security Requirements Modelling Language to Secure Cloud Computing Environments. In Enterprise, Business–Process and Information Systems Modeling (pp. 337–345). Springer, Cham. (Shei, Mouratidis & Delaney, 2017). This publication supports Chapter 3 and Chapter 5.
2. Argyropoulos, N., Shei, S., Kalloniatis, C., Mouratidis, H., Delaney, A., Fish, A., & Gritzalis, S. (2017, January). A Semi–Automatic Approach for Eliciting Cloud Security and Privacy Requirements. In: Proc. 50th Hawaii International Conference on System Sciences (Argyropoulos et al., 2017). This paper combines the research efforts of business process modelling with cloud security requirements. This approach addresses high level organisational needs and transforms them to operational cloud security requirements. This publication supports Chapter 3 and Chapter 5.

3. Mouratidis, H., Argyropoulos, N., Shei, S., Dimitris Karagiannis, Heinrich C. Mayr & John Mylopoulos (2016). Domain–Specific Conceptual Modeling. Springer International Publishing (Mouratidis, Argyropoulos & Shei, 2016). This book chapter presents the Secure Tropos methodology; the concepts, process and tool-support. We outline our extensions to the Secure Tropos methodology in order to model and reason about cloud security requirements. This publication supports Chapter 2 and Chapter 3.

4. Shei, S., Kalloniatis, C., Mouratidis, H., & Delaney, A. (2016, September). Modelling Secure Cloud Computing Systems from a Security Requirements Perspective. In International Conference on Trust and Privacy in Digital Business (pp. 48–62). Springer International Publishing (Shei, Kalloniatis, Mouratidis & Delaney, 2016). In this publication we discuss our cloud modelling language, the abstract concepts and a holistic model. We present a meta-model to visualise the relationships between our cloud and security concepts. The Cloud Environment Model captures the organisation, application and infrastructure concepts of a cloud system. This is achieved by instantiating our concepts in the meta-model to represent corresponding components in the cloud system. This publication supports Chapter 3.

5. Shei, S., Márquez Alcañiz, L., Mouratidis, H., Delaney, A., Rosado, D.G., Fernández-Medina, E. (2015, August). Modelling secure cloud systems based on system requirements. In Evolving Security and Privacy Requirements Engineering (ESPRE), 2015 IEEE 2nd Workshop on (pp. 19–24). IEEE. (Shei, Alcaniz et al., 2015). This publication discusses the mapping between three conceptual layers in the cloud paradigm and how our proposed views capture this information. We demonstrate how our work complements the analysis stage in the cloud migration framework SMiLe2Cloud (Márquez, Rosado, Mouratidis, Mellado & Fernández-Medina, 2015) to model cloud security requirements. This publication supports Chapter 3 and Chapter 4.

6. Shei, S., Delaney, A., Kapetanakis, S., & Mouratidis, H. (2015). Visually Mapping Requirements Models to Cloud Services. In DMS (pp. 108-114) (Shei, Delaney, Kapetanakis & Mouratidis, 2015). This publication presents the con-

cepts of the Secure Tropos modelling language and our extensions to support cloud requirements. The paper proposes the idea of the goal-plan-resource pattern which identifies requirements in order to model secure cloud systems. This publication supports Chapter 3.

Chapter 2

Literature Review

In Chapter 1 we have discussed several security challenges specifically impacting the cloud computing paradigm and motivated a security requirements engineering approach to target these challenges. Section 2.1 investigates the unique characteristics of the cloud computing paradigm to further our understanding of security in the cloud computing domain. In order to capture cloud computing properties, we search the literature for cloud computing definitions, taxonomies and modelling approaches in Section 2.1.1. In Section 2.1.2 we review literature describing the specific security issues and challenges faced in cloud computing.

However there is a broad range of research work in the field of security requirements engineering. Thus in Section 2.2 we narrow down the scope of our investigations by focusing on security requirements engineering approaches from the early requirements stage. In Section 2.2.1 we examine goal-oriented requirements engineering approaches as a precursor to recent security requirements engineering approaches. Additionally in Section 2.2.2 we examine model-driven security requirements engineering approaches which include a visual modelling element to represent the system-under-design.

2.1 Cloud Computing Properties

2.1.1 Definitions, Taxonomies and Modelling the Cloud

In order to perform security reasoning in cloud computing systems from a security requirements engineering perspective, we first need to describe the concepts and relationships within a cloud computing system. Therefore we need an understanding of concepts and the level of detail required to create a model representing a cloud computing system. We need to understand and model the stakeholders security needs and objectives, from a developers point of view. Thus we approach from an early requirements perspective, narrowing the scope to concepts at higher levels of abstraction. A high level approach allows the developer to describe cloud computing components and their relation to stakeholder needs, without the limitations of considering specific technical implementations. Thus in this subsection we review works focusing on dissecting the cloud computing paradigm through common definitions, taxonomic and modelling approaches.

There are many works in the state of the art which surveys current work to determine a standard definition for cloud computing (Vaquero, Rodero-Merino, Caceres & Lindner, 2008; Foster, Zhao, Raicu & Lu, 2008; Qian, Luo, Du & Guo, 2009; Armbrust et al., 2010). Vaquero et al. studies over twenty definitions in order to extract a consensus and the minimum set of essential characteristics (Vaquero et al., 2008). Foster et al. discusses the overlap between the cloud paradigm and existing technologies such as grid computing and distributed systems in (Foster et al., 2008). Both (Vaquero et al., 2008) and (Foster et al., 2008) make comparisons between the grid paradigm and cloud computing, noting the similarities and highlighting the differences. For example virtualisation existed before cloud computing, however it enables key cloud computing features such as on-demand resource pooling and security by isolation. Qian et al. provides a reference architecture of the cloud through a core stack and the management, where the core stack consists of three layers; resource, platform and application (Qian et al., 2009). Qian et al. also provide a classification of cloud computing using two categories; service boundary and service type. They relate the context of each category to the target user group, such as enterprises, developers and external parties. Moreno-Vozmediano

et al. presents the concept of services available on the Internet in the Internet of Services (IoS) model (Moreno-Vozmediano, Montero & Llorente, 2013). In their view some of the challenges encountered in cloud computing are based on areas that are already defined in the existing literature, such as virtualisation, grid computing and automated computing. CloudML-UFPE is a XML-based approach describing services offered by cloud providers (Bergmayr, Wimmer, Kappel & Grossniklaus, 2014). The modelling language focuses on the infrastructure level in cloud offerings, describing concepts such as cloud services and resources as nodes with links between them. Bergmayr et al. provide detailed description of nodes at the hardware level using properties such as CPU, storage and memory. Zhang et al. divides the cloud computing environment into four layers; the hardware/datacenter layer, the infrastructure layer, the platform layer and the application layer (Zhang, Cheng & Boutaba, 2010). Zhang et al. examine the challenges in cloud data security, outlining the responsibility of infrastructure providers to maintain confidentiality when accessing and transferring data and ensuring the auditability of application security settings and policies. Due to the virtualised nature of the cloud environment, Zhang et al. highlight the need to consider security on all four layers of the cloud computing environment.

In addition to works in academia, we also examine the standards and specifications ensuring best practises in cloud computing. The standards provided by standards bodies and organisations aim to guide the development and adaption towards cloud computing systems for enterprises. It also facilitates the trustworthiness, interoperability and auditability of cloud computing systems through recommended guidelines. The Organization for the Advancement of Structured Information Standards(OASIS) defines the Topology and Orchestration Specification for Cloud Applications(TOSCA) in order to formally describe a "service template" which specifies the "topology" and "orchestration" of IT services (OASIS, n.d.). The formal description of IT services include their structure, properties, and behaviour, in accordance to constraints and policies when achieving service level objectives. Thus TOSCA provides a fine-grained formal description of services in the context of cloud applications, including the operations, deployment and implementation artifacts. These specifications allows a developer to describe cloud-specific technical details, though this is more applicable towards the design and implementation stages of the

development process. Similarly the Open Commons Consortium(OCC) (formerly the Open Cloud Consortium) supports the development of standards, benchmarks and open source implementations for cloud computing and frameworks for interoperability between clouds (Center for Computational Science Research, n.d.). However the OCC focuses on large-scale data compute clouds, thus falling outside the scope of our standardised approach to cloud computing. The Open Cloud Computing Interface(OCCI) is a set of specifications described by the Open Grid Forum (OGF), applicable to cloud resource management (Metsch, Edmonds et al., 2010). As an example, one of the area of focus is interoperability in the IaaS cloud API. Again this falls outside the scope of identifying cloud computing concepts during the early requirements stage, in order to produce an abstract model of the cloud system. The Object Management Group (OMG) focuses on modelling and model-based standards. OMG formed the Cloud Standards Customer Council in 2011 (Group, n.d.), which complements existing cloud standards and supports the transition from traditional IT environments by focusing on client-driven requirements. However their modelling approach is from a business-oriented perspective with the Business Process Model and Notation (BPMN) (Jordan & Evdemon, 2011). The Security, Trust & Assurance Registry(STAR) is a security assurance program introduced by the Cloud Security Alliance(CSA). This program covers three incremental levels of assurance for providers and customers through the alignment of the Consensus Assessments Initiative Questionnaire(CAIQ) with the Cloud Control Matrix(CCM) and "Security Guidance for Critical Areas of Focus in Cloud Computing" (Alliance, 2011). The CCM provides security principles cross–referencing cloud computing properties with industry-accepted security standards, regulations, and controls. The CAIQ is a questionnaire for measuring security properties and eliciting conformance of cloud providers through a series of binary questions, which corresponds to the security principles covering 16 domains and 133 controls in the CCM framework. Thus the CCM provides a pattern library for developers to reference specific controls and standards satisfied in a cloud computing system.

One common theme throughout these works is the classification of cloud computing using two categories; deployment and service models. The deployment models describe the boundary of the cloud, while the service model describes the type of service offered to users. This is further defined in the NIST definition (Mell, Grance

et al., 2011).

In the papers reviewed so far, the NIST definition is frequently referenced in the state of the art and commonly accepted by academics and practitioners (Yang & Tate, 2012; Rimal & Choi, 2012; Modi, Patel, Borisaniya, Patel & Rajarajan, 2013; Hoberg, Wollersheim & Krcmar, 2012). Expanding on the NIST definition of cloud computing in 1.1, NIST (Mell, Grance et al., 2011) defines five characteristics, three service models and four deployment models. The essential cloud computing characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. The cloud service models are Software-as-a-Service(SaaS), Platform-as-a-Service(PaaS) and Infrastructure-as-a-Service(IaaS). These three cloud service models form the basis for all cloud services provided by cloud service providers in cloud computing. The cloud deployment models are public, private, community and hybrid. The public and private cloud deployment models are more commonly seen in comparison to community or hybrid models. However a private or hybrid model doesn't imply a stronger sense of security, though the difference between the exposed attack surface should be taken into consideration (Subashini & Kavitha, 2011). These characteristics represent the standard terminology used when examining cloud computing security issues and systems (Kumar & Vajpayee, 2016; Fernandes et al., 2014; Rong, Nguyen & Jaatun, 2013; Los, Shackleford & Sullivan, 2013). For example the top ten cloud threats in (Los et al., 2013) are mapped to cloud service models in terms of impact on assets, while in (Kumar & Vajpayee, 2016) cloud threats are mapped to attack types and layers. Chou compares the cloud security risks and threats from three various perspectives based on the nature of the cloud service models, using real-world examples of cloud exploits to demonstrate the techniques which can be used against cloud computing systems (Chou, 2013). Therefore the properties in the NIST definition are able to, from the early requirements level, describe cloud computing models at an unified level of detail. This balances generalisation of the fundamental cloud characteristics with the possibility of providing formal and fine-grained technical details through cloud standards and specifications.

2.1.2 Security Issues and Challenges in Cloud Computing

In order to understand how cloud security requirements can be satisfied, we need to first define and capture the security issues and challenges in the cloud computing paradigm. We will determine precisely, what assets we are protecting in the system-under-design, how these assets can be compromised and what measures should be taken to prevent this. More specifically in our approach, we will examine the literature to determine the correlation between cloud computing characteristics and what constitutes a cloud-specific threat and vulnerability.

Grobauer et al. leverages the NIST definition of cloud computing characteristics to reason about cloud computing issues, where the authors define four indicators of cloud-specific vulnerabilities (Grobauer, Walloschek & Stocker, 2011). In no particular order, a vulnerability is cloud specific if; it is intrinsic in a core cloud computing technology, it is prevalent in established cloud offerings, the root cause is one of the cloud computing characteristics defined by NIST, when a cloud-specific feature makes it difficult or impossible to enforce established security controls. In (Armbrust et al., 2010) Armbrust et al. focuses on examining cloud computing environments from a SaaS users perspective. The authors present the top ten categories of obstacles and opportunities influencing the growth of cloud computing. For example, in order to address availability and business continuity, the authors propose the use of multiple cloud providers to distribute workload of processes and data. The authors provide suggestions such as focusing on horizontal scalability of virtual machines, rapid upward and downward scaling of software applications and the optimisation of infrastructure software to increase performance of virtual machines. From a security perspective, the proposed suggestions introduces new issues at different levels of abstract. As an example, the use of multiple cloud providers indicates that the cloud user has place trust in that the providers are able to satisfy their security needs and that security policies are actually implemented and enforced.

Pearson et al. expand on the issues of privacy, security and trust in cloud computing and provide the initial steps towards addressing these issues (Pearson & Benameur, 2010). They highlight issues such as disparate distribution of physical cloud computing resources in terms of geographical location, the impact of compli-

ance in the context of data traceability and the issues with legislation. They also raise the question of the re-usability of existing security mechanisms in the cloud, due to the shared issues with related paradigms such as web-based services and SoA.

The findings from these work infer that in order to address the current security challenges in cloud computing, it is crucial to have an understanding of the existing security issues in areas and software systems that are considered traditional or legacy. Specifically existing security issues are exacerbated in a cloud computing environment (Takabi, Joshi & Ahn, 2010), primary due to the multidomain environment where security, privacy and trust requirements need to be considered.

Rong et al. highlights three areas of interest in current cloud computing challenges; trusted data sharing, extending or modifying the SLA to cover issues such as confidentiality and integrity through security mechanisms aimed at cloud SLA and holistic mechanisms to address accountability in public clouds (Rong et al., 2013). The authors categorise cloud security issues into traditional and cloud-specific challenges. where the authors focus on concerns with the privacy and confidentiality of user data. Standards and interoperability is another area that bears investigation, as the authors point out that while there exists many standard bodies with a variety of offerings and agendas, there are no currently no dedicated cloud standards. Our approach is to identify the most comprehensive standard body, in this case the CSA and apply their proposed framework towards a cloud standard with the CCM.

The concept of virtualization on data security is examined by Lou et al.(Luo et al., 2011), where they present a Virtualization Security Framework towards improving the security, reliability and availability of virtual computing environments. The framework consists of two parts; virtual system security and virtualization security management which targets the risks cause by the current virtualization security and management.

The Cloud Security Alliance (CSA) present the top issues during 2011 in cloud computing with focus on the industrial sector in "Security Guidance For Critical Areas of Focus in Cloud Computing V3.0" (Alliance, 2011), categorising the issues and challenges into 14 domains. While all domains are relevant to our research, we are most interested in the compliance, information management and data security, interoperability, application security and virtualization. A more recent white paper published by CSA titled "The Notorious Nine: Cloud Computing Top Threats in

2013" lists 9 critical challenges in cloud security in industry, with comments and recommendations from experts (Los et al., 2013). These threats are decomposed into impact, implications, links to security controls and the relevant domains described by CSA in (Alliance, 2011).

We now discuss work which proposes security governance frameworks to align cloud security standards with cloud service provider offerings or user requirements, in order to ensure compliance and transparency are guaranteed. Ensuring that security standards and guidelines are followed and certified is crucial for building and maintaining a healthy relationship based on trust, assurance and transparency between cloud service providers and customers (Hashizume, Rosado, Fernández-Medina & Fernandez, 2013).

Rebollo et al. perform a systematic review of information security governance frameworks in the cloud computing environment, where they define a comparative framework to identify criteria and focus on the properties found in cloud models (Rebollo, Mellado & Fernández-Medina, 2012). They also present their own information security governance framework *ISGCloud* in a real-life case study with empirical evaluation validating the results (Rebollo, Mellado, Fernández-Medina & Mouratidis, 2015).

The majority of related work in the literature attempts to align cloud security standards with cloud service providers to assess compliance and quantitatively measure providers offerings (Ullah, Ahmed & Ylitalo, 2013; Pumvarapruek & Senivongse, 2014; Bhensook & Senivongse, 2012; Hale & Gamble, 2013; Thaweejinda & Senivongse, 2014; Ruo-xin, Cui, Gong, Ren & Chen, 2014). Very few authors focus on eliciting the security requirements and constraints from existing systems or consider an approach from the users perspective in this process.

Bhensook et al. identifies the compliance of cloud service providers to security requirements using the Goal Question Metric (GQM), which brings three levels of measurements; conceptual, operational and quantitative (Bhensook & Senivongse, 2012). They define a weighted scoring model for assessment. The security goals and the questions that address the goals were based on the CCM and CAIQ from the STAR programme by CSA, from which more refined questions and defined metrics are created to help provide quantitative answers and placing focus on evidence of security compliance from the cloud service providers. The architecture of their scoring

system is based on CloudAudit, where they go on to assess the Amazon Web Service (AWS) as an example in this paper. While they talk about security requirements in this paper, the requirements are addressed to assess the compliance of the cloud service providers to the pre-defined list questions found in the CAIQ. They do not take the security requirements from the customers point of view into consideration in their assessment process. In this respect, while the work presented here provides a generic architecture for assessing the compliance of security requirements from the service providers perspective, it is not sufficiently refined to take the multiple layers of complexities introduced by real-life systems.

In a similar work, Thaweejinda et al. presents a semantic search framework based on the CCM security controls, from which they construct a security ontology and build provider profiles providing evidence of conformance to the security controls and ranking data based on customer queries (Thaweejinda & Senivongse, 2014). While their framework allows the customer to enter queries regarding compliance to specific standards, they do not take the specific security requirements of a potential system into account.

Pumvarapruek et al. adapts a text classification approach, where they extract and classify the security conformance and compliance of cloud service providers based on the published information provided by cloud service providers on their web pages (Pumvarapruek & Senivongse, 2014). They study the CCM and CAIQ to define a set of security concepts, from which they form the basis for comparison against the cloud service providers to determine their security conformance level. During the classification process, they compile a list of security principles from the CCM and CAIQ, from which they build a security ontology with control groups and domains. The customers will extract the HTML content consisting of security concepts from the websites of cloud service providers, which is classified using concept-based classification with the conformance to the CCM. The results produced are degrees of conformance from 0 to 1 for each cloud service provider to a control group or domain. In essence, the cloud computing characteristics Pumvarapruek et al. compile in their security ontology is based on the proposed cloud ontology presented by Youseff et al. (Youseff, Butrico & Da Silva, 2008). While the text classification approach proposed by Pumvarapruek et al. seeks to determine the conformance of cloud service providers to security concepts devised from the

CCM and CAIQ, they base their evaluation process on published information by cloud service providers. The primary limitation in this case is that the requirements of users are not taken into account, therefore the results of the classification are presented as high level guidelines. Our approach considers the requirements of the stakeholders during the early requirements stage, therefore the security needs of the system-under-design are clear from the beginning.

Kao et al. introduces a framework for self-governance within cloud security (Kao, Mao, Chang & Chang, 2012). Their approach is based on the secure system development life cycle (SSDLC), with the addition of risk assessment. They consider the process from the organizational perspective, where the system takes the security concerns into consideration during each activity. In this work, the security measures are based on the guidelines, standards and vulnerabilities in cloud computing identified by the CSA (Alliance, 2011; Los et al., 2013) and NIST (Kissel et al., 2008). While the main novelty of the proposed governance framework builds on the consideration of security concepts during each stage in the system development life cycle, the authors mention but did not provide sufficiently detailed analysis of the exact standards, security controls or mapping between cloud service and deployment models towards the envisioned system.

Ruo-xin et al. propose a model to quantitatively assess cloud service providers and their safety levels, through a security evaluation index system (Ruo-xin et al., 2014). The evaluation index system is based on security assessment criteria derived from two branches; technical and management requirements. This approach cannot be carried out during the early requirements stage, as there is an insufficient level of detail regarding technical or managerial requirements. However this would be more suitable towards the design and implementation levels.

The work reviewed in this section are primary proposed as part of governance frameworks, targeted at providing security compliance, legacy to cloud migration and security policy guidance for cloud users and cloud service providers. The majority of approaches taken propose low level, technical implementation of security mechanisms with a strong focus on addressing existing systems.

2.2 Security Requirements Engineering

2.2.1 Requirements Engineering Approaches

Modern organisations depend on complex information systems for business continuity and deliver services of value to stakeholders (Armbrust et al., 2010). However software projects are focused on rapid delivery of features and often go through the development process without considering security issues or concepts (Howard & Lipner, 2006). Security concepts are then tacked on as an afterthought in live systems, often after the event of a security breach. This results in costly solutions to address serious security shortcomings, which often requires a complete redesign of the existing system. While the there has been interest and research targeting this field, the concept of security requirements engineering is often not well understood. For example developers may attempt to define security mechanisms in terms of design solutions instead of describing the concept of the root cause and protection. Thus taking a security requirements engineering approach is critical in the cloud computing environment in order to support developers in understanding their stakeholder goals and security needs. An understanding of the complexities in cloud computing systems facilitates the identification and selection of design alternatives towards a secure system.

In the previous section, we have reviewed a subset of work which defines the scope of cloud computing and the security issues and properties in cloud computing systems. We now limit the research domain to approaches within requirements engineering and security requirements engineering. Specifically we look at approaches starting from the early requirements stage with a visual language and modelling component. We now expand on the goal-oriented requirements engineering approaches discussed in Chapter 1.

i* (Eric, 2009) is a highly influential agent-oriented framework in the field of requirements engineering. The framework is designed to capture the strategic interests of multiple agents in complex systems. We use the original i* framework defined by Eric Yu (E. Yu, 2011) in his thesis dissertation as the primary reference in our discussions. i* defines two models corresponding to different levels of abstraction involving actors with strategic intentions; the Strategic Dependency(SD) model rep-

resents intentional concepts while the Strategic Rationale (SR) represents rational concepts. The SD model describes actors, sets of dependencies and the dependum; which can be a resource, task, goal or softgoal concept. The SR model refines the intentional elements of an actor inside their boundary through the means-end and task-decomposition links. While the i* framework can be able to capture the strategic intent of actors in a cloud system, it does not support the language concepts or graphical notation for describing cloud specific components. In Section 2.2.2 we discuss several extensions of i* which support security concepts (Elahi & Yu, 2007; Liu, Yu & Mylopoulos, 2003).

Goal-oriented Requirements Language (GRL) is an internationally recognised standard for goal-oriented modelling (Amyot et al., 2010), which integrates the core concepts of i* (E. S. Yu, 1997) and the Non-Functional Requirements (NFR) framework (Chung, Nixon, Yu & Mylopoulos, 2012). GRL offers a visual goal-modelling language with a clear separation between model concepts and their graphical representations. The modelling language supports qualitative and quantitative attributes, through contribution links with icons, numbers and text. The GRL syntax is based on the i* language, sharing common concepts such as actor, goal, resource and task. A GRL diagram describes the high level organisational business goals and non-functional requirements of stakeholders with alternative ways to achieve them. GRL supports evaluations by analysing trade-offs between conflicting goals, through qualitative or quantitative satisfaction values. A strategy is the starting point of evaluations, given initial satisfaction values between intentional elements. Three directions of propagation is supported between linked intentional elements while taking contribution types into account, providing a global assessment of a system. The qualitative and quantitative attributes has the potential to support the refinement and selection of cloud services based on user needs. However the GRL language does not support the specialised concepts required to capture cloud-specific characteristics, nor the security concepts found in cloud systems.

Tropos is an agent-oriented software development methodology, focusing on the development life cycle from early requirements to implementation (Bresciani et al., 2004). The Tropos modelling language is based on the i* framework, which describes models in Tropos through instances from a metamodel (Susi, Perini, Mylopoulos & Gi, 2005). The metamodel defines the abstract syntax of the modelling language,

in this case Tropos incorporates many concepts and relationships from i*. However instead of defining types of models such as the SD and SR in i*, Tropos uses views to represent the different levels of abstraction between phases. The Tropos methodology has five development phases: early requirements , late requirements, architectural design, detailed design and implementation. Tropos focuses on the early and late requirements stages.

KAOS (Darimont et al., 1997) is a goal-oriented requirements engineering method which elaborating objectives to be achieved by the system-under-design into requirements and assumptions, where the responsibilities are assigned to agents. The method focuses on the feasibility, completeness and consistency of requirements through a semi-formal graphical notation or formal when needed. In (Van Lamsweerde et al., 2007) van Lamsweerde consolidates all his previous research on KAOS to include formalisation of requirements using linear time temporal logic in (Dardenne, Van Lamsweerde & Fickas, 1993), analysis for conflicting requirements in (Van Lamsweerde, Darimont & Letier, 1998), and the use of anti-models to elaborate security requirements in (Van Lamsweerde, 2004). The KAOS method considers multiple stakeholders in a system-under-design and defines multiple views corresponding to different models. For example in a goal model, stakeholder goals are refined through an AND/OR refinement tree. Thus requirements are represented through leafs assigned to agents. Again while the KAOS method allows refinement of goals to represent stakeholder needs, the language lacks the expressiveness to capture cloud specific concepts.

So far we have defined and motivated key requirements engineering approaches, capturing stakeholder and system requirements from an agent and goal-oriented perspective. We now describe work which extend these approaches to integrate security concepts through a security requirements engineering approach.

2.2.2 Security Requirements Engineering Approaches

Mellado et al. presents their systematic literature review of existing work in security requirements engineering, providing a state-of-the-art of approaches in the field (Mellado, Blanco, Sánchez & Fernández-Medina, 2010). Their process is based on identifying initiatives in work adapting a security requirements approach and fo-

cusing from the early stages of the software development life-cycle. Thus the authors work motivates our review of security requirements engineering approaches which support modelling and reasoning about cloud security requirements.

Benjamin et al. proposes a conceptual framework to consolidate central concepts used in security requirements engineering in (Fabian, Gürses, Heisel, Santen & Schmidt, 2010). They review a range of approaches including UML-based, goal-based, multilateral, problem frame-based, risk analysis-based and common criteria-based. Thus Benjamin et al. provide a mapping between the diverse terminology to their proposed framework, providing specific approaches according to the scope of the issue. They stress the importance of accounting for multiple stakeholder views, where only the Multilateral security requirements analysis (MSRA) (Gürses & Santen, 2006), Security quality requirements engineering methodology (SQUARE) (Mead & Stehney, 2005), Keep All Objectives Satisfied (KAOS) (Darimont et al., 1997), and the Secure Tropos variants (Massacci, Mylopoulos, Zannone et al., 2007; Mouratidis & Giorgini, 2007) address this issue.

STS-ml (Dalpiaz, Paja & Giorgini, 2011) is a security-oriented approach capturing the security requirements of multi-agent socio-technical systems. The approach focuses on describing the social interactions between social and technical actors in a system-to-be. Specifically they define commitments to denote an agents security needs, based on the satisfaction of security properties such as non-disclosure of confidential data or non-repudiation of a delegated goal. While their approach tackles socio-technical systems based on social interaction between agents, it does not support a specialised vocabulary for capturing cloud computing concepts.

Secure Tropos (Mouratidis & Giorgini, 2007; Mouratidis, 2009) is a goal-oriented software engineering methodology extending the Tropos methodology to model security concerns throughout the software system development process, based on the notion of agents and related notions such as actors and goals and focusing on the early analysis and design stages. Mouratidis et al. presents a framework to elicit the security and privacy requirements of software systems and select a cloud service provider based on satisfaction of the requirements. They define a modelling language as part of the process, the language extends from several concepts in the software engineering discipline, such as the i* framework, PriS (Kalloniatis, Kavakli & Gritzalis, 2008) and Secure Tropos which provides specialization in requirements

engineering, security engineering and privacy engineering respectively (Mouratidis, Islam, Kalloniatis & Gritzalis, 2013).

Bandara et al. carries out a comparative evaluation of model-based security patterns to examine the extent of support of constructs provided by security requirements engineering approaches. They cover three main categories of modelling approaches; design, goal-oriented requirements and problem-oriented. Their results suggest that "current approaches to security engineering are, to a large extent, capable of incorporating security analysis patterns" (Nhlabatsi et al., 2010).

We now review recent work which tackles security issues in cloud computing systems from a security requirements engineering approach. More specifically we discuss approaches which support the developer during the modelling and design of cloud computing systems, guiding them through the process of refining components and security needs.

Kalloniatis et al. presents a methodology towards eliciting and analysing security and privacy requirements of software systems and the selection of appropriate cloud deployment models (Kalloniatis, Mouratidis & Islam, 2013). Their framework provides a modelling language based on Secure Tropos (Mouratidis & Giorgini, 2007), the agent-oriented modelling language i* and the PRiS (Kalloniatis et al., 2008) language, which incorporates concepts from security requirements engineering and cloud engineering through a systematic process.

Beckers et al. proposes a pattern-based method to elicit cloud security requirements aimed at guiding cloud customers during the process of modelling cloud systems (Beckers, Côté, Faßbender, Heisel & Hofbauer, 2013). They define a metamodel for the Cloud System Analysis Pattern (CSAP) in order to specify validation conditions, serving as the basis for their eclipse-based tool support which enables searching and instantiating cloud system analysis patterns and validating the security requirements against elements of the cloud. Our approach allows both cloud users and cloud service providers to model cloud computing systems, where we obtain a comprehensive and detailed view of the cloud components using a multi-layer approach.

Zardari et al. argues that the Goal Oriented Requirements Engineering (GORE) approach provides a paradigm which addresses the lack of requirements engineering methodologies applicable in cloud adoption (Zardari & Bahsoon, 2011). To capture

user requirements at various levels of detail, they decompose goals down to business, core and operational goals. Their proposed lifecycle facilitates the negotiation and alignment of user requirements with cloud provider provisioning, taking into account mismatches, trade-offs and risk management. Their outcome is the selection of the optimal cloud service provider given a set of user requirements. Our approach focuses on security requirements of cloud systems from a developers perspective, while taking into account the security needs of cloud users through constraints. We also consider the impact of cloud-specific characteristics such as virtualisation and multi-tenancy, implicitly modelling these concepts in the system-under-design. This allows developers to assess the system from a cloud perspective, identifying threats unique to cloud systems.

Iankoulova et al. carries out a systematic review of work in the literature which addresses security requirements in cloud computing. Their research goal was to provide a comprehensive view of areas that are under-researched and most investigated. They identified nine sub-areas; "Access Control, Attack/Harm Detection, Non-repudiation, Integrity, Security Auditing, Physical Protection, Privacy, Recovery, and Prosecution" (Iankoulova & Daneva, 2012) where non-repudiation, physical protection, recovery and prosecution were the most under-researched from a sample of 55 selected papers. This work helped clarify the research gaps in the cloud computing security domain, though a limitation is that the results of the study is dated.

Menzel et al. fosters a model-driven approach that allows the user to define security requirements at the modelling layer and facilitate a transformation based on security configuration patterns towards enforceable security policies (Menzel, Warschofsky, Thomas, Willems & Meinel, 2010). The security requirements are assumed to reside in security policies in service-based systems, in order to facilitate negotiation between users and service providers. The authors state the challenges in understanding and coding such policies, thus their approach integrates security intentions and allows modellers to state basic requirements. Their cloud-based Service Security Lab provides a virtualised testing environment for users to monitor and analyse the enforcement of their security requirements and policies.

Paja presents a security engineering methodology for social-technical systems in her doctoral thesis (Paja, 2014). One of her key contributions is the STS-ml modelling language, which specifies security requirements as social contracts constrain-

ing the social interactions and responsibilities of actors in her STS methodology. STS-ml also defines a set of formal semantics which allows automated reasoning to support developers in modelling and reasoning around conflicts between security requirements and actors' business policies. Paja also proposes automated reasoning techniques to calculate the impact of social threats on actors' information and their objectives. While Paja's STS methodology focuses on security requirements in social technical systems, there's no support for cloud computing concepts.

Aljawarneh et al. proposes a generic framework to deal with security vulnerabilities early in the Cloud Software Development Life Cycle (CSDLC) (Aljawarneh, Alawneh & Jaradat, 2017), where they add an extra level for security concepts and demonstrate the applicability through a case study. While their approach considers security at an early level, specifically security vulnerabilities in the cloud SaaS model, we argue that it is not sufficient to only focus on one stage. Our approach embeds security throughout the modelling process, where the goal is to help developers define and understand the cloud security requirements of the system-under-design.

Lockheed Martin's cyber kill chain (Martin, 2014) describes an multi-layered defence approach in order to segment, analyze and mitigate a cyber attack. This process includes seven stages; reconnaissance, weaponization, delivery, exploitation, installation, command and control and actions on objectives. While the cyber kill chain is intended to consider the mitigation strategy once an intrusion takes place, the scope of our work focuses on eliciting the security requirements of a cloud computing system to prevent intrusions during the requirements stage. Hahn et al. proposes a security analysis framework (Hahn, Thomas, Lozano & Cardenas, 2015) utilising a cyber–physical kill-chain, which builds upon the cyber kill chain to include cyber–physical systems. While cloud computing systems fall within the domain of cyber–physical systems, the approach proposed by Hahn et al. has a strong focus on physical systems and their constituent parts, for example an unmanned aerial system. We argue that a stronger focus on capturing and modelling the social and software components and relationships in cloud computing systems is required, in addition to the infrastructure encompassing a cloud computing environment.

Structured Threat Information Expression (STIX) is an open source language and serialization format used to exchange cyber threat intelligence (Barnum, 2012). The STIX standards are governed by the OASIS Cyber Threat Intelligence Tech-

nical Committee (CTI TC). The latest version of STIX as of July 2017 is STIX 2.0. STIX aims to provide a more expressive sets of indicators through the specification, capture, characterization and communication of standardized cyber threat information in the practice of cyber threat information sharing. The twelve key objects in the STIX language is as follows; attack pattern, campaign, course of action, identity, indicator, intrusion set, malware, observed data, report, threat actor, tool, vulnerability. While this language is specialised in the domain of threat modelling, it lacks the expressions for describing requirements-stage or cloud computing specific concepts in the scope of our work. Specifically it lacks the concept of goal-oriented requirements engineering approaches, in order to capture the social and organisational objectives and intents of actors in a cloud computing system. The focus of STIX is placed towards the actions around threat actors and their approach towards compromising a system. Our focus in this work is to support developers in expressing the security requirements of cloud computing systems, through the process of modelling their organisational environment to capture information on the organisational, application and infrastructure level.

We have carried out a literature review of modelling languages and approaches in the requirements engineering and security requirements engineering fields, several of which are extended by the Secure Tropos methodology. We have chosen to extend the Secure Tropos methodology (Mouratidis, 2011) because it combines concepts and approaches from agent-oriented, goal-oriented and security-oriented requirements engineering domains. This allows us to capture cloud computing characteristics from a high levels of abstraction, while being able to model stakeholder goals and security needs. In addition the methodology has a visual language component which defines graphical notation for concepts and relationships to represent software systems. Visual representations of system models provides a holistic view of cloud systems, supporting developers in understanding relationships and reason about security requirements from multiple perspectives. The Secure Tropos methodology begins from the early requirements stage, leading to late requirements, design level and finally the implementation stage.

2.3 Secure Tropos

To recap, Secure Tropos is a security-enhanced, goal-oriented requirements engineering methodology which extends the Tropos methodology. The Tropos methodology is an agent-oriented requirements engineering approach, which is based on the i* framework, a goal-oriented requirements engineering approach supporting actor and goal concepts from the early phase of system modelling.

2.3.1 Secure Tropos Modelling Language

We now present the concepts we will extend from the Secure Tropos modelling language, following a definition, notation and example format.

Definition 2.3.1. Actor An actor represents an entity that has intentionality and strategic goals within a software system or around the organisational environment (E. Yu, 2011; Mouratidis & Giorgini, 2007).

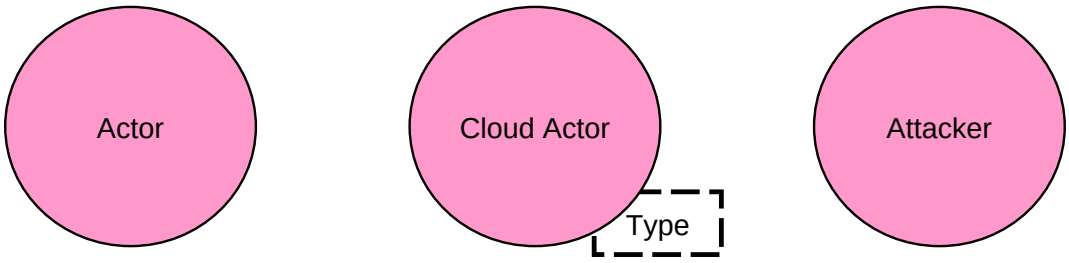

Figure 2.1: From left to right, the graphical notation of an actor, cloud actor and malicious actor.

Secure Tropos represents stakeholders, entities or roles using the notion of an actor, drawing from goal-based modelling approaches. The common categories of stakeholders represented as actors are direct or indirect stakeholders, end-users of the system and domain-specialists involved throughout the development process. Entities can be components of a system or a physical entity.

Figure 2.1 shows the graphical notation for an actor, cloud actor (Specialisation of Actor) and malicious actor (See Definition 2.3.2). The actors are visualised as a

pink circle with a textual description denoting the instance name. The cloud actor also has a rectangle box with a shaded outline denoting the *type* of service model associated with the cloud actor.

Definition 2.3.2. Malicious actor This is a specialisation of the actor that represents a stakeholder with the intentionality and strategic goals related to breaking the security of the system (Mouratidis, 2011).

The graphical notation for a malicious actor is shown in Figure 2.1 as the pink circle on the far right. The concept of an actor in requirements engineering approaches such as Tropos (Bresciani et al., 2004) and i* (Eric, 2009) do not explicitly capture entities with goals and intentions focusing on compromising systems or assets through misuse, exploits or threats. Thus a specialised class of actor is defined in security requirements engineering approaches in order to capture stakeholders with the intention of causing harm on the system, in this case the concept of an attacker is proposed in Secure Tropos as a malicious actor. The malicious actor perform attacks on systems in order to exploit vulnerabilities and compromise assets.

Consider a malicious insider, a threat to an organisation which is initiated from stakeholders within the organisation. We are able to identify and model malicious insiders using the concept of a malicious actor in order to differentiate between a typical employee and an employee with malicious intent, such as compromising the system by gaining access to credit card details through the abuse of authorised access privileges.

Definition 2.3.3. Goal A goal represents the strategic interests of stakeholders within the context of a system (Mouratidis & Giorgini, 2007).

Figure 2.2: Graphical notation of a goal as a dark green elongated oval.

The general description of a goal in requirements engineering is a way for actors to achieve objectives (Eric, 2009; Darimont et al., 1997; Bresciani et al., 2004). In

Secure Tropos a goal represents the strategic interests of stakeholders within the context of a system (Mouratidis & Giorgini, 2007). Figure 2.2 shows the graphical representation of a goal as a dark green elongated oval.

Definition 2.3.4. Resource A resource represents a physical or informational entity that one of the actors requires (Bresciani et al., 2004).

Figure 2.3: Graphical notation of a resource as a stretched yellow rectangle.

The graphical notation for a resource is shown in Figure 2.3 as a stretched yellow rectangle with a textual description in the middle representing the name of the resource. The main concern when dealing with resources is whether the resource is available and who is responsible for its delivery.

Definition 2.3.5. Threat Threats represent circumstances that have the potential to cause loss; or problems that can put the security features of the system in danger (Low, Mouratidis & Henderson-Sellers, 2010).

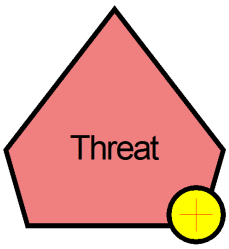

Figure 2.4: Graphical notation of a threat as a light red pentagon.

The graphical notation for a threat is shown in Figure 2.4 as a light red pentagon with a textual description in the middle representing the name of the threat. The concept of a threat is crucial when performing any security analysis on a system, as it describes circumstances under which security properties are violated; leading to

an insecure system. Here we describe threats in a system using high-level descriptions, as we do not have sufficient information at the early requirements stage to describe fine-grained technical attributes from a security requirements engineering perspective.

Definition 2.3.6. Vulnerability A weakness of an asset or group of assets that can be exploited by one or more threats (ISO & Std, 2011)

Figure 2.5: Graphical notation of a vulnerability as a dark red elongated oval.

The graphical notation for a threat is shown in Figure 2.5 as a dark red elongated oval with a textual description in the middle representing the name of the vulnerability. "Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, and attacker capability to exploit the flaw." (Hughes & Cybenko, 2014). In this context, a vulnerability is also known as the attack surface.

Definition 2.3.7. Attack method An action aiming to cause a potential violation of security in the system (Mouratidis, 2011).

Figure 2.6: Graphical notation of an attack method as a dark yellow heptagon.

The graphical notation for an attack method is shown in Figure 2.6 as a dark yellow heptagon with a textual description in the middle representing the name of the attack method. An attack method embodies a specific way of carrying out a

threat in order to exploit vulnerabilities in the system, causing harm to assets within the system and the system itself.

Definition 2.3.8. Security constraint Security Constraints are used in the Secure Tropos methodology to represent security requirements; where a security requirements is defined as "a manifestation of a high-level organisational policy into the detailed requirements of a specific system." (Mouratidis, 2011).

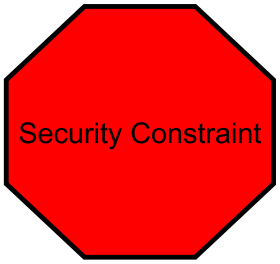

Figure 2.7: Graphical notation of a security constraint as a red octagon.

In Secure Tropos, a security constraint is a specialisation of the concept of a constraint (Mouratidis, 2011). Figure 2.7 shows the graphical notation for a security constraint, which is represented as a red octagon. In the context of software engineering, a constraint is usually defined as a restriction that can influence the analysis and design of a software system under development by restricting some alternative design solutions,by conflicting with some of the requirements of the system, or by refining some of the system's objectives. In other words, constraints can represent a set of restrictions that do not permit specific actions to be taken or prevent certain objectives from being achieved. Often constraints are integrated in the specification of existing textual descriptions. However, this approach can often lead to misunderstandings and an unclear definition of a constraint and its role in the development process. Consequently, this results in errors in the very early development stages that propagate to the later stages of the development process causing many problems when discovered; if they are discovered. Therefore, in the Secure Tropos modelling language we define security constraints, as a separate concept. To this end, the concept of security constraint has been defined within the context of Secure Tropos

as: *A security condition imposed to an actor that restricts achievement of an actor's goals, execution of plans or availability of resources.* Security constraints are outside the control of an actor. This means that, differently than goals, security constraints are not conditions that an actor wishes to introduce but it is forced to introduce.

Definition 2.3.9. Security objective A security objective represents a set of principles or rules that contribute towards the achievement of the system's security (Mouratidis, 2011).

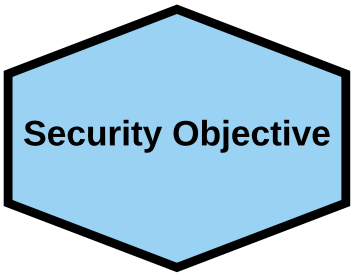

Figure 2.8: Graphical notation of a security objective as a blue hexagon.

Figure 2.8 illustrates the graphical notation of a security objective, shown as a blue hexagon. This concept is similar to the notion of a plan in requirements engineering, which describes at an abstract level one approach for satisfying a goal. However the security objective focuses on security-oriented concepts, in that it provides a description of the security-related criteria required to satisfy security needs. In this sense a security objective embodies security policies as it systematically provides description of conditions and steps required to ensure security needs are enforced in a system. Thus security objectives are used when there exists unsatisfied security needs in a system, indicating how a security need can be addressed and the steps required to enforce this process.

Definition 2.3.10. Security mechanism A security mechanism represents standard security methods for helping towards the satisfaction of the security objectives (Mouratidis, 2011).

Figure 2.9: Graphical notation of a security mechanism as a green stretched hexagon.

The graphical notation for a threat is shown in Figure 2.9 as a stretched green hexagon with a textual description in the middle representing the name of the security mechanism. Some standard security methods are able to prevent security attacks, whereas others are able only to detect security breaches. It must be noted that furthered analysis of some security mechanisms is required to allow developers to identify possible security sub-mechanisms. A security sub-mechanism represents a specific way of achieving a security mechanism. For instance, authentication denotes a security mechanism for the fulfilment of a protection objective such as authorisation. However, authentication can be achieved by sub-mechanisms such as passwords, digital signatures and biometrics.

A security objective may be enforced by one or more security mechanisms, where the security mechanisms may share joint responsibility in order to enforce the security objective; in this case the security mechanisms are annotated with the *"AND"* notation to indicate the need to implement all connected security mechanisms to ensure the enforcement of a security objective. It is also possible to model alternative options, where any of the proposed security mechanisms are able to enforce a security objective without depending on the implementation of other security mechanisms. This is indicated by the *"OR"* notation, where each linked security mechanism can be independently implemented in order to enforce a security objective. It is the security experts responsibility for selecting and realising security mechanisms during the implementation stage, where they decide the most suitable security mechanism through our cloud models. In order to facilitate this decision, we provide attributes in our concepts which contribute towards the weighting determining the degree of satisfaction and suitability of security concepts.

2.3.2 Secure Tropos Syntax

This subsection discusses the syntax of the Secure Tropos modelling language in relation to Secure Tropos Views. Secure Tropos is a security requirements engineering methodology aimed at fully capturing the properties of software systems and the organizational environment, focusing on modelling security (Mouratidis & Giorgini, 2007). The language extends the concepts of an (social) actor, goal, task, resource and social dependency from the i* modelling language and redefining existing concepts introduced in the Tropos language and development process (Nhlabatsi et al., 2010). The Secure Tropos methodology closely follows the software development lifecycle with emphasis on security and privacy requirements, allowing the developer to incrementally refine models of the system-to-be during the analysis and design stage.

The Secure Tropos notation is fully defined in (Mouratidis & Giorgini, 2007). The meta-model of the Secure Tropos methodology is presented in Figure 2.10. The white boxes indicate different classes in the modelling language. The grey boxes indicate the relationships which link different classes together. The concrete notation is presented within *views* below, where each view denotes a specific phase of activity in the modelling process. We now discuss Secure Tropos Views.

Organisational View

The diagram in Figure 2.11 illustrates the main nodes of an organisational view of Secure Tropos. It depicts a node-link diagram enclosed in a bounding rectangle. The nodes in the node-link diagram vary in shape according to the type of Secure Tropos element that they depict. The links similarly vary.

1) Actor: The circular node depicts an *actor*. An actor can be a physical or abstract manifestation, with strategic goals and intentions. An example actor labelled "Actor 1" can be seen in Figure 2.11.

2) Goal: The semi-oval node depicts a *goal*. Goals represent an actors strategic interests, which can be decomposed into sub-goals and combined using Boolean operations. An example goal labelled "Goal 1" can be seen in Figure 2.11. Goals are linked through a *Dependency* link, depicted by one semi-circles on each side of the goal element.

3) Dependency: A *Dependency* link indicates that an actor depends on another

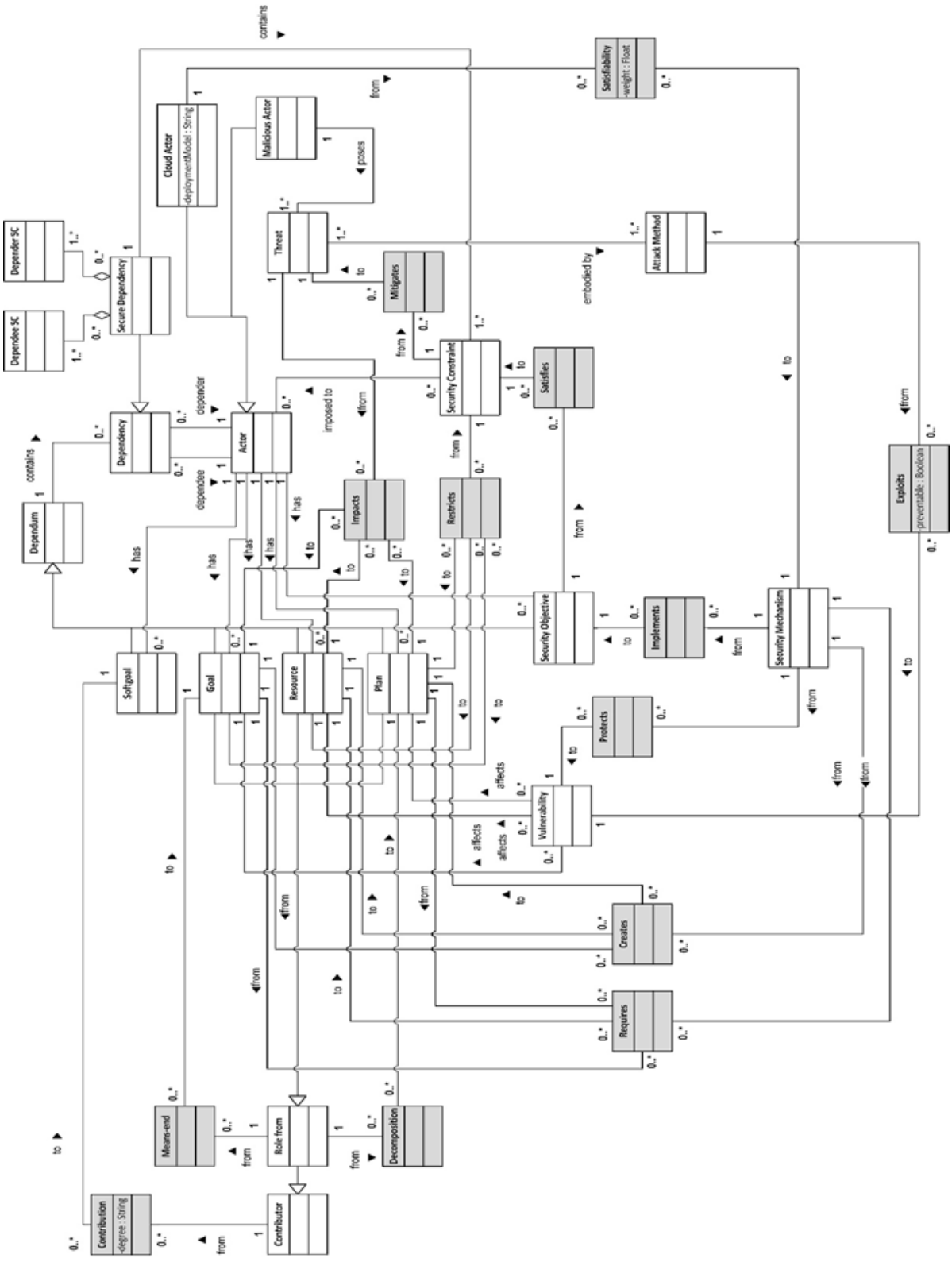

Figure 2.10: The Meta-Model of the Secure Tropos methodology

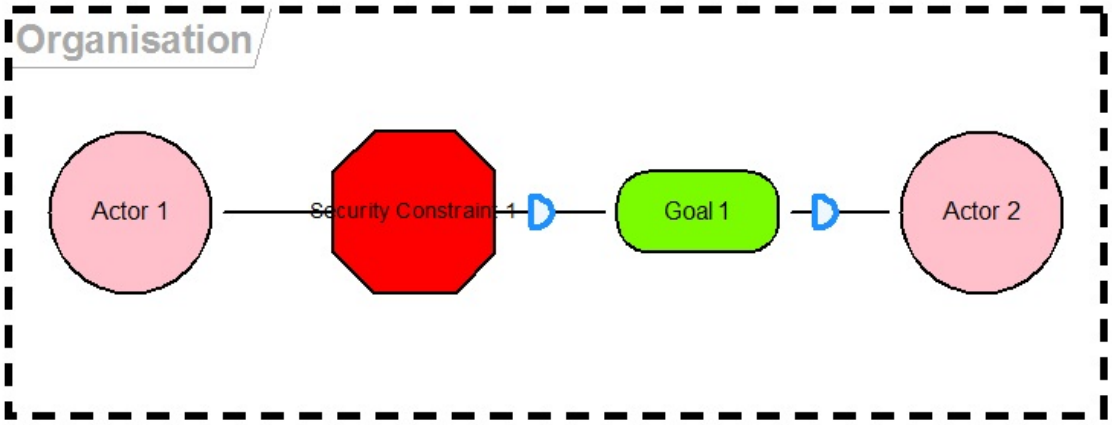

Figure 2.11: An example of an organisational view in Secure Tropos

actor in order to achieve some goal or to obtain a resource, where the direction the semi-circle is pointing towards denotes the dependee. An example dependency link can be found linking the goal "Goal 1" with the actor "Actor 1" who depends on the actor "Actor 2" to achieve the goal.

4) Security Constraint: *Security Constraints* are depicted by the octagon node. Security constraints define security requirements through a set of restrictions that limit the way goals can be carried out. An example of a security constraint "Security Constraint 1" can be found from the actor "Actor 1" to the goal "Goal 1".

5) Secure Dependency: A *secure dependency* defines a case where given a dependum of type goal or resource with a depender and dependee actor, the depender actor imposes on the dependee actor zero or more security constraints which restricts the dependum and conversely the dependee actor imposes on the depender actor zero or more security constraints which restricts the dependum. As an example the depender actor "Actor 1" has the security constraint "Security Constraint 1" restricting the dependum "Goal 1", where the security constraint is imposed on the dependee actor "Actor 2".

Security Requirements View

The diagram in Figure 2.12 illustrates the security requirements view, which provides a detailed analysis of the organisational view. This view depicts a node-link diagram enclosed in a bounding circle, defined by an actor that is delegated as the solution

"system". Several new elements are introduced in this view.

1) Plan: The elongated hexagon node depicts a *Plan*. A plan specifies the details and conditions under which a goal or measure is operationalised. "Plan" is an example of a plan that is linked to a goal, in this case "Sub Goal 1".

2) Resource: The rectangle node depicts a *Resource*. *Resource* represent a physical or virtual entity. *Resource* can be linked to goals using a *Requires* link. An example of a resource is "Resource" which is linked to the goal "Sub Goal 1" via a requires link. The requires link indicates that the goal requires this specific instance of a resource in order achieve and satisfy the goal.

3) Threat: The pentagon node depicts a *Threat*. A threat indicates the potential loss or problems that can put the system at risk. For example, "Threat" is linked via the *Impacts* link to the goals "Sub Goal 1" and "Sub Goal 2", indicating that both goals are impacted by this threat.

4) Security Objective: The hexagon node depicts a *Security Objective*. An example of a security objective addressing a security constraint is indicated by "Security Objective", which is linked to "Security Constraint" via the *Satisfies* link. The *Security Mechanism* "Security Mechanism 1" and "Security Mechanism 2" fulfils the *Security Objective*, which is indicated by the *Implements* link.

5) Security Mechanism: The hexagon node with two parallel horizontal lines depicts a *Security Mechanism*. A security mechanism is a method or procedure that enforces security objectives.

6) Restricts: The *Restricts* link shows that the security constraint places a restriction upon a goal, as an example "Security Constraint" which restricts the goal "Sub Goal 2".

7) Service Identification: As part of the ongoing work towards modelling and analysing cloud-based security, we have proposed a **Goal-Plan-Resource** pattern for identifying services based on the existing notations found in Secure Tropos (Shei, Delaney et al., 2015). We are then able to build around the services to generate an infrastructural view of a cloud deployment, thus allowing us to identify threats and vulnerabilities on the cloud level.

Cloud Analysis View

In the cloud analysis view shown in Figure 2.14, we provide a visual indication to

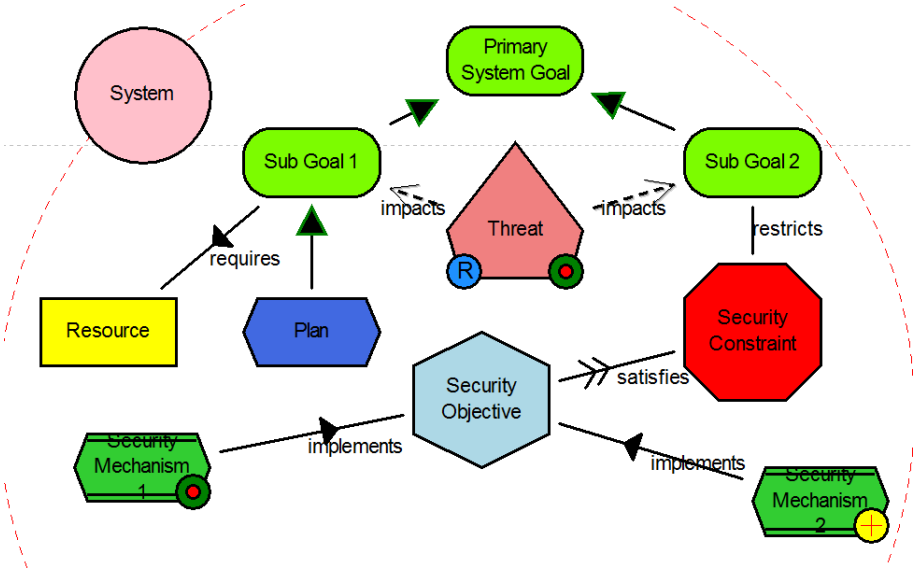

Figure 2.12: An example of a security requirements view in Secure Tropos

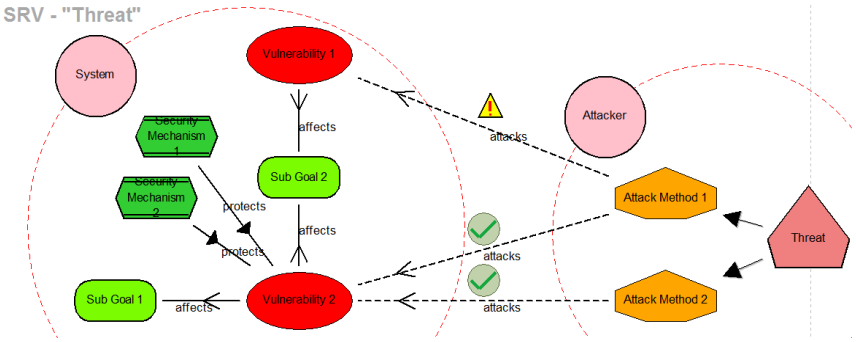

Figure 2.13: An example of a security attacks view in Secure Tropos

facilitate the selection of appropriate cloud service providers based on the security requirements identified in the previous views. In particular, we evaluate how specific service providers satisfy the security mechanisms identified in the previous views.

1) Cloud Service Provider: The circular node indicates a cloud service provider, which provides resources such as services and infrastructure.

2) Satisfiability: A metric between *not satisfied (0)* and *fully satisfied (1)* indicates how well the cloud service provider satisfies the linked security mechanism.

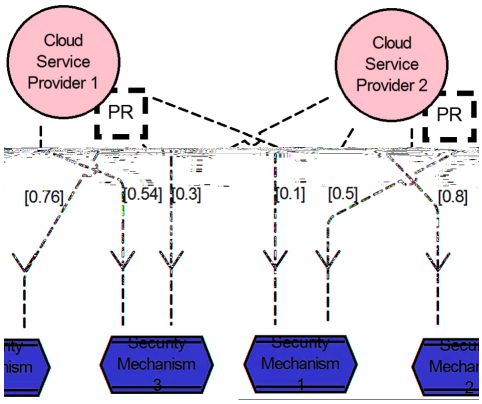

Figure 2.14: An example of a cloud analysis view in Secure Tropos

SecTro2 Tool

"SecTro2"[1] is a CASE tool based on the ADOxx meta-modelling platform, which allows system modelling using Secure Tropos methodology. Besides standard modelling activities it also aids developer in validating created models and running various analyses against them. "SecTro2" supports generating graphical images as well as producing Word and PDF reports of the created models and their features.

SecTro's workspace consists of the drawing canvas in the centre, on the top there is a series of tabs for showing the developed diagrams for each stage of Secure Tropos and on the left a toolbox containing the graphical representations of all the concepts of Secure Tropos (Pavlidis, Islam & Mouratidis, 2011). The different supported views along with their notations have been introduced in the previous section.

2.4 The Secure Cloud Environment Framework

We have presented the semantics and syntax of the Secure Tropos modelling language in Section 2.3.1 and Section 2.3.2, providing the background knowledge required to understand our work. We now discuss the extension of Secure Tropos concepts through our Secure Cloud Environment Framework.

We now propose a framework in order to answer our three research questions.

[1] Available at: http://austria.omilab.org/psm/content/sectro/info

In our work we examine cloud computing systems from a security requirements engineering perspective. Specifically we define a visual language to graphically model secure cloud systems from the early requirements stage. Our approach for modelling secure cloud systems is based on the components shown in Figure 2.15, namely defining a modelling language, procedure and analysis techniques. The Secure Cloud Environment framework consists of three components; the Cloud Security Modelling Language combines the concepts in cloud computing and security engineering, the Secure Cloud Process provides a systematic approach for modelling and analysing cloud systems and a set of formal analysis concepts with three cloud analysis techniques is proposed to enable the semi-automatic analysis and enrichment of cloud security requirements.

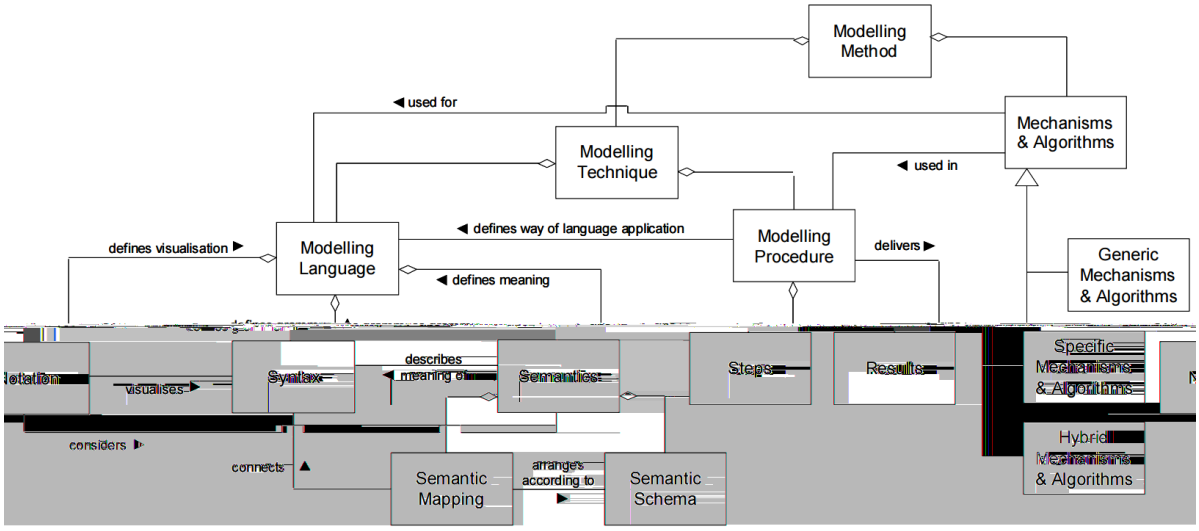

Figure 2.15: Components of modelling methods from Karagiannis D. & Kühn H. (Karagiannis & Kühn, 2002).

2.4.1 Overview of the Secure Cloud Environment Framework

An overview of the framework is shown in Figure 2.16. In order to fulfil the requirements of the framework specified previously, our Secure Cloud Environment Framework will address these issues in a systematic and semi-automated manner.

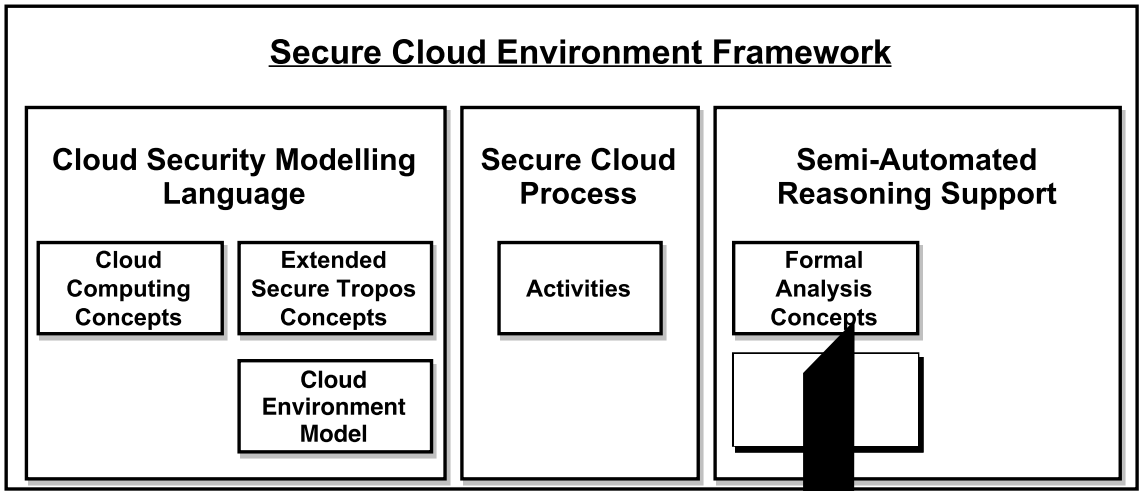

Figure 2.16: An overview of the Secure Cloud Environment Framework.

In the framework we provide:

- The Secure Cloud Modelling Language to define the concepts and relationships required when modelling cloud-based systems at varying degrees of granularity, capturing the level of abstraction required for generating cloud security requirements.
- The Secure Cloud Process describing the activities which practitioners should follow, in order to apply our concepts and construct models representing cloud system components, properties and security needs.
- Semi-Automated Reasoning Support defines formal analysis concepts and cloud security techniques to facilitate semi-automated reasoning. Tool-support is provided through extensions to an existing visual modelling tool.

2.5 Chapter summary

In this chapter we have reviewed literature on cloud computing characteristics, properties and provided an overview of security challenges and issues in cloud computing. We then examined security requirements engineering approaches in the literature,

focusing on goal-oriented methodologies and languages. With goal-oriented approaches we are able to capture stakeholders, their security needs and how to achieve them. We have reviewed approaches focusing on the early requirements stage, in order to describe systems at an abstract level without considering implementation details. We highlighted approaches supporting a visual modelling language to graphically represent abstract models of systems. A selection of reasoning techniques were presented in the supported approaches, such as validation of models and elements, constraint resolution, propagation and evaluation of satisfaction of security needs.

Thus in our approach we have extended the Secure Tropos methodology. We address the language limitations of the Secure Tropos methodology by extending the expressiveness of the modelling language with cloud computing concepts. We then propose enhancements for a systematic approach to support developers in the modelling and understanding of security requirements in cloud computing systems. Finally we define three types of cloud analysis in order to support semi-automated reasoning in order to validate and derive cloud security requirements from cloud environment models.

Chapter 3

Cloud Security Modelling Language

In this chapter we present our cloud modelling language which enables the expression of concepts and relationships of cloud computing systems from a security requirements engineering perspective.

In Section 3.1 we specify the requirements of the cloud modelling language in order to answer our research question (RQ1 in Section 1.4) and research objectives (RO1, RO2 in Section 1.4.2). In Section 3.2 we present our concepts for describing cloud computing from a security requirements engineering perspective. In Section 3.3 we define the relationships between our cloud computing concepts and security concepts. In Section 3.4 we define the properties of our concepts, in order to capture cloud security characteristics from a fine-grained perspective. The concrete syntax and instance syntax of our cloud modelling language is presented in Section 3.5. Finally in Section 3.6, we present four models capturing layer specific security properties in cloud computing systems.

3.1 Requirements of the Modelling Language

We have determined through the survey of the literature, the cloud computing characteristics and security concepts which need to be captured, in order to model and understand security requirements in cloud computing environments. In order to an-

swer the research question RQ1 and address the research objectives RO1 and RO2, we now define the requirements of the modelling language:

1. Extending the Secure Tropos modelling language in order to capture security concepts in the cloud computing domain.
   (a) The modelling language will capture the concept of stakeholders from a cloud computing perspective. **Addressed by: Def. 3.4.1**
   (b) The modelling language will capture the concept, specifications and relationships of a cloud service. **Addressed by: Def. 3.2.1, Def. 3.4.3**
   (c) The modelling language will distinguish between the physical and virtual components of cloud computing assets. **Addressed by: Def. 3.2.2, Def. 3.2.3, Def. 3.2.4**
   (d) The modelling language will support instance-specific descriptions of cloud computing characteristics using properties. **Addressed in: Section 3.4, Def. 3.4.1 to Def. 3.4.14**
2. The modelling language will aggregate platform-specific concepts in cloud computing systems through fine-grained models.
   (a) The modelling language will describe a model focusing on organisational concepts and relationships. **Addressed by: Def. 3.6.1**
   (b) The modelling language will describe a model focusing on application-oriented concepts and relationships. **Addressed by: Def. 3.6.2**
   (c) The modelling language will describe a model focusing on infrastructure-based concepts and relationships. **Addressed by: Def. 3.6.3**

3.2 Cloud Computing Concepts

In this section we discuss the concepts required to model cloud computing systems. We demonstrate how our concepts are applied from a practitioners perspective using the health-care example introduced in Section 1.3. Figure 3.1 illustrates the concepts

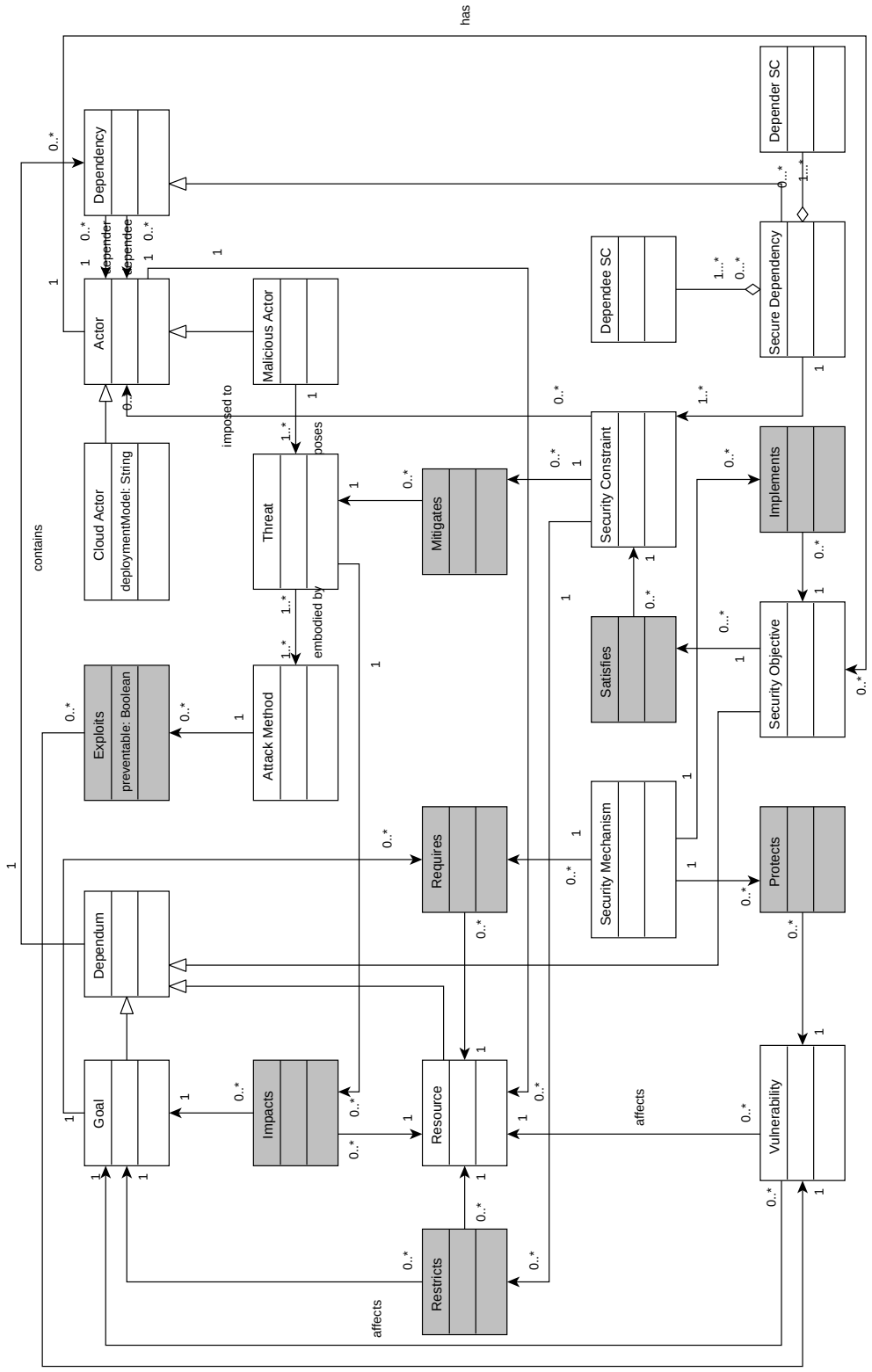

Figure 3.1: A fragment of the Secure Tropos metamodel showing the concepts which are extended in our work.

from a fragment of the Secure Tropos metamodel (modelled in UML) which are extended in our work.

We now present our cloud computing extensions by systematically constructing a metamodel, highlighting each addition of a concept through a definition, description and example format.

Definition 3.2.1. Cloud service A cloud service provides a specific computing capability, is managed and owned by actors and requires virtual and physical resources in order to deliver its capability.

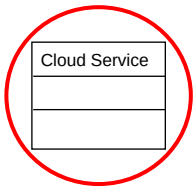

Figure 3.2: The conceptual representation of the cloud service in the metamodel.

Figure 3.2 highlights the concept of the cloud service in our metamodel. The cloud service concept is a specialisation of the goal concept because a cloud service embodies a way to achieve a specific stakeholder need through cloud computing capabilities. The cloud computing capability is delivered through a combination of virtual and physical resources, which are owned or managed by various stakeholders.

Definition 3.2.2. Virtual Resource A virtual resource represents intangible assets in a cloud computing system.

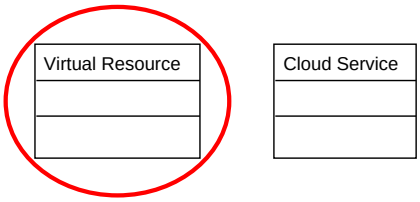

Figure 3.3: The conceptual representation of the virtual resource in the metamodel.

Figure 3.3 highlights the concept of the virtual resource in our metamodel. In order to differentiate between tangible and intangible resources, we create a specialisation of the resource concept to represent intangible resources as virtual resources. An example of an intangible resource is patient data, which has an owner representing an entity that produces the original copy as well as responsible parties representing entities handling the data.

Definition 3.2.3. Physical Infrastructure A physical infrastructure represents a tangible system which, given a geographical location, hosts a group of physical assets within its local proximity.

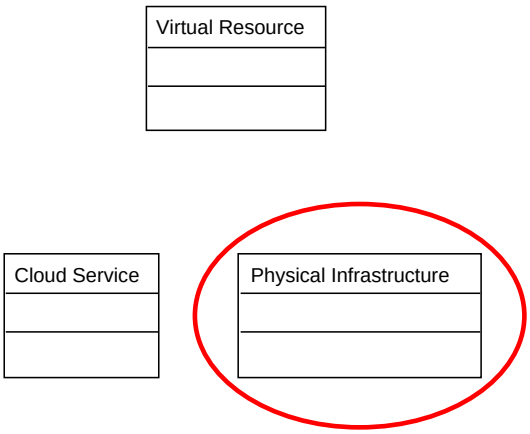

Figure 3.4: The conceptual representation of the physical infrastructure in the metamodel.

Figure 3.4 highlights the concept of the physical infrastructure in our metamodel. We define this concept as a specialisation of the resource concept, given that cloud computing resources are hosted in physical infrastructure such as a data-centre. This is essential as properties belonging to the physical infrastructure contain fields such as geographical location, ownership and responsible parties; which is required for performing security analysis.

Definition 3.2.4. Infrastructure Node An infrastructure node represents a single instance of a computing component such as a server, data storage or network connection.

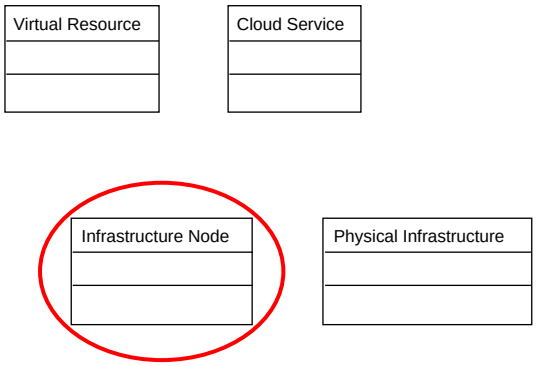

Figure 3.5: The conceptual representation of the infrastructure node in the modelling language.

Figure 3.5 highlights the concept of the infrastructure node in our metamodel. In this case a tangible resource is defined as a specialisation of the resource concept using the notion of an infrastructure node. Each infrastructure node is part of a physical infrastructure, to conceptually represent that an infrastructure node is physically hosted within a structure or area. This allows us to capture the properties of individual nodes through fields such as multi-tenancy, physical location and responsible parties. This is essential for capturing cloud computing concepts at the physical components level, in order to facilitate cloud security analysis from a security requirements engineering perspective.

3.3 Relationships

In this section we outline the relationships linking together concepts from the cloud computing and security requirements engineering domains. We now define the relationships in our modelling language by building upon the metamodel in the previous section, following a definition, description and example format.

Definition 3.3.1. Specialisation of Goal The Cloud Service concept is a specialisation of the goal concept.

A cloud service represents a fine-grained way to achieve a goal based on the cloud computing paradigm, in the context of a cloud computing system. A cloud service

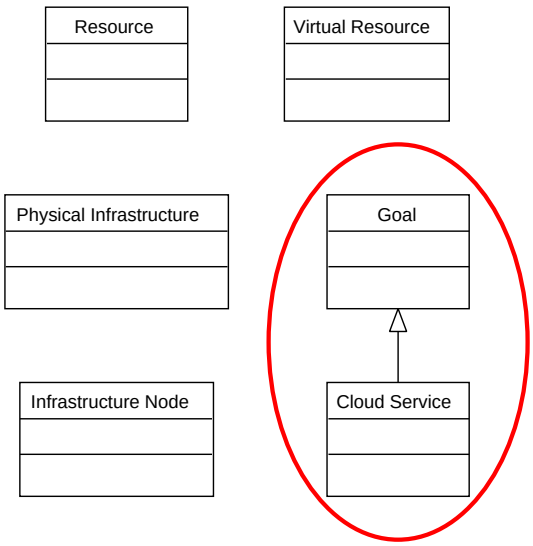

Figure 3.6: The cloud service is a specialisations of the goal concept.

involves relationships with concepts such as actors, resources and dependencies. Thus we define the cloud service concept as a specialisation of the goal concept, which allows the inheritance of properties and relationships from the goal concept. This relationship is indicated in Figure 3.6, where the arrow shows that the cloud service concept is a specialisation of the goal concept.

Definition 3.3.2. Specialisations of Resource The Virtual resource, infrastructure node and physical infrastructure are all specialisations of the resource concept.

In order to model the concept of tangible and intangible resources, we define three specialisations using the resource concept. The first specialisation is the virtual resource which represents virtual data, the infrastructure node represents individual physical nodes, while the physical infrastructure represents a structure or container composed of zero or more infrastructure nodes. This relationship is indicated in Figure 3.7, where the arrow shows that the virtual resource concept, infrastructure node concept and the physical infrastructure concept are all specialisations of the resource concept.

Definition 3.3.3. Infrastructure Node Composition The infrastructure node concept is part-of the physical infrastructure concept, indicating that a physical

Figure 3.7: The virtual resource, infrastructure node and physical infrastructure are all specialisations of the resource concept.

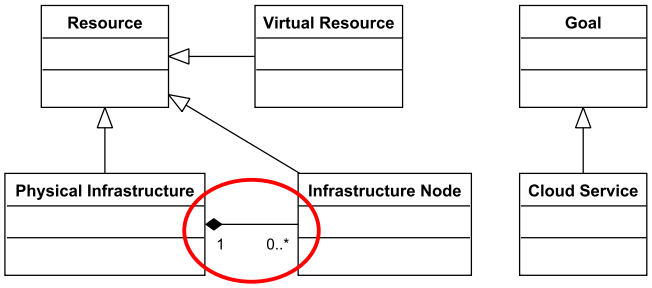

Figure 3.8: The composition relationship from infrastructure node concept to the physical infrastructure concept.

infrastructure can contain zero or more instances of infrastructure nodes.

Figure 3.8 highlights the composition relationship between the infrastructure node concept and the physical infrastructure concept. A physical infrastructure concept has zero or more infrastructure nodes, which conceptually represents the grouping of physical computational resources within an infrastructure.

Definition 3.3.4. Permeates This indicates the relationship which interrelates data-in-transit and data-at-rest from the virtual resource concept to the infrastructure node concept.

Note that we follow the notation used in the Secure Tropos metamodel, which is loosely based on version 2.0 of Unified Modelling Language(UML). Figure 3.9 highlights the *permeates* relationship between the virtual resource and infrastructure node concept. A virtual resource is said to be traceable to an infrastructure node

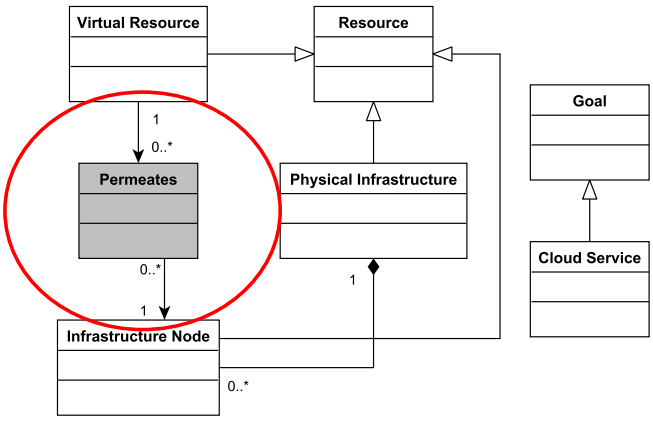

Figure 3.9: The permeates relationship between the virtual resource and infrastructure node concept.

if the physical component hosts the virtual resource. For example if user data is stored on a physical hard drive, the data is traceable to the hard drive.

Definition 3.3.5. Requires A goal, cloud service or resource requires a cloud service or resource, in order to satisfy a stakeholder need, fulfil a capability or collaborate with other resources or cloud services.

Figure 3.10 highlights the *Requires* relationship between the goal, cloud service and resource concepts. This relationship indicates the resource or cloud service instances required by a goal, cloud service or resource. A cloud service requires zero or more resources to deliver their capability, where the resource can be of the type virtual resource, physical infrastructure or infrastructure node. For example the cloud service *Patient Details Service* requires *Virtual Resource: Patient Data*, indicating that the cloud service requires digital patient records to perform computational tasks such as creating, editing and deleting data.

Definition 3.3.6. Owns Indicates an actors' level of responsibility where they possess ownership over a physical asset, is the creator of a virtual asset or has data ownership over a virtual asset.

Figure 3.11 highlights the *Owns* relationship between the actor, resource and cloud service concepts. This relationship is used to depict the level of responsibility

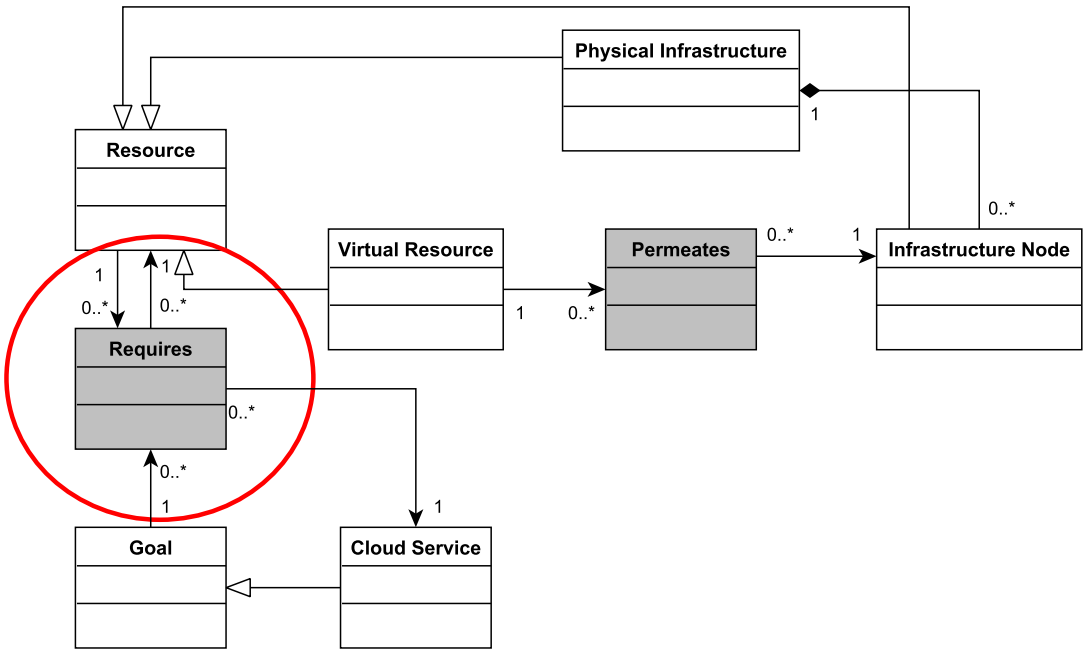

Figure 3.10: The requires relationship between the resource, goal and cloud service concepts.

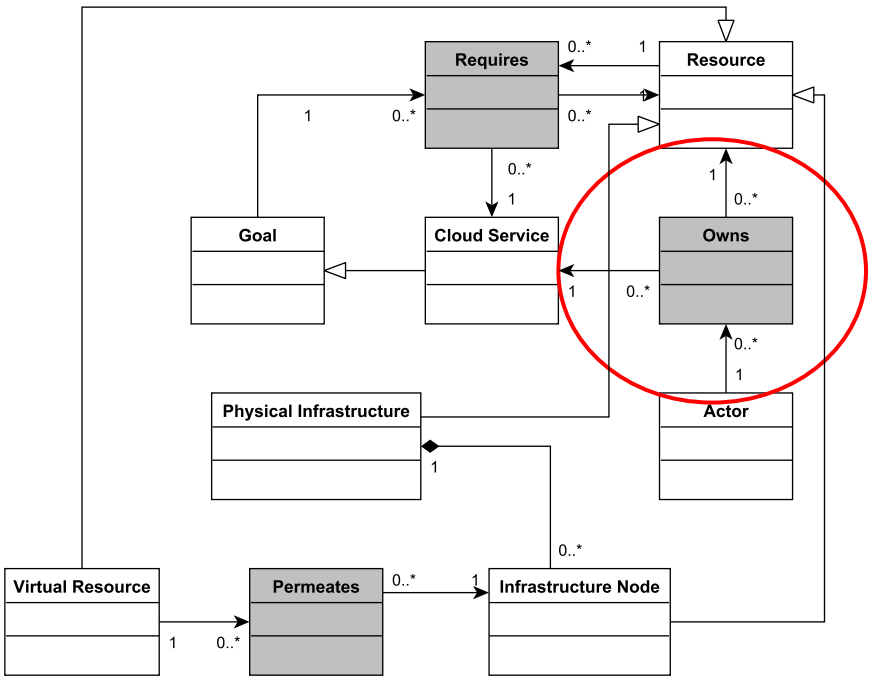

Figure 3.11: The owns relationship between the resource, actor and cloud service concepts.

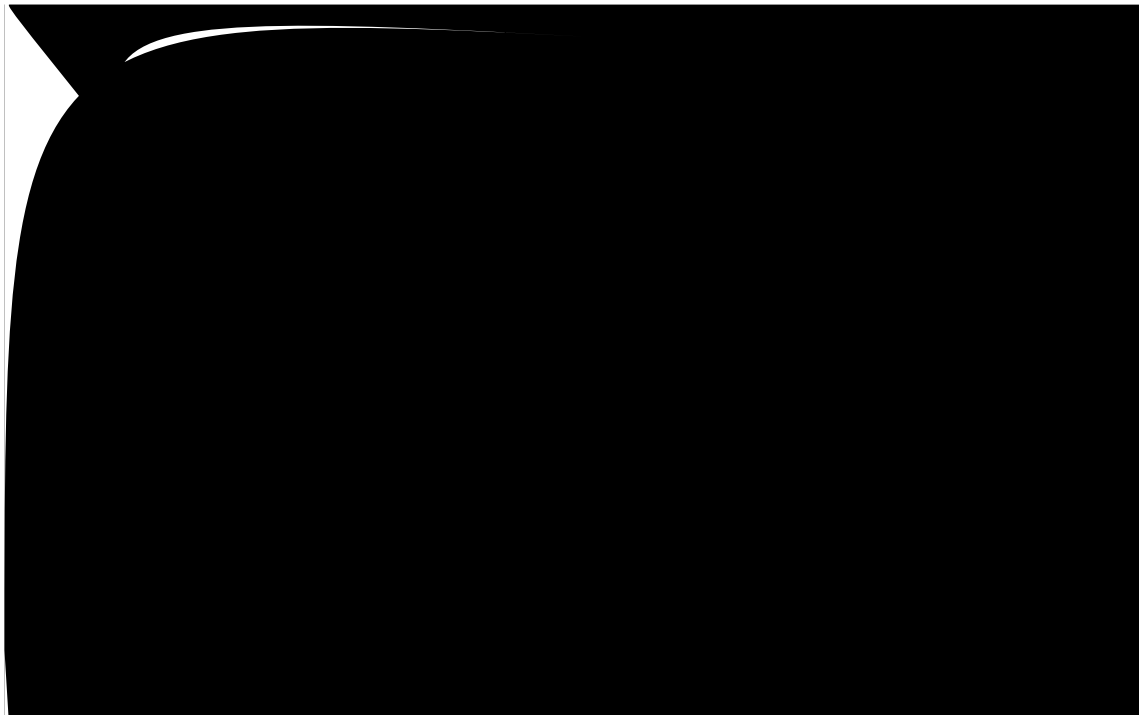

Figure 3.12: The manages relationship between the actor and cloud service concepts.

an actor posses in relation to a cloud service or resource. It is important to define the difference between the data creator, data ownership and physical ownership. The data creator refers to the case where an actor produces data, and thus is the creator of the data in the legal sense. Data ownership refers to the case where data is physically stored on assets owned by third party providers, therefore the third party providers are responsible for the handling of the data. Physical ownership refers to the case where an actor is the owner of a physical asset, such as a server or data-centre.

Definition 3.3.7. Manages Indicates an actors level of responsibility in the configuration and delivery of a cloud service.

Figure 3.12 highlights the *Manages* relationship between the actor concept and the cloud service concept. An actor can have zero or more *Manages* relationships, indicating that some actors are not involved in the management of cloud services. A cloud service can be the target of one or more *Manages* relationships from a range of actors, indicating that a cloud service is managed by one or more actors.

One or more actors are responsible for managing a cloud service. We use the term manage to represent parties responsible for providing resources required by cloud services, configuring cloud components and ensuring security and jurisdictional requirements are fulfilled.

3.4 Properties

In this section we define properties in order to describe an instance of a concept in more detail. A concept has different types of supporting information, which ranges from technical specifications to high level descriptions. For instances, given the concept of an infrastructure node, a highly abstract description of an instance would be called *"Disk Storage"*. We can further specify the *"Disk Storage"* instance by adding additional detail to properties, such as *"Multi-tenancy"* to the *Tenancy* property, *"Seagate"* to the *Vendor* property, *"Storage"* to the *ResourceType* property and *"ST2000DM001"* to the *Version* property. This allows the developer to provide both abstract and technical details to instances of a concept, therefore supporting varying degrees of granularity when describing the system-under-design. The level of analysis which can be performed on a model is dependent on the granularity of the properties provided. In other words, as more information is provided in a model, the richer the analysis results.

Definition 3.4.1. Cloud Actor The cloud actor concept has three properties; *Description* of the instance, *DeploymentModel* representing the deployment model and *Type* to determine the types of cloud actors an instance plays.

- Description: String
- DeploymentModel: String
- Type: «Enumeration» CloudActorType [1..*]

Figure 3.13 highlights the *Cloud Actor* concept with extended properties and the enumeration *CloudActorType*. The *Description* provides additional details of the instance .The *DeploymentModel* property represents the type of deployment model, which is of type String. The *Cloud Actor Type* property represents the role the cloud

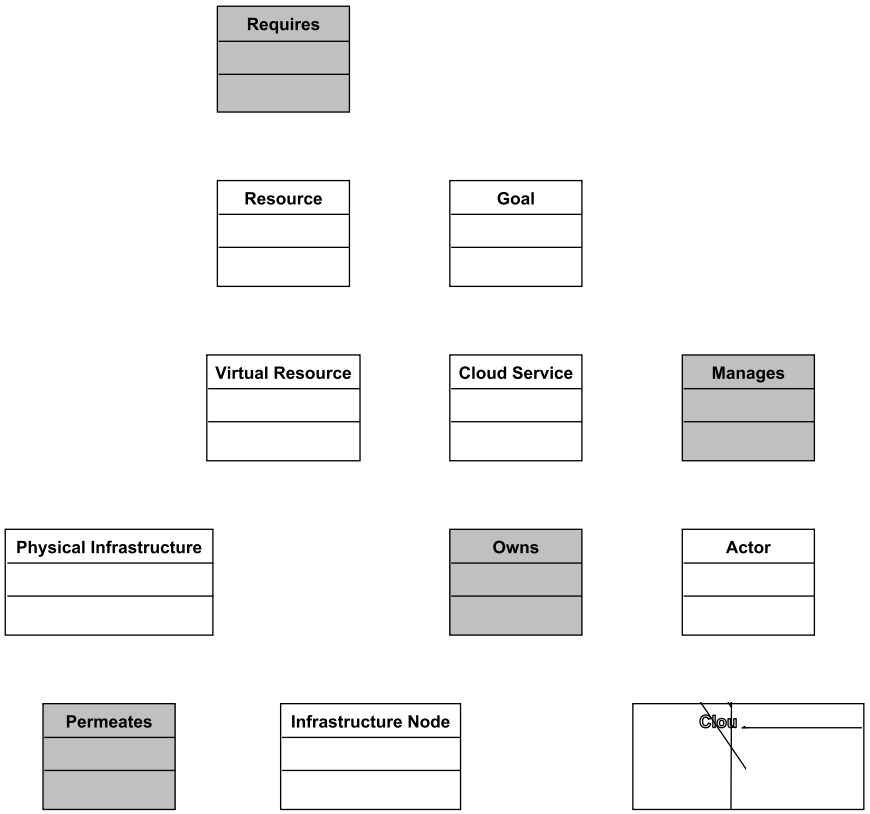

Figure 3.13: Highlighting the Cloud Actor concept with extended properties and the enumeration CloudActorType.

actor plays in a cloud computing context. The enumerated values are based on the five types of cloud actors defined by NIST in (Bohn, Messina, Liu, Tong & Mao, 2011); Cloud Service Provider, Cloud Consumer, Cloud Broker, Cloud Carrier and Cloud Auditor. The type of the cloud actor determines the level of responsibility in a *manages* or *owns* relationship. The type of the cloud actor also constrains the validity of relationships with other concepts, based on pre-defined logic or a set of rules defined by the developer. The *Cloud Actor Type* property consists of one or more values from the enumeration *CloudActorType*. This represents the case where a cloud actor may play more than one role simultaneously.

For example *A2: Hospital* is a cloud service provider because they provide cloud service *cloud service 1: Patient Details Service* to the cloud consumer *A1: Patient*. But *A2: Hospital* is themselves dependent on the cloud service provider *A5: CSP* to provide computing components such as physical infrastructure as a cloud service. Therefore in this example *A2: Hospital* is a cloud service provider due to the dependency relationship with *A1: Patient*, however *A2: Hospital* is also involved as a cloud consumer in a dependency relationship with *A5: CSP*.

Definition 3.4.2. Owns The *Owns* relationship has the property *Responsibility*, which indicates the type of ownership an actor possess in relation to a cloud service or a resource and its specialisations.

- Responsibility: «Enumeration» Ownership

Figure 3.14 highlights the *Owns* relationship and its property, in addition to the enumeration *Ownership*. An actor can initiate zero or more *Owns* relationships to a cloud service, resource or their specialisation. A cloud service, resource or their specialisation can be the target of zero or *Owns* relationships initiated from an actor. The enumeration *Ownership* contains the following values; data creator, data ownership and physical ownership. The data creator represents an actor that generates virtual resources, for example personal information created by a patient. The data ownership represents an actor in possession of virtual resources, for example a hospital is responsible for their patients medical records. The physical ownership represents actors who are responsible for processing resources through their own physical infrastructure, for example a cloud service provider has physical ownership over customer data stored on the cloud service providers physical infrastructure.

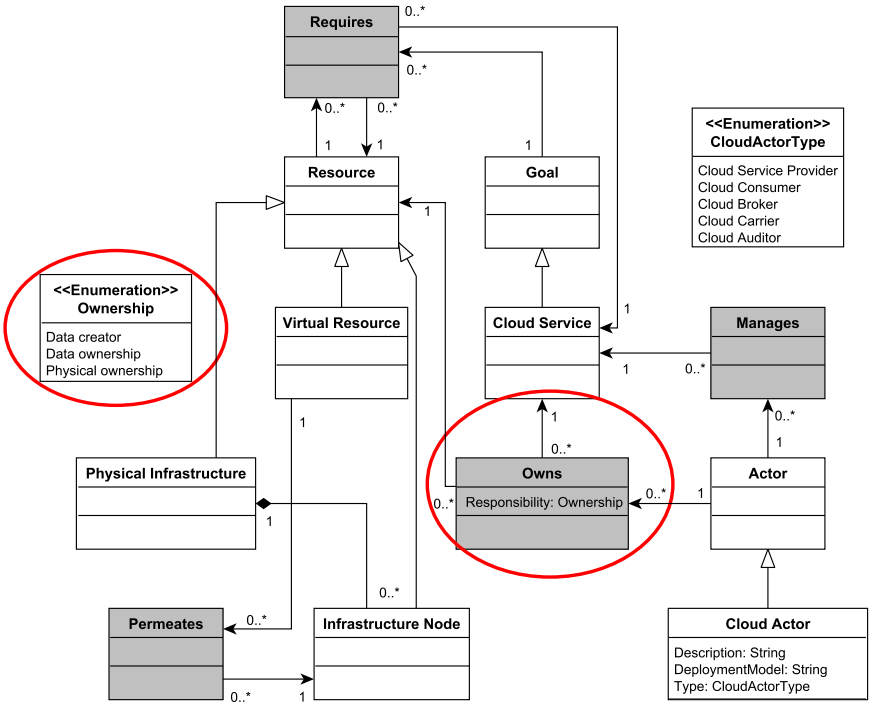

Figure 3.14: Highlighting the Owns relationship with extended properties and the enumeration Ownership.

Definition 3.4.3. Cloud Service The *Cloud service* concept has the properties *Capability*, *Security Property*, *Deployment Model* and *Service Model*, indicating the specific computing capability, security property, cloud service deployment model and the service model.

- Capability: String
- Security Property: «Enumeration» SecurityProperty [1..*]
- Deployment Model: «Enumeration» DeploymentModel
- Service Model: «Enumeration» ServiceModel

Figure 3.15 highlights the *Cloud Service* concept and its properties, in addition to the enumerations *DeploymentModel* and *ServiceModel*. The capability property is of type string, which provides a description of the strategic value or work the cloud service is capable of delivering. The *Security Property* property describes one

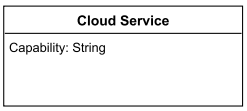

Figure 3.15: Highlighting the *Cloud Service* concept with extended properties and the enumerations *DeploymentModel* and *ServiceModel*.

or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* (Defined later in 3.21) consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repuditation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users. The *Deployment Model* property specifies which type of cloud service deployment model is deployed by the cloud service, where the enumeration *DeploymentModel* includes the values *Public*, *Private*, *Hybrid* and *Community*. The *Service Model* property specifies the service model of the cloud service, where the enumeration *ServiceModel* includes the values *SaaS*, *PaaS*, *IaaS* and *XaaS*.

Definition 3.4.4. Manages The *Manages* relation has the properties *Cloud Service*, *Deployment Model*, *Service Model* and *Manager*, indicating the instance of the cloud service managed by the actor, the deployment model and service model the actor is responsible for and the instance of the actor initiating the relationship.

- Cloud Service: Cloud Service

- Deployment Model: «Enumeration» DeploymentModel
- Service Model: «Enumeration» ServiceModel
- Manager: Actor

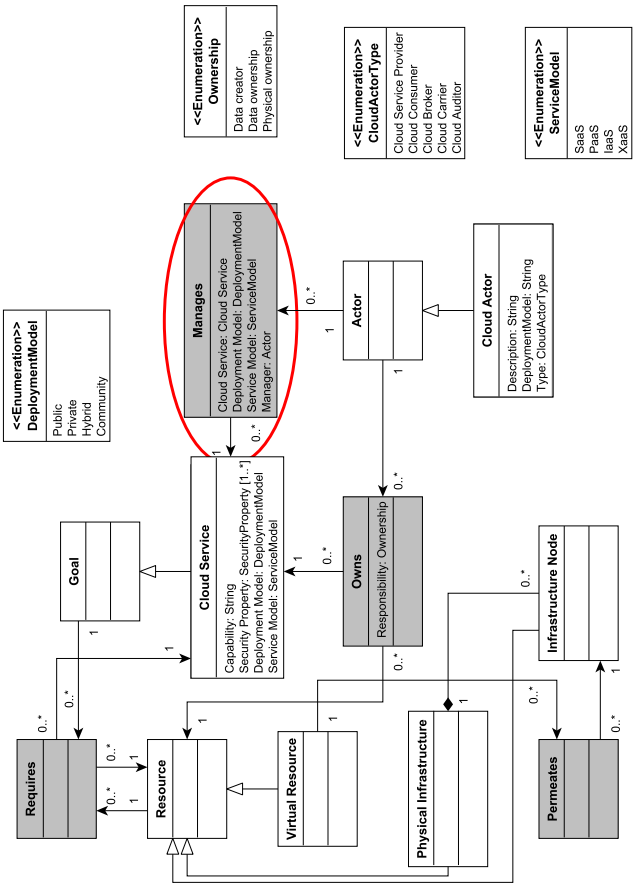

Figure 3.16: Highlighting the *Manages* relationship with extended properties.

Figure 3.16 highlights the *Manages* relationship and its properties. The *Cloud Service* property holds an instance of the *Cloud Service* concept, representing the target of the manage relationship. The *Deployment Model* property specifies which type of cloud deployment model is managed, where the enumeration *Deployment-Model* includes the values *Public*, *Private*, *Hybrid* and *Community*. The *Service Model* property specifies which level of the service model an actor manages, where

the enumeration *ServiceModel* includes the values *SaaS*, *PaaS*, *IaaS* and *XaaS*. The *Manager* property holds an instance of the *Actor* concept, representing the actor managing the cloud service.

Definition 3.4.5. Resource The *Resource* concept has the properties *Description*, *Value*, *Security Property*, *Product*, *Vendor* and *Version*.

- Description: String
- Value: Float
- Security Property: «Enumeration» SecurityProperty [1..*]
- *Product*: String
- *Vendor*: String
- *Version*: String

Figure 3.17 highlights the *Resource* concept and its properties. The *Description* property of type string provides a description of the resource. The *Value* property of type float represents a numerical value associated with user-defined metrics, which assists users in carrying out semi-automated analysis. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is added by the developer. The *Product*, *Vendor* and *Version* properties are values which is specified by the developer to refine the granularity of resources. Note that all properties of the resource concept are inherited by the specialised concepts of resource; the virtual resource, physical infrastructure and the infrastructure node.

Definition 3.4.6. Virtual Resource The *Virtual Resource* concept has the properties *Type* and *Visibility*.

- Type: «Enumeration» ResourceType
- Visibility: «Enumeration» Visibility

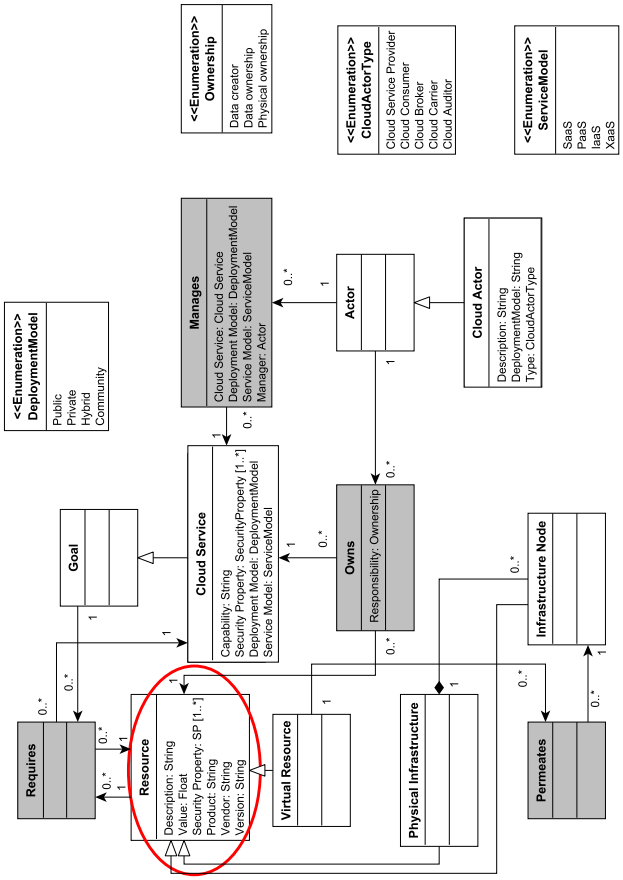

Figure 3.17: Highlighting the *Resource* concept with extended properties.

Figure 3.18 highlights the *Virtual Resource* concept and its properties. The *Type* property denotes the type of resource using the enumeration *ResourceType*, with values; *Data* and *Software*. The Visibility property denotes the level of visibility of a resource, using the enumeration *Visibility* with values; *Public*, *Private* and *Group*.

Definition 3.4.7. Physical Infrastructure The *Physical Infrastructure* concept has the properties *Jurisdiction*.

- Location: «Enumeration» Jurisdiction [1..*]

Figure 3.19 highlights the *Physical Infrastructure* concept and its properties. The *Location* property denotes one or more jurisdictional constraints on the physical in-

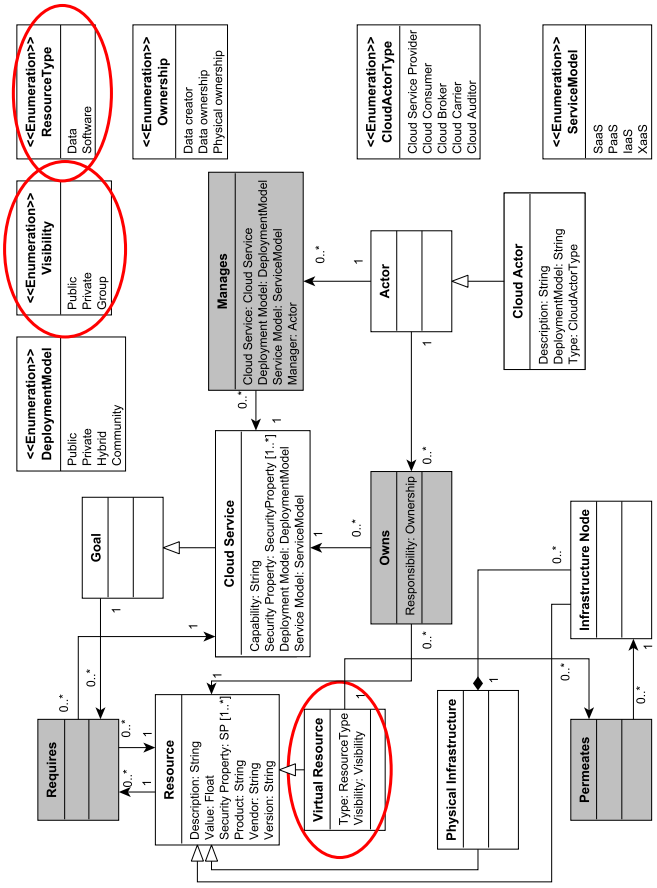

Figure 3.18: Highlighting the Virtual Resource concept with extended properties and the enumerations ResourceType and Visibility.

frastructure using values defined in the enumeration *Jurisdiction*, which consists of the values; *GDPD*. The values defined in the enumeration *Jurisdiction* is optionally defined by the user, where they are able to input custom entries. This represents which jurisdiction the asset falls under, and therefore has to adhere to. As an example, the developer may identify a process which should adhere to the jurisdictional requirements set in the EU. However given a physical infrastructure with the jurisdiction value of *"US"*, the developer is able to determine the conflict in the jurisdictional requirements in the between the concepts and propose an alternative approach.

Definition 3.4.8. Infrastructure Node The *Infrastructure Node* concept has the

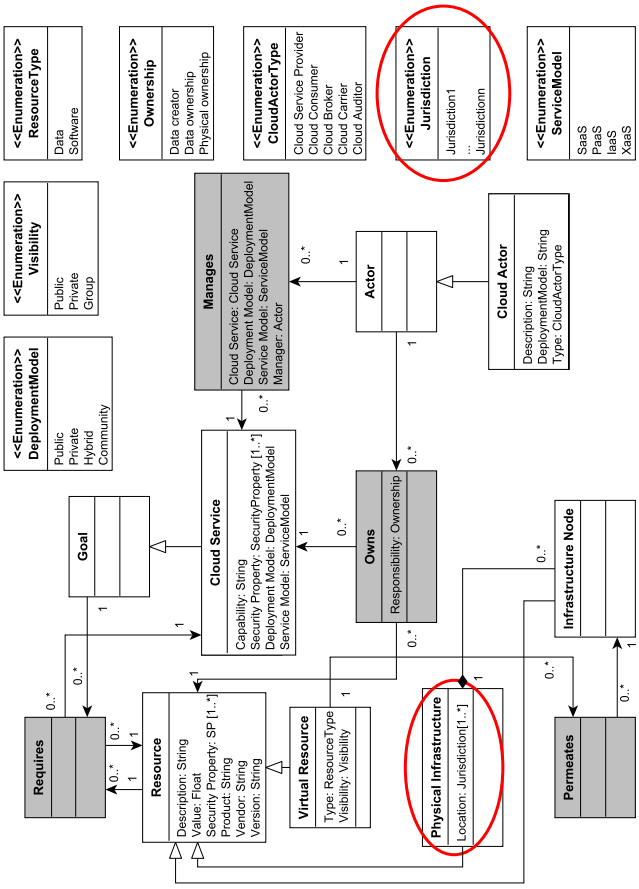

Figure 3.19: Highlighting the *Physical Infrastructure* concept with its extended property and the enumeration *Jurisdiction*.

properties *Type* and *Tenancy*.

- Type: «Enumeration» NodeType
- Tenancy: «Enumeration» Tenancy

Figure 3.20 highlights the *Infrastructure Node* concept and its properties. The *Type* property denotes the type of infrastructure node, which is defined through the enumeration *NodeType* using the values *Compute*, *Network* and *Storage*. The *Tenancy* property denotes the tenancy of the infrastructure node, which is enumerated through *Tenancy* with the values *Single* and *Multiple*.

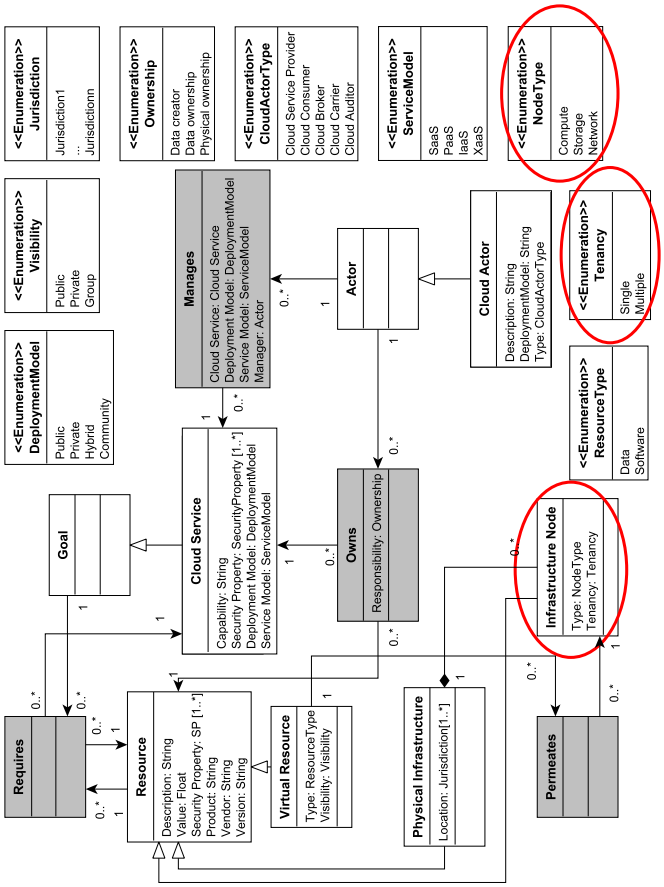

Figure 3.20: Highlighting the Infrastructure Node concept with extended properties and the enumerations NodeType and Tenancy.

Virtualisation is one of the cloud computing characteristics defined in the language. This characteristic is captured using the concept of virtual machines and hypervisors. Specifically a virtual machine is represented as an instance of a virtual resource. A hypervisor is represented using an instance of an infrastructure node. The relationship between an instance of a virtual machine and an instance of a hypervisor is captured through the permeates relationship, where an instance of a hypervisor is associated, through the permeates relationship, to one or more instances of a virtual machine. This relationship determines the tenancy of the hypervisor instance, which we now explain.

Multi-tenancy is another cloud computing characteristic defined in the language.

The concept of tenancy is captured through the *Tenancy* property of the infrastructure node concept, where the value *"Single"* defines a single-tenant instance and the value *"Multiple"* defines a multi-tenant instance. When there is one or less instances of a virtual resource connected to an infrastructure node through the *"Permeates"* relationship, the infrastructure node has the value *"Single"* in the *Tenancy* property. When there is two or more instances of a virtual resource connected to an infrastructure node through the *"Permeates"* relationship, the infrastructure node has the value *"Multiple"* in the *Tenancy* property.

The enumeration *Jurisdiction* consists of the following items: *"US", "UK", "EU" and "Asia"*. This represents which jurisdiction the asset falls under, and therefore has to adhere to. The enumeration *Tenancy* indicates whether a process is limited to a single tenant, through *"Single"*, or if two or more cloud users are involved, through *"Multiple"*.

Definition 3.4.9. Security Constraint The *Security Constraint* concept has the properties *Description* and *Security Property*.

- Description: String
- Security Property: «Enumeration» SecurityProperty [1..*]

Figure 3.21 highlights the *Security Constraint* concept and its properties. The *Description* property is of type string, which provides a description of the security constraint. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repuditation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users.

Definition 3.4.10. Requires The *Requires* relationship has the property *Filter Security Property*.

- Filter Security Property: «Enumeration» SPFilter

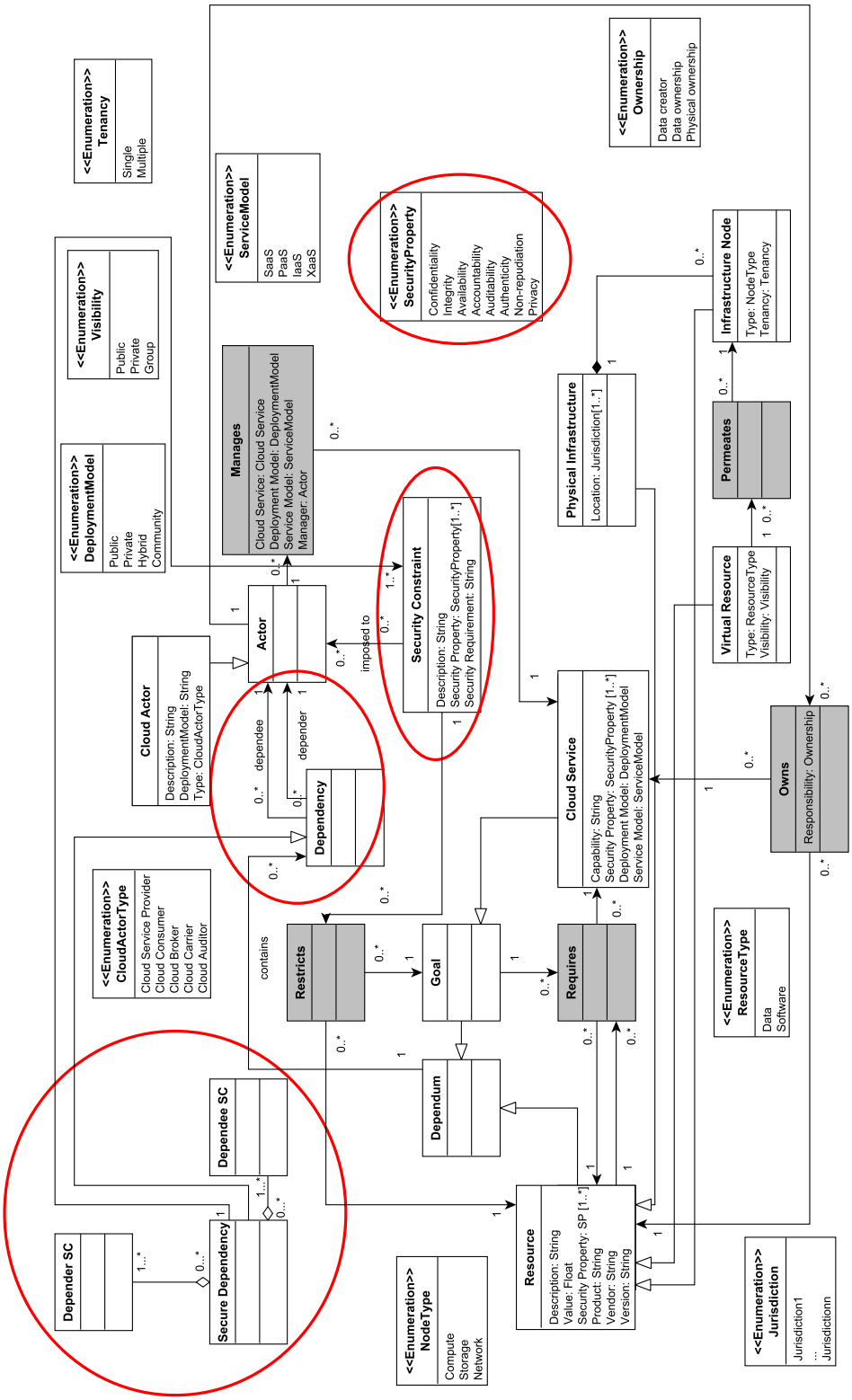

Figure 3.21: Highlighting the *Security Constraint* concept with extended properties and the enumeration *SecurityProperty*.

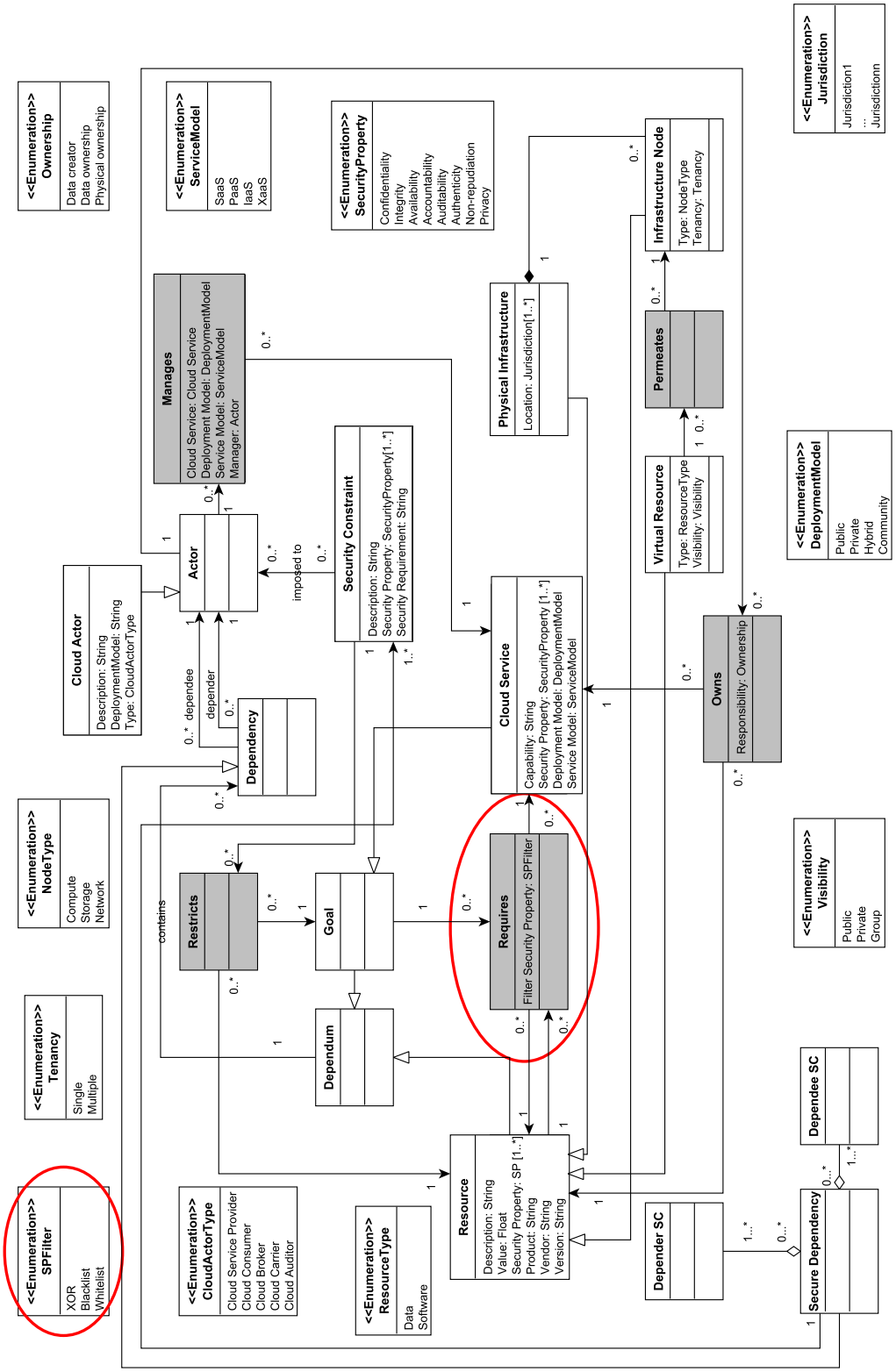

Figure 3.22: Highlighting the Requires relationship with its extended property.

Figure 3.22 highlights the *Requires* relationship and its property. The *Filter Security Property* property denotes the type of filter used to determine the security properties inherited in the *Requires* relationship. The purpose of this property is to allow the *Requires* relationship to associate security properties from the source concept to the target concept.

Definition 3.4.11. Threat The *Threat* concept has the properties *Description* and *Security Property*.

- Description: String
- Security Property: «Enumeration» SecurityProperty [1..*]

Figure 3.23 highlights the *Threat* concept and its properties. The *Description* property is of type string, which provides a description of the threat. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users.

Definition 3.4.12. Vulnerability The *Vulnerability* concept has the properties *Description*, *Attack Method*, *Vulnerability Metric* and *Security Property*.

- Description: String
- Attack Method: String [1..*]
- Vulnerability Metric: String
- Security Property: «Enumeration» SecurityProperty [1..*]

Figure 3.24 highlights the *Vulnerability* concept and its properties. The *Description* property of type string provides a description of the vulnerability. The *Attack Method* property describes one or more attack methods targeting an instance of the vulnerability, using the enumeration *AttackMethods* consisting of default values and optionally values defined by the user. The *Vulnerability Metric* property

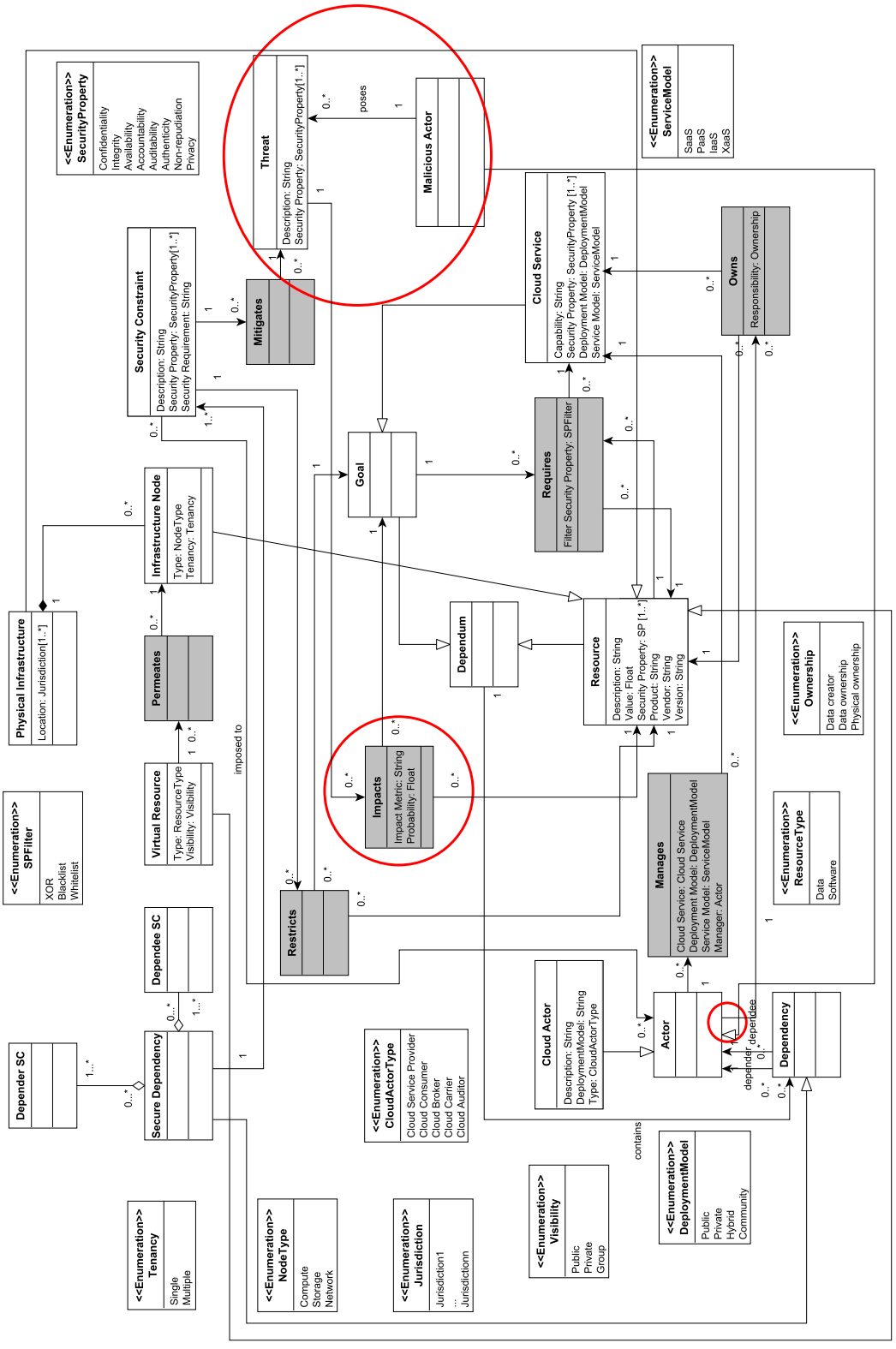

Figure 3.23: Highlighting the Threat concept with extended properties.

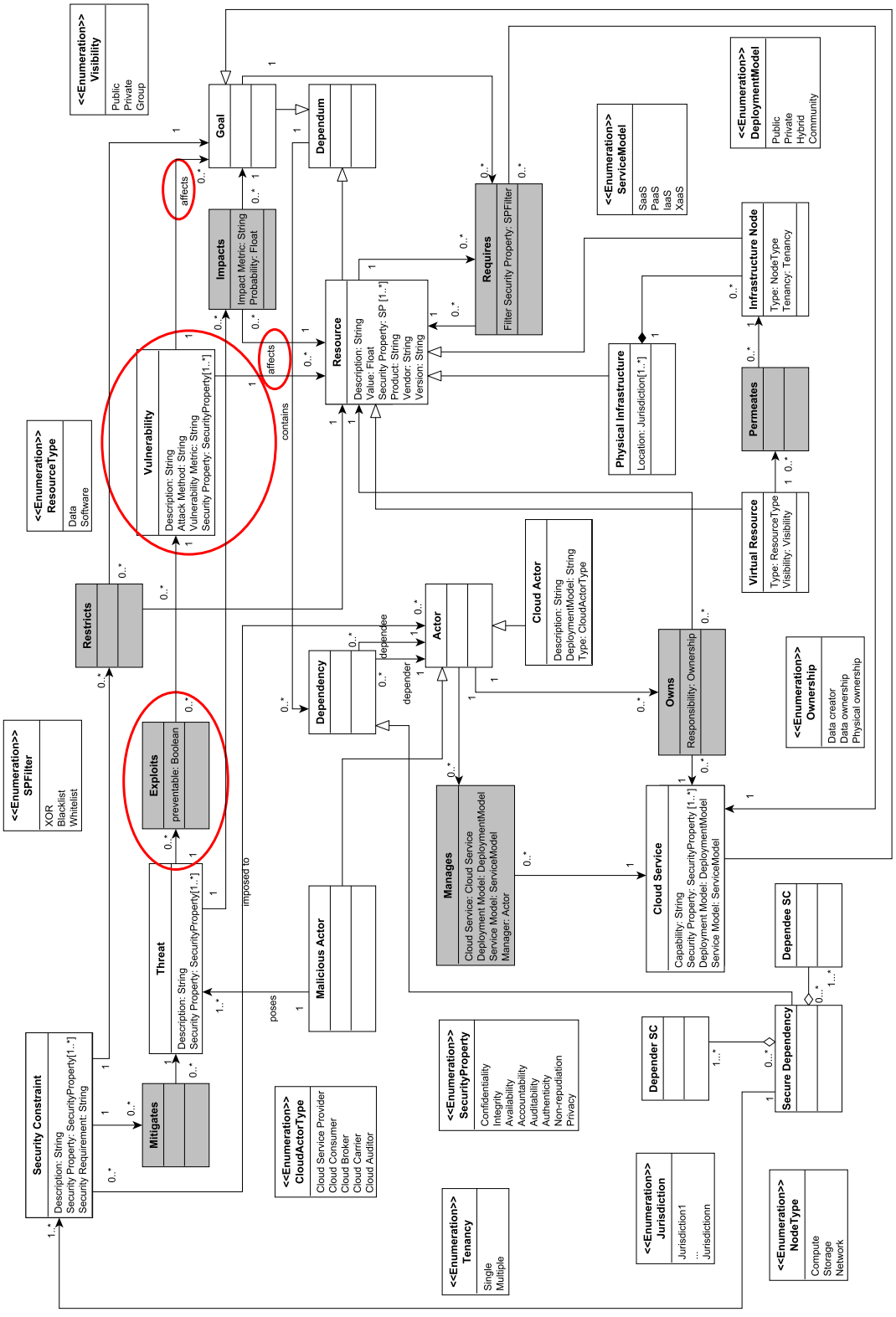

Figure 3.24: Highlighting the Vulnerability concept with extended properties.

of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users.

Definition 3.4.13. Security Mechanism The *Security Mechanism* concept has the properties *Description*, *Mechanism Metric* and *Security Property*.

- Description: String
- Security Property: «Enumeration» SecurityProperty [1..*]
- Metric: String

Figure 3.25 highlights the *Security Mechanism* concept and its properties. The *Description* property of type string provides a description of the security mechanism. The *Metric* of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users. The *Metric* property of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis.

Definition 3.4.14. Security Objective The *Security Objective* concept has the properties *Description*, *Security Property* and *Objective Metric*.

- Description: String
- Security Property: «Enumeration» SecurityProperty [1..*]
- Objective Metric: String

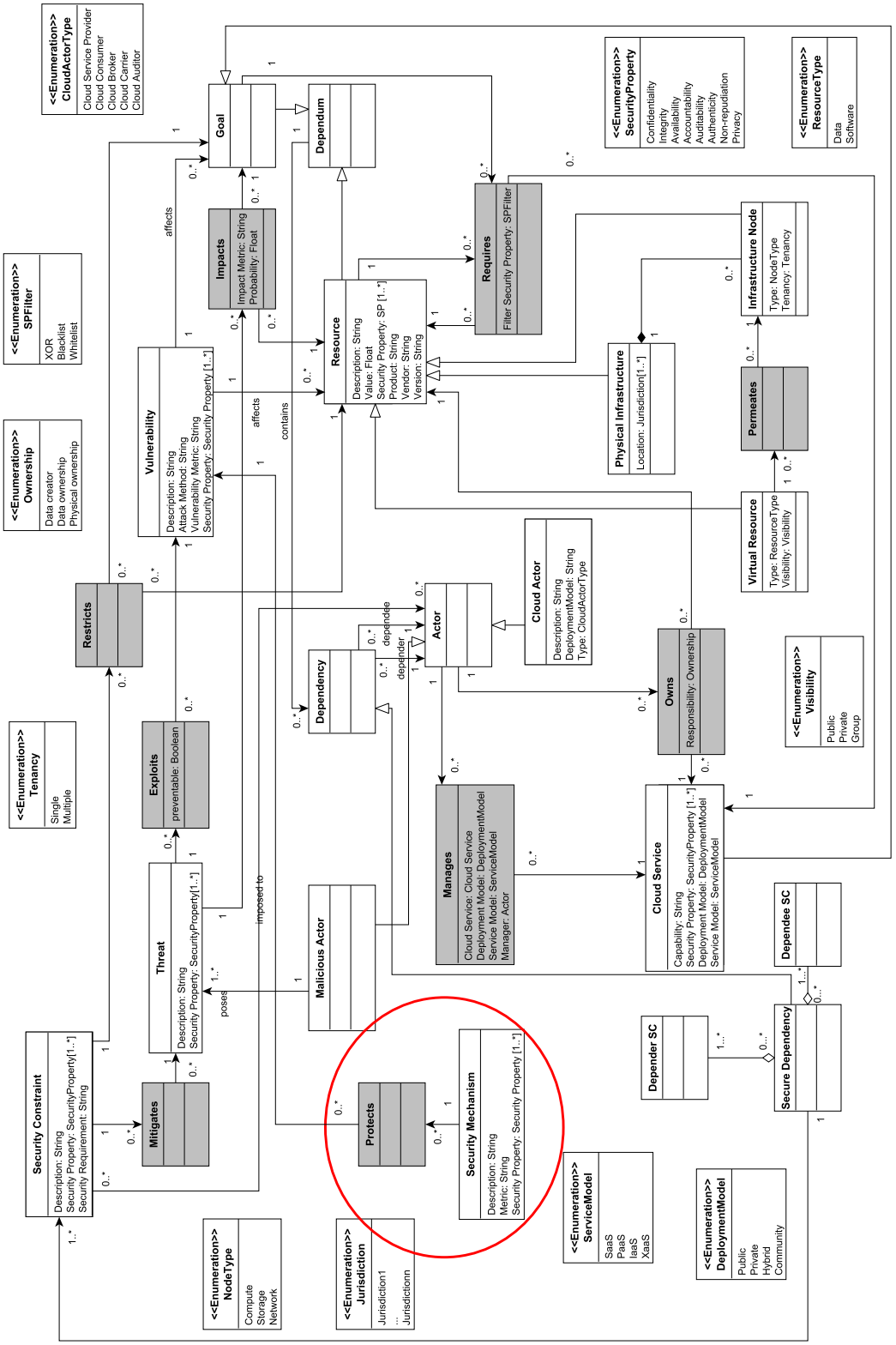

Figure 3.25: Highlighting the Security Mechanism concept with extended properties.

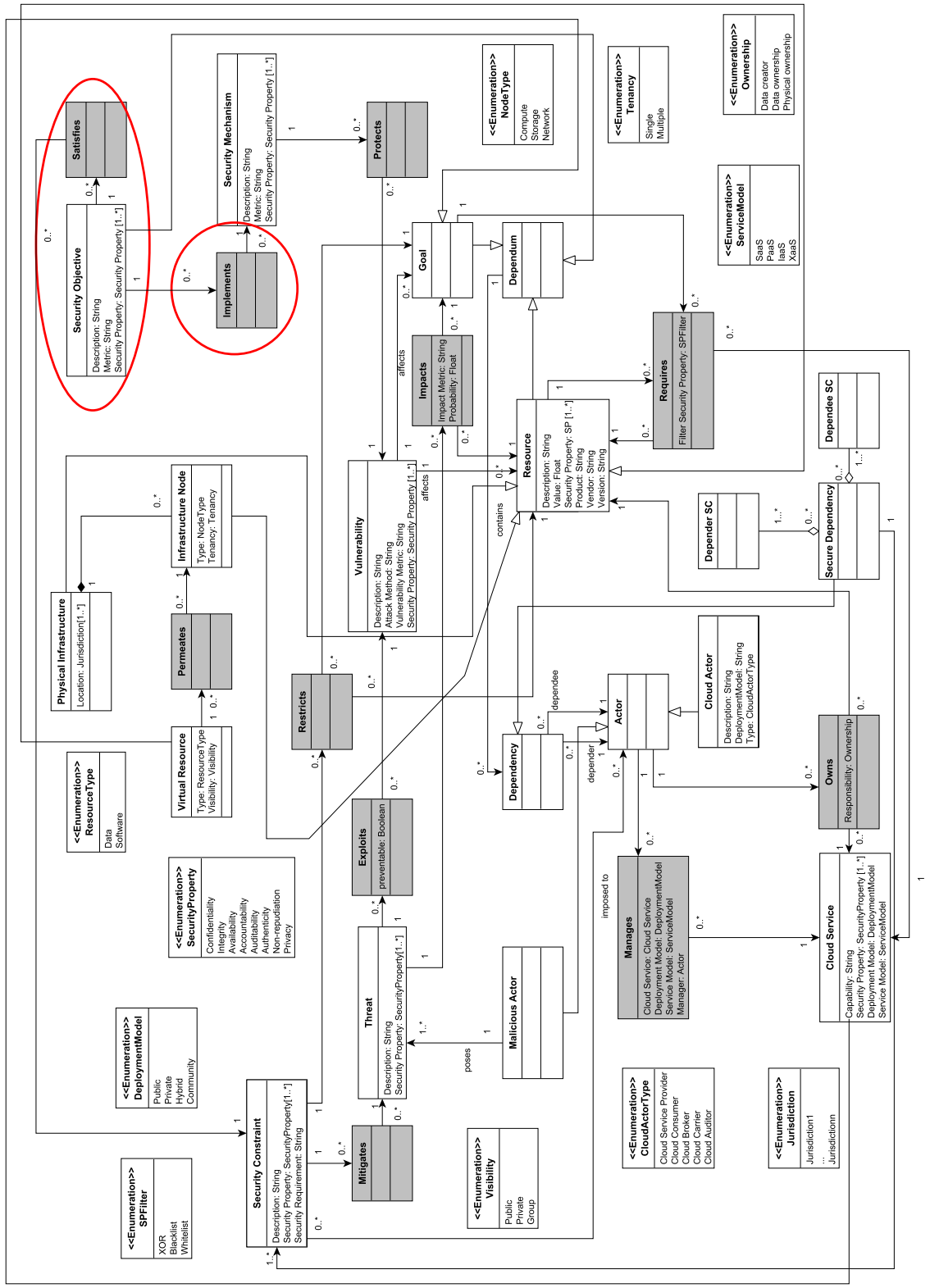

Figure 3.26: Highlighting the Security Objective concept with extended properties.

Figure 3.26 highlights the *Security Objective* concept and its properties. The *Description* property of type string provides a description of the security objective. The *Security Property* property describes one or more security properties of the security constraint, which is selected from the enumeration *SecurityProperty* consisting of; *Confidentiality*, *Integrity*, *Availability*, *Accountability*, *Auditability*, *Authenticity*, *Non-repudiation* and *Privacy*. The *SecurityProperty* enumeration also accepts custom values which is optionally added by users. The *Metric* property of type string allows the user to define a numerical value associated with user-defined metrics, which assists them in carrying out semi-automated analysis.

3.5 Syntax

In this section we present the concrete syntax of the modelling language. The concrete syntax is visualised using graphical notation, where each concept in the modelling language is mapped to an unique graphical notation. The shapes of the graphical notation was chosen arbitrarily in order to distinguish between the original Secure Tropos notation and our novel cloud computing concepts. The graphical notation is rendered using the SectroCloud tool, which is described later in Section 5.4. Note that the concept attribute shown in the figures with the properties of the concrete syntax is automatically generated by the SectroCloud tool, therefore it is not part of the model.

We also describe the instance syntax of the modelling language. The instance syntax is a textual encoding of the concrete syntax which provides an one-to-one mapping of a concept from our cloud metamodel to an instance of a concept. The purpose of the instance syntax is to provide a formal representation of concept instances, in a machine readable format in order to perform analysis on cloud models through tool-support. Thus the instance syntax allows the unambiguous encoding of concepts in a textual format, which describes the instantiated concepts from a cloud model to facilitate security analysis. For the purpose of clarity and presentation in this section, properties without values are not full described. As an example, instead of writing *INSTANCE(A,B,C,D)* as *INSTANCE(„,test)* where the only non-empty property is *D* with the value *"test"*, we write *INSTANCE(test)*.

Definition 3.5.1. Cloud Service The instance syntax of a cloud service is *CS(D,CAP,SP,DM,SM)*. The instance syntax for a cloud service describes the following: *CS()* describes an instance of a cloud service with associated concepts encapsulated inside the parenthesis, *D* provides a description of the cloud service, *CAP* describes the capability of the cloud service, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*, *DM* is the deployment model selected from a list of enumerated values and *SM* is the service model selected from a list of enumerated values.

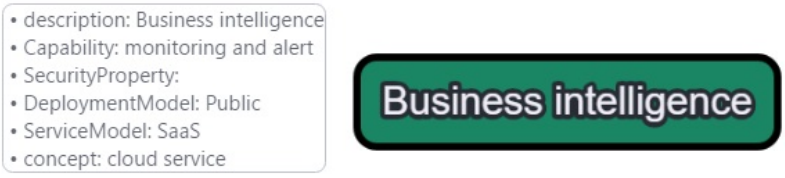

Figure 3.27: Concrete syntax of a cloud service.

Figure 3.27 shows the concrete syntax of a cloud service, which is represented as a light green rectangle with a solid black outline. The textual description inside the rectangle on the left denotes the properties of the cloud service instance. The instance syntax in this case is CS(Business_intelligence, monitoring_and_alert,Public,SaaS)

Definition 3.5.2. Cloud Actor The instance syntax of a cloud actor is *CA(D,{T})*.

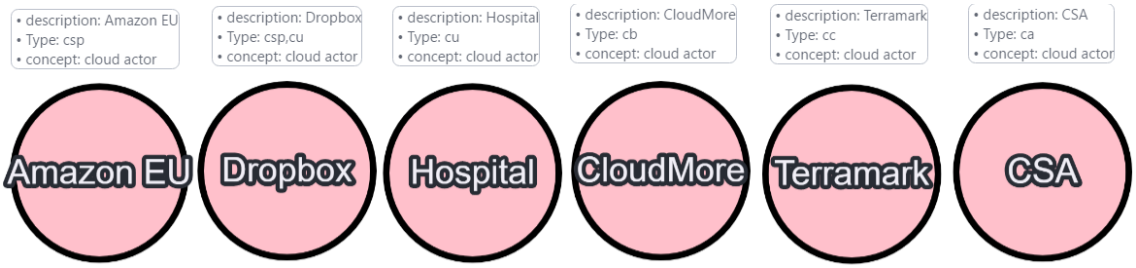

Figure 3.28: A Cloud Actor is visualised as a light pink circle.

The instance syntax for a cloud actor describes the following: *D* is a description of the cloud service provider, {*T*} is a set denoting one or more roles played by a

cloud actor. The type of role played by a cloud actor is selected from the enumeration *CloudActorType* with the following list of roles; Cloud Service provider(*csp*), Cloud Consumer(*cu*), Cloud Broker(*cb*), Cloud Carrier(*cc*), and Cloud Auditor(*ca*). For example a cloud service provider with the description *hospital* is encoded as *CA(hospital,{csp})*. In case of cloud actors playing multiple roles, for example *dropbox* who is a cloud service provider and also a cloud consumer is encoded as *CA(dropbox,{csp,cu})*.

Figure 3.28 shows the concrete syntax of a cloud actor, which is represented as a light pink circle. The text in the rectangle on the left denotes the properties of the cloud actor. In Figure 3.28 a range of cloud actor types are shown, where the properties of each instance is located above the light pink circle for the purpose of demonstration. Note that the *Dropbox* cloud actor has both the cloud service provider and a cloud consumer types, as indicated by the {*csp,cu*} textual description. This means the *Dropbox* cloud actor is responsible for the role of a cloud service provider, while also playing the part of a cloud consumer.

Definition 3.5.3. Virtual Resource The instance syntax of a virtual resource is *VR(D,VAL,SP,T,VEN,VER,RT,VIS)*. The instance syntax for a virtual resource describes the following: *D* is the description of the virtual resource instance, *VAL* denotes the value of the asset which can be used to calculate metrics such as mitigation costs or determining impact on assets, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*, *T* specifies the technical type of the instance, *VEN* describes the organisational vendor of the instance, *VER* is the version number of the instance, *RT* is the resource type selected from a list of enumerated values, *VIS* is the visibility type selected from a list of enumerated values. For instance the virtual resource *Patient data* with the description *Patient data*, resource type *data* and visibility type *private* is encoded as *VR(Patient_data,data,private)*.

Figure 3.29 shows the concrete syntax of a virtual resource, which is represented as a yellow rectangle with a dashed yellow outline. The textual description inside the rectangle on the left denotes the properties of the virtual resource instance.

Definition 3.5.4. Physical Infrastructure The instance syntax of a physical infrastructure is *PI(D,VAL,SP,T,VEN,VER,L)*. The instance syntax for a phys-

Figure 3.29: A virtual resource is visualised as a yellow rectangle with thin dashed outlines.

ical infrastructure describes the following: *D* is the description of the physical infrastructure, *VAL* denotes the value of the asset which can be used to calculate metrics such as mitigation costs or determining impact on assets, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*, *T* specifies the technical type of the instance, *VEN* describes the organisational vendor of the instance, *VER* is the version number of the instance, *L* is an enumerated list of jurisdictions which the physical infrastructure falls under. For example the physical infrastructure *hospital information system* with the description *hospital information system* and jurisdiction *General Data Protection Regulation* (GDPR) is encoded as *PI(hospital_information_system,GDPR)*.

Figure 3.30: A physical infrastructure visualised as a yellow rectangle with thick outlines.

Figure 3.30 shows the concrete syntax of a physical infrastructure, which is represented as a yellow rectangle with a solid black outline. The textual description inside the rectangle on the left denotes the properties of the physical infrastructure instance.

Definition 3.5.5. Infrastructure Node The instance syntax of a cloud service is *IN(D,VAL,SP,T,VEN,VER,NT,TE)*. The instance syntax for an infrastructure node describes the following: *D* is the description of the infrastructure node, *VAL* denotes the value of the asset which can be used to calculate metrics such as mitigation costs or determining impact on assets, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*, *T* specifies the technical type of the instance, *VEN* describes the organisational vendor of the instance, *VER* is the version number of the instance, *NT* determines the type of the infrastructure node from a list of enumerated values, *TE* determines the type of tenancy from a list of enumerated values. As an example the infrastructure node *amazon server 1* with the *compute* node type and *single* tenancy is encoded as IN(amazon_server_1,compute,single).

Figure 3.31: An infrastructure node visualised as a yellow rectangle with thick dashed outlines.

Figure 3.31 shows the concrete syntax of an infrastructure node, which is represented as an yellow cylinder with a black outline. The text inside the rectangle on the left denotes the description of the infrastructure node.

Definition 3.5.6. Security Constraint The instance syntax of a security constraint from is *SC(D,SP,SR)*. The instance syntax for a security constraint describes the following: *D* is the description of the instance, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*, *SR* determines the security requirements of the instance which is entered by the developer or through security analysis.

Figure 3.32 shows the concrete syntax of a security constraint, which is represented as a red octagon with a black outline. The text inside the rectangle on the

Figure 3.32: A security constraint visualised as a red octagon with a black outline.

left denotes the description of the security constraint node.

Definition 3.5.7. Vulnerability The instance syntax of a vulnerability is: *V(D,AT,VM,SP)*. The instance syntax for a vulnerability describes the following: D is the description of the instance, AT represents zero or more attack methods for exploiting the instance, VM denotes a value which can be used to calculate metrics such as mitigation costs or determining impact on assets, SP determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*.

Figure 3.33: A vulnerability visualised as a red rectangle with a black outline.

Figure 3.33 shows the concrete syntax of a vulnerability, which is represented as an red rectangle with a black outline. The text inside the rectangle on the left denotes the description of the vulnerability.

Definition 3.5.8. Threat The instance syntax of a threat is *T(D,SP)*. The instance syntax for a threat describes the following: D is the description of the instance, SP determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*.

Figure 3.34: A threat visualised as a light red pentagon with a black outline.

Figure 3.34 shows the concrete syntax of an threat, which is represented as a light red pentagon with a black outline. The text inside the rectangle on the left denotes the description of the threat.

Definition 3.5.9. Security Mechanism The instance syntax of a security mechanism is *SM(D,MM,SP)*. The instance syntax for a security mechanism describes the following: *D* is the description of the instance, *MM* denotes a value which can be used to calculate metrics such as mitigation costs or determining effectiveness of counter-measures, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*.

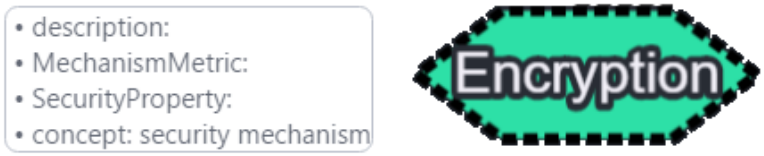

Figure 3.35: An security mechanism visualised as a green hexagon with black dashed outlines.

Figure 3.35 shows the concrete syntax of a security mechanism, which is represented as an green hexagon with black dashed outlines. The textual description inside the hexagon denotes the description of the security mechanism.

Definition 3.5.10. Security Objective The instance syntax of a security objective is *SO(D,OM,SP)*. The instance syntax for a security objective describes the following: *D* is the description of the instance, *OM* denotes a value which can be

used to calculate metrics such as mitigation costs or determining the priority of objectives, *SP* determines the security properties of the instance which is selected from zero or more security properties in the list of enumerated values *SecurityProperty*.

Figure 3.36: An security objective visualised as an aqua hexagon with a black outline.

Figure 3.36 shows the concrete syntax of a security objective, which is represented as an aqua hexagon with a black outline. The text inside the rectangle on the left denotes the description of the security objective.

3.6 Cloud Computing Models

Models are abstractions of a system, where we capture the elicited requirements and specifications of components associated with the system. Models are constructed using a modelling language, at various levels of granularity dependent on the development stage. For example at the early requirements stage, models capture the stakeholder requirements using goals and components such as resources and dependencies. At the late requirements stage the goals are decomposed to refine dependencies and determine responsible parties. Additional properties are captured as the developer progresses through the various stages of development. For example cloud properties such as multi-tenancy, single-tenancy and security properties are added to existing concepts.

So far in this chapter we have described the concepts, relationships, properties and concrete syntax of the cloud modelling language to address the first half of *RQ1*. To recap, *RQ1* is: *"How do we describe cloud computing concepts to capture cloud systems from a security requirements engineering perspective?"*. We now describe how we capture the information in cloud computing systems through models, based on the modelling language we have defined. Specifically we describe three

views derived from the cloud environment model, focusing on the different level of abstraction offered in each view.

3.6.1 Organisational Cloud View

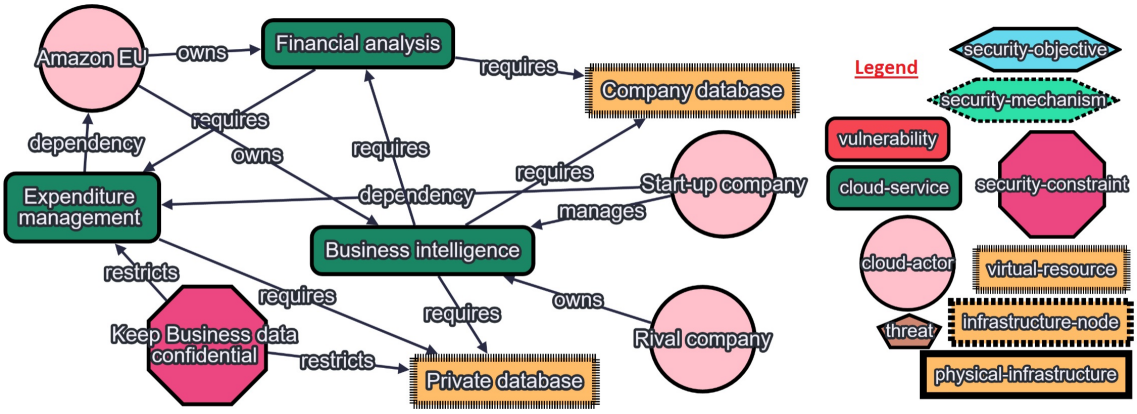

Figure 3.37: The organisation cloud view showing cloud actors, cloud services and virtual resources.

We begin our approach for capturing security requirements in cloud computing systems by modelling cloud services based on stakeholder needs. We make the assumption that the developer has completed the requirements elicitation process and has the requirements of the stakeholders at hand. Thus the developer begins the modelling process from the organisational cloud view, capturing the stakeholders, goals, security needs and resources. More specifically the organisational cloud view consists of the following concepts and relationships: *Goal*, *cloud service*, *actor*, *cloud actor*, *virtual resource*, *security constraint*, *restricts*, *has*, *requires*, *dependency*. Figure 3.37 illustrates an example of the organisational cloud view using the graphical notation of the cloud modelling language.

Stakeholder Needs

This conceptual layer captures organisational needs from a social perspective through properties such as roles played by actors, delegation of responsibilities and relationship resolution. The organisational view allows the developer to model stakeholders, the roles played and their relationships to required components. Thus the model

guides the developer through the process of identifying direct and indirect stakeholders as actors or cloud actors with roles determined through their interaction with cloud services and components. Each type of role plays a part when capturing the security needs of stakeholders, for example a cloud service provider will have different responsibilities and relationships compared to a cloud consumer.

Stakeholder needs are modelled using a cloud actor as the depender, a cloud actor as the dependee and a cloud service as the dependum. This is the *dependency* relationship where the depender is a cloud stakeholder that depends on another cloud stakeholder to deliver a cloud service in order to satisfy their needs. System component needs are modelled from a resource or goal, to a resource or cloud service. This is the *requires* relationship which indicates that a system component requires a specific resource or cloud service. For example a database may depend on storage to backup data, or a cloud service may depend on other cloud services to deliver specific functions. Security constraints capture the security needs of stakeholders. In the organisational view security constraints are placed by stakeholders on their needs, which corresponds to goals of cloud actors. Thus a security constraint is placed onto a cloud service from an actor to another actor in indicate the satisfaction of the stakeholder needs by the latter actor.

3.6.2 Application Cloud View

The application cloud view captures the software-oriented concepts of the cloud computing paradigm, realised through fine-grained descriptions of cloud services. In this view the developer refines the virtual resources, cloud actor relationships and decomposes cloud services. More specifically the application cloud view consists of the following concepts and relationships: *Cloud service*, *actor*, *cloud actor*, *virtual resource*, *security constraint*, *threat*, *vulnerability*, *restricts*, *has*, *requires*, *dependency*, *impacts* and *affects*. Figure 3.38 shows an example of the application cloud view, building on the cloud model refined in the organisational cloud view.

This layer represents the abstract concepts for software and applications in the system-under-design, centring around cloud services, components interacting with cloud services and the security impacts. In our running example we model two cloud services, the security issues impacting them, the virtual resources they require

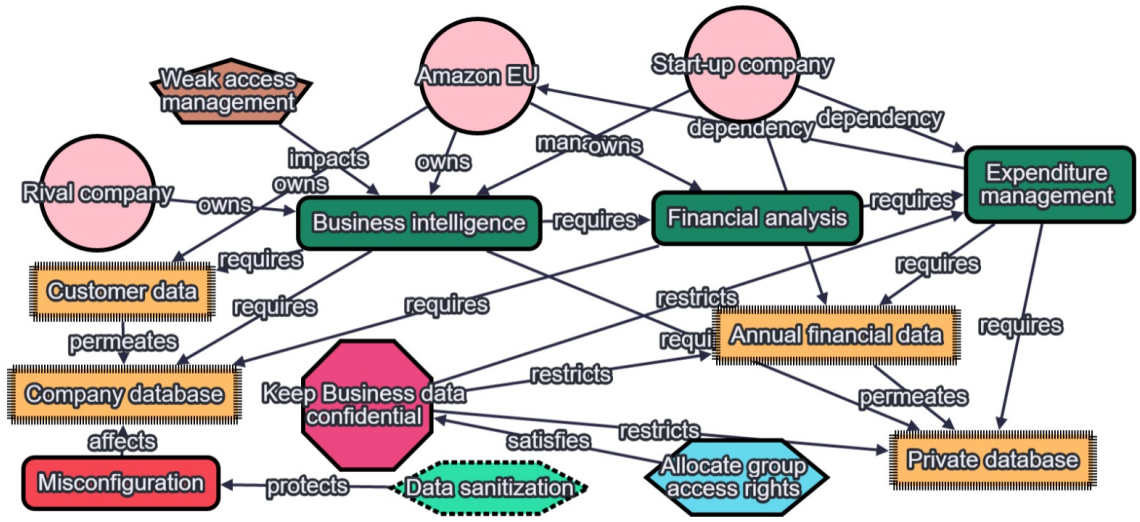

Figure 3.38: The application view showing refinement of virtual resources, cloud actor relationships and security concepts.

and partial solutions for mitigation. The service and deployment models of each cloud service determines the actors that owns the cloud service, actors responsible for managing the cloud service, security issues and propagation of dependencies.

3.6.3 Infrastructure Cloud View

This view describes the hardware-oriented concepts supporting cloud services, where the goal is to capture tangible cloud computing components. Thus the developer describes assets such as computing servers, storage devices and networks, where properties such as geographical location and managing actors are identified in each instance. More specifically the infrastructure cloud view consists of the following concepts and relationships: *Cloud service*, *actor*, *cloud actor*, *virtual resource*, *physical infrastructure*, *infrastructure node*, *security constraint*, *restricts*, *has*, *requires*, *dependency*, *impacts*, *affects*, *threat*, *vulnerability*, *security objective* and *security mechanism*. Figure 3.39 shows an infrastructure cloud view with refinements to the infrastructure components and additional properties, such as multi-tenancy in infrastructure nodes.

We define this layer to abstractly model physical components required to realise cloud computing services, which we capture as infrastructure nodes belonging to one

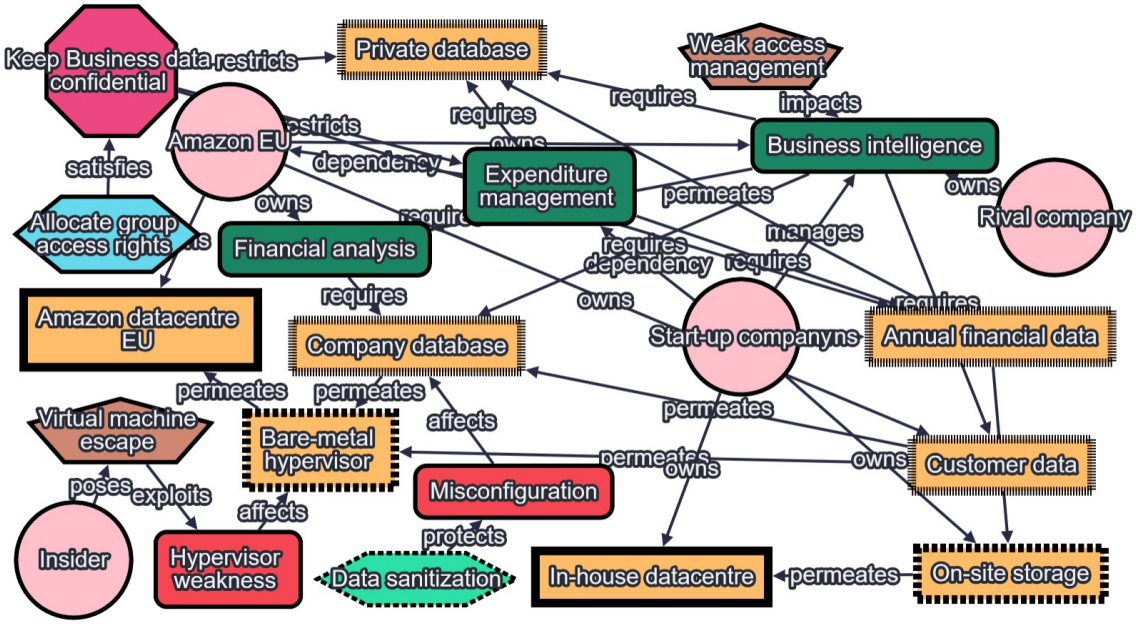

Figure 3.39: The infrastructure view showing the refinement of infrastructure nodes, cloud actor relationships and security concepts.

or more physical infrastructure containers representing IT infrastructure.

3.6.4 Cloud Environment Model

Existing approaches in security requirements engineering and cloud security capture system and stakeholders needs through disparate models, where each model describes properties unique to self-contained domains. However by adapting a discrete and isolated approach to elicit requirements, critical properties are omitted during the process. The cloud environment model is thus refined by the developer throughout the three views to provider finer granularity to the cloud system. By providing the required level of detail, specific analysis can be carried out to support the developer when performing reasoning on the security requirements of the cloud system.

3.7 Chapter summary

This chapter presents the cloud security modelling language, which extends the secure tropos methodology Mouratidis and Giorgini, 2007 with cloud computing concepts, relationships and properties to describe security requirements in cloud computing environments. The concepts used to capture cloud computing systems are defined as the cloud service, virtual resource, infrastructure node and physical infrastructure. Interactions between the proposed cloud computing concepts and security concepts are described using relationships, namely through the concept of ownership, management and permeate. Properties are enumerated in the language to facilitate analysis techniques, describing in detail the configurations and specifications of an instance. As an example an instance of the cloud service concept has the properties service level and deployment model, which specifies the category security issues affecting the instance. A concrete syntax is defined using graphical notation, mapping instances of concepts to visual representations, allowing visual modelling of cloud computing systems. An instance syntax of the language is then defined, where the information of an instance is encoded textually and this is machine-readable. The instance syntax facilitates security analysis, in order to perform algorithmic functions such as pattering matching for known vulnerabilities on cloud models encoded in the machine-readable instance syntax. Finally we describe how the cloud environment model is constructed using the concepts, relationships and properties defined in this chapter.

Chapter 4

Secure Cloud Process

This chapter outlines our systematic approach in order to model the alignment of organisational needs with security concepts in the context of cloud computing systems. This approach is intended to guide developers through the process of capturing cloud computing systems through modelling techniques, defining stakeholder requirements and security needs. Each step of the process procedurally constructs a holistic view of the system-under-design, represented through models comprising system needs and cloud-specific security properties from a security requirements engineering perspective. The approach outlines the types of analysis supported and the prerequisites the developer should follow in order to perform semi-automated reasoning. We also indicate where input from security requirements engineers or cloud computing experts are required.

4.1 Overview of the Secure Cloud Process

We describe an iterative process which supports developers to systematically capture and refine cloud computing relationships, security properties and organisational needs. Each activity defines a step which contributes towards defining and constructing a cloud environment model representing the system-under-design. An overview of the Secure Cloud Process is shown in Figure 4.1 with the following activities:

- **Organisational goal model:** identify organisational needs, stakeholders, assets and relationships, producing an organisational goal model as output.

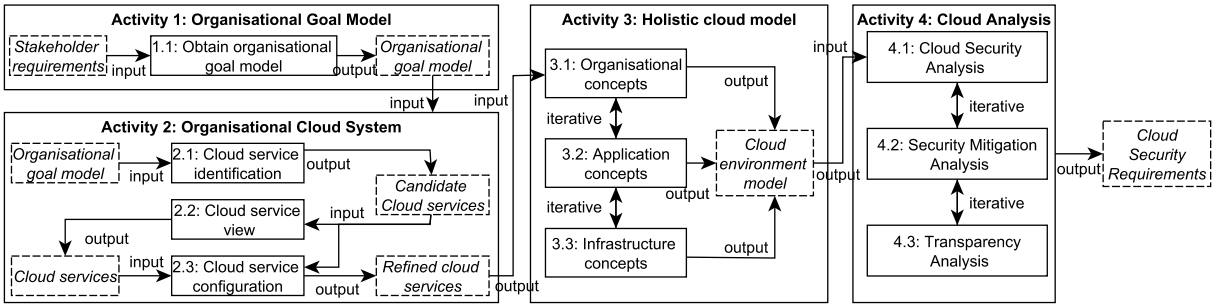

Figure 4.1: An overview of the Secure Cloud Process.

- **Organisational cloud system:** identify and configure cloud services, generating a list of cloud services as output.
- **Holistic cloud model:** refinement of layer-specific concepts through a fine-grained approach, focusing on the organisational, application and infrastructure levels of the system-under-design to output a cloud environment model.
- **Cloud analysis:** performing analysis techniques from the cloud security analysis, security mitigation analysis and transparency analysis to help developers identify and understand the cloud security requirements of the system-under-design. .

4.2 Activity 1: Organisational Goal Model

This is the first activity of the secure cloud process, where the input is either an existing goal model, or constructed by the developer based on existing requirements. In the latter case, we assume that the process of eliciting system needs, analysis and production of requirements has been carried by security requirements engineers and the developer has access to the requirements of the system-under-design. This assumption is made on the basis that the process for producing goal models of existing software systems or from initial requirements falls out of the scope of this thesis. Due to the maturity of established work in requirements engineering, we do not attempt to redefine the existing process of goal modelling. Instead we focus on extending the existing work to capture security requirements issues in cloud comput-

ing, building upon goals models to represent these concepts. In this case the SecTro modelling tool is used to create organisational models from existing requirements, because the Secure Tropos methodology provides a goal-oriented approach in area of security requirements engineering. The organisation goal model includes the following concepts; actor, resource, goal and security constraint. Thus the organisation goal model would be the output of this step, which can be generated using existing approaches such as Secure Tropos.

4.3 Activity 2: Organisational Cloud System

The goal of the second activity of the secure cloud process is to identify a list of cloud services based on the requirements of the system-under-design. We take as input an organisational goal model from Activity 1 and produce, as output, a list of cloud services. This activity adds the following concepts in order to refine the organisation goal model from the previous activity; cloud service, cloud actor and virtual resource.

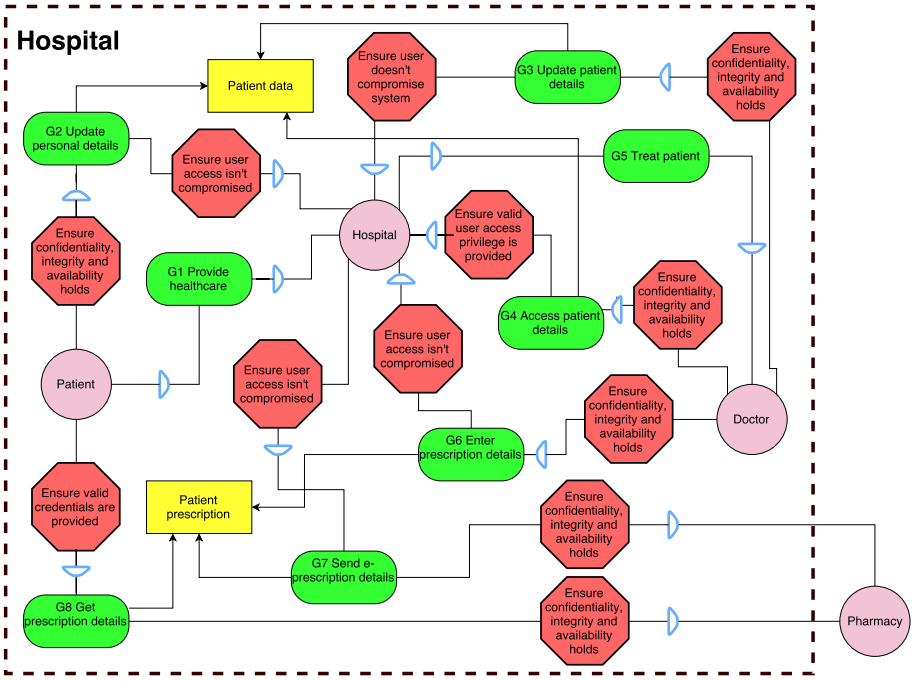

Figure 4.2: Simple organisational goal model of hospital processes.

In order to demonstrate the steps involved in this activity, we now refer to the health-care running example presented in Chapter 1, where a goal model is produced using existing security requirements engineering approaches, as shown in Figure 4.2. The goal model describes a scenario where one hospital wishes to partially offload their patient records management system to the cloud, in order to facilitate health information exchange through the availability and interoperability of patient records. We now demonstrate how to identify and specify cloud services.

Step 2.1: Cloud Service Identification

We begin this step by taking the organisation goal model produced in the previous step as input, identifying and selecting from the organisational goals a list of cloud services and generating as output the list of cloud services identified. By default the cloud service selection criteria is defined such that each goal corresponds to a cloud service in an one-to-one mapping. However the developer is able to define their own selection criteria, such as identifying their own cloud services, creating compositions consisting of one or more cloud services and modifying attributes based on their needs.

Cloud Service Name	Goal
Capability	Capability
Actors	
	Actors
Resource	
	Resource
Relationships	
Dependency	A depends on C to provide a service B.
End-user	A uses service B provided by C.
Requires	A requires resource (implicitly provided by B).
Security Constraint	A places a security constraint on B.
Service Model	{SaaS || PaaS || IaaS}
Deployment Model	{Public || Private || Hybrid || Community}

Figure 4.3: The Cloud Service Template.

We use the organisational goal model as input and apply the Cloud Service Template shown in Figure 4.3 to identify and generate a list of candidate cloud services. The purpose of the Cloud Service Template is to provide a set of guidelines assisting developers through the process of identifying cloud services. Practically the Cloud Service Template is used to map each selected goal in the goal model into a candidate cloud service, extracting the information required to describe a cloud

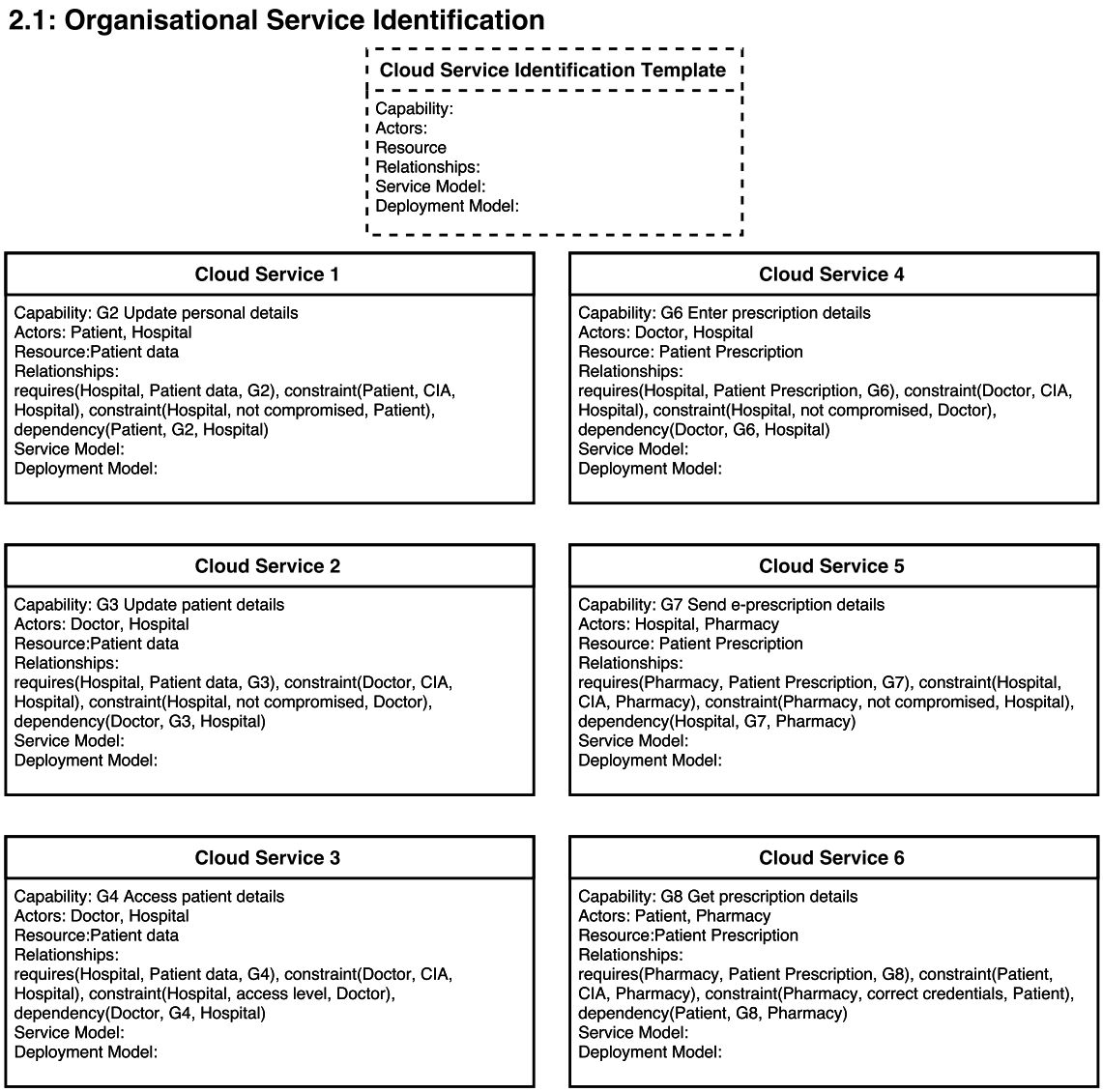

Figure 4.4: Default fields in a cloud service with a list of candidate cloud services.

service based on our definition. An example of a candidate list of cloud services is shown in Figure 4.4. Practically the developer will use pen and paper to produce a list of candidate cloud services, before creating a model using tool-support. Fields which require user input will be left blank at this step i.e. the service and deployment models. We use the term candidate to describe the extracted cloud services due to the fact that at this stage in the activity, essential attributes such as service and deployment models which abstractly determine how cloud services are instantiated have yet been finalised, in addition to logical aggregation of similar activities such as self-contained capabilities which contribute towards a common primary function or goal. The output of this sub-step is a list of candidate cloud services, in this case a list of cloud services that have yet been fully instantiated. The user then has a choice of either visualising the selected cloud services in Step 2.2 through the organisational cloud service view, or to specify and configure their chosen cloud services in Step 2.3.

Step 2.2: Cloud Service View

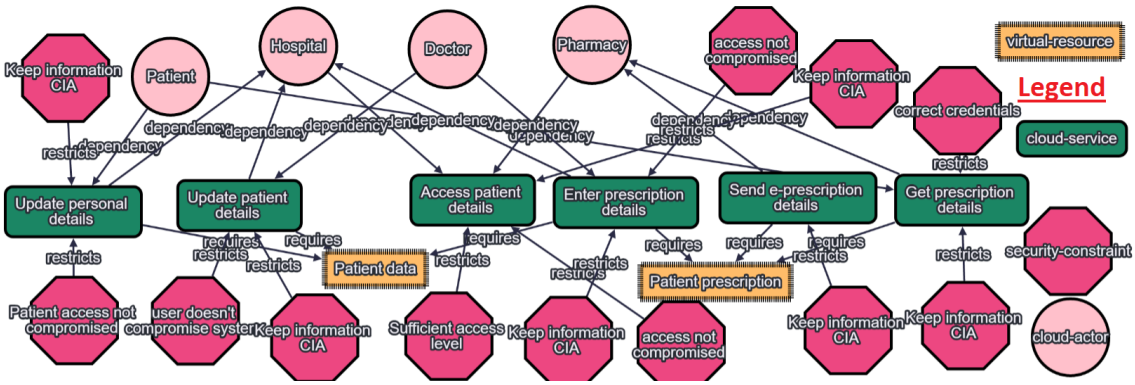

Figure 4.5: A view showing the default candidate cloud services generated during step 2.1.

We use the generated list of candidate cloud service from Step 2.1 as input, presenting a visual model of the actors, cloud services, resources and relationships. This view offers a high level graphical representation of how cloud services interact, what resources are required and the actors involved. From this step the user can proceed to the next step and specify the exact configuration of cloud services. An

example of this view is shown in Figure 4.5, illustrating the dependencies from the actors Patient, Hospital, Doctor and Pharmacy to cloud services and the security constraints.

Step 2.3: Cloud Service Configuration

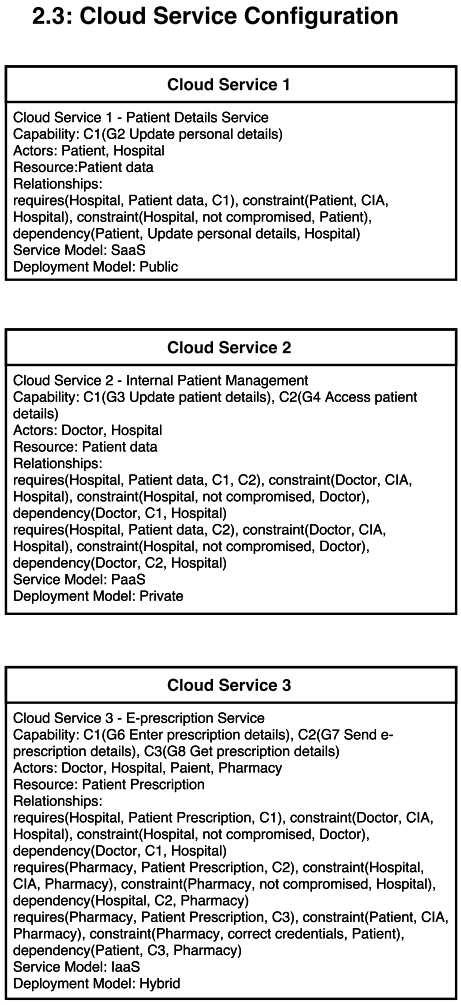

Figure 4.6: Configuring selected cloud services from the running example.

During this step the user refines the attributes of cloud services, such as the cloud service and deployment models for each cloud service. They are also able to view lists of candidate cloud services in order to aggregate related cloud services or modify existing connections. Figure 4.6 shows an example of three cloud services,

where the user completes the cloud service configuration step by entering the desired deployment level, service level and any missing attributes for each cloud service. Optionally they are able to visualise the cloud services to reflect the impact of

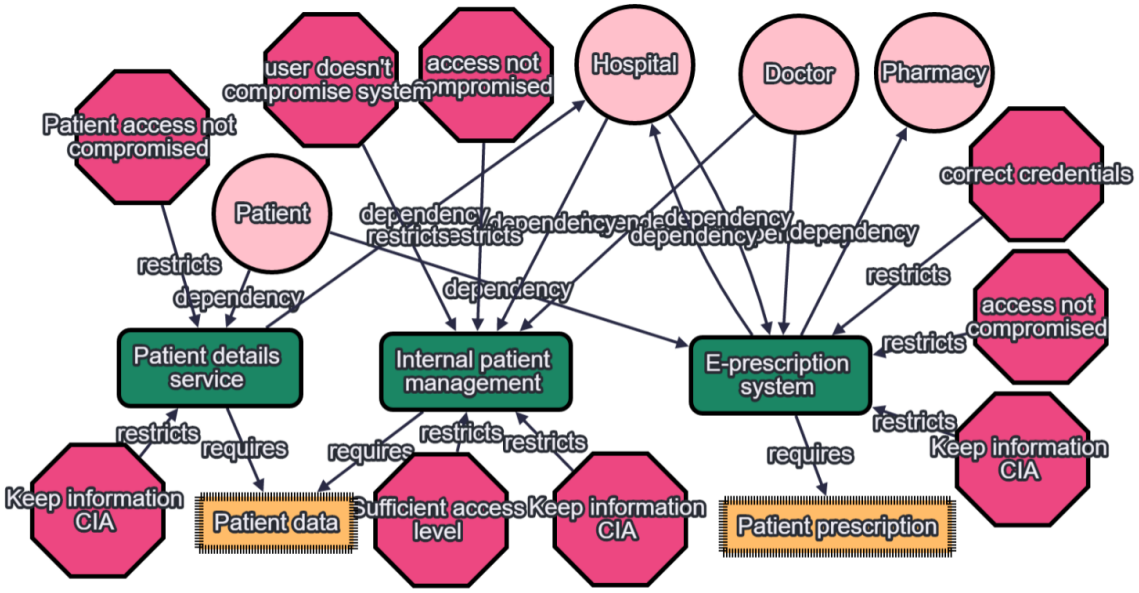

Figure 4.7: A view showing the user-configured cloud services.

selecting and aggregating different services by iterating through Step 2.2. Figure 4.7 shows an example view of cloud services after user input. Once the user is satisfied that the set of cloud services defined sufficiently meets the stakeholder and system needs, they proceed to Activity 3.

4.4 Activity 3: Holistic Cloud Model

The purpose of this activity is to model properties related to cloud services at different conceptual levels and granularity. We examine the identified cloud services from previous activities and model the essential characteristics required for defining a cloud service, which is distributed through our three conceptual layers. The organisation model embodies the social aspects of a software system, such as interactions between stakeholders and the ownership and management of entities. The Application model represents concepts related to the operational level of software systems, where we primary focus on the detailed characterisation of cloud services

to represent entities capable of realising stakeholder and system needs. Finally the infrastructure model provides an abstract mapping of how cloud services are enabled, through the enabling components residing on the physical level. In this step we use the list of cloud services identified in the previous step as input and systematically generate cloud components and properties across three conceptual layers in accordance to our transformation rules and optional user input.

At the organisational layer all actors involved in each of the cloud services are populated in this model, eliminating duplicate entries as needed if they conceptually represent the same actor e.g. references to *doctor* will be the actor group representing the role of a *doctor* as opposed to a single personified doctor.

The cloud services are generated on the application layer as they are conceptually offered and delivered at a software level through programming-enabled interfaces i.e. web technologies, where users would interact with the cloud service through application programming interfaces(API) or web-pages depending on their level of access and role e.g. an application developer would have access to the cloud API, enabling them to directly invoke capabilities and services in order to enhance their development process. In this view cloud services with more than one capabilities are expanded to encapsulate the capabilities they offer, thus providing a fine-grained view of the exact offerings provided by each cloud service. This is visually indicated using the *requires* relationship, where a cloud service requires one or more cloud services to deliver their capability.

Conceptually this represents atomic and composite services, where a cloud service may consist of one or more syntactically related capabilities that collaborate to achieve a common high-level task. An atomic cloud service is synonymous to a cloud service offering a single capability e.g. an example of an atomic cloud service is one that converts a stream of floats to integers, as this embodies the object-oriented design approach in programming where a software system is composed of self-contained objects each capable of achieving a specific task. This encapsulation approach allows programmers to conceptually define specialised objects with re-usability and portability in mind, creating a set of atomic buildings blocks that can be assembled to enable the design of complex systems. Thus a composite cloud service may consist of one or more cloud services, which themselves may be either atomic or composite services. While we are able to model these relationships se-

mantically using our cloud modelling language, visualising the fully expanded links may generate graphically impractical models that are too complex to provide any useful insights. Therefore we will instead generate unique instances of each cloud service and capabilities such that any relationships representing encapsulation and composite services will be indicated through the *requires* relationship between cloud services.

In existing organisational goal models, stakeholder needs are represented through security constraints on goals. This explicitly expresses one or more high-level security needs which should be satisfied during the process of realising stakeholders goals.

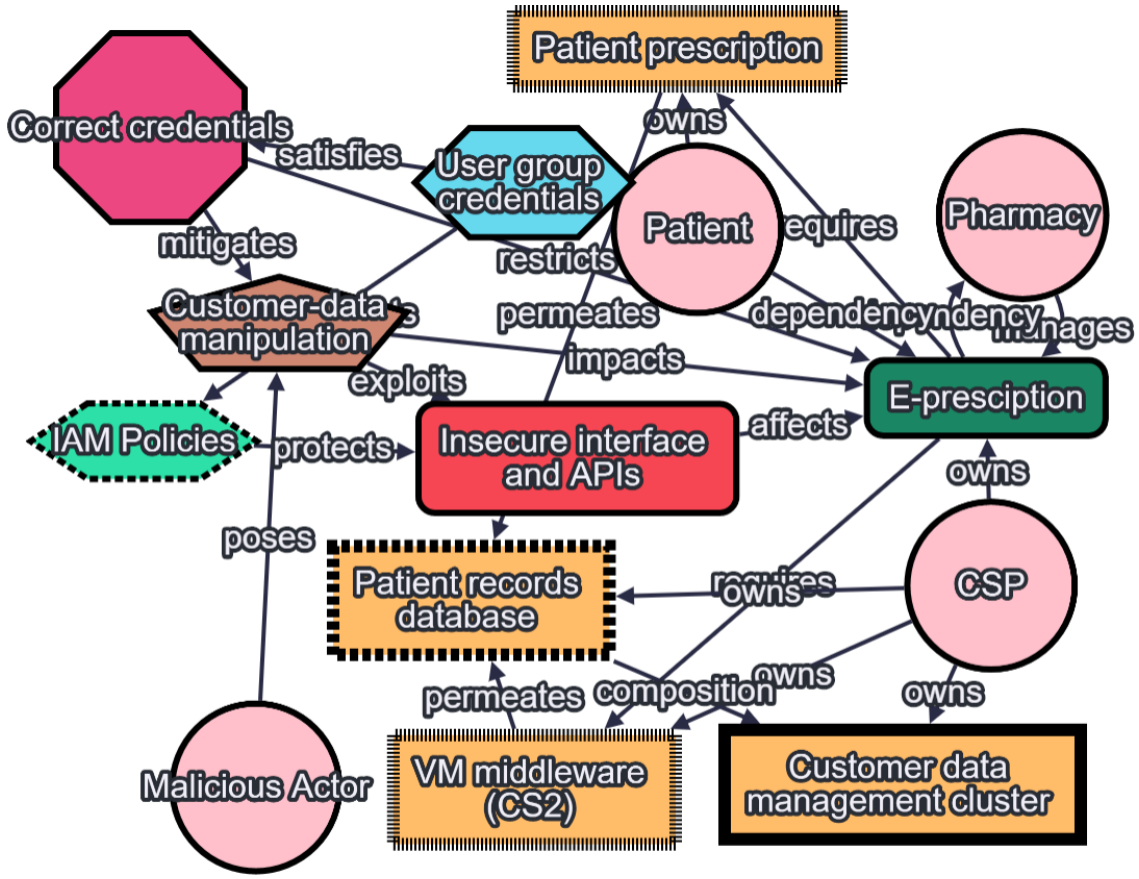

Figure 4.8: Example of the cloud environment view during Activity 3 in the secure cloud process.

In Figure 4.8 we illustrate an example of a view showing a cloud security-

2.1: Organisational Service Identification

Cloud Service 1
Capability: G2 Update personal details Actors: Patient, Hospital Resource:Patient data Relationships: requires(Hospital, Patient data, G2), constraint(Patient, CIA, Hospital), constraint(Hospital, not compromised, Patient), dependency(Patient, G2, Hospital) Service Model: SaaS Deployment Model: Public

Cloud Service 6
Capability: G8 Get presecription details Actors: Patient, Pharmacy Resource:Patient Prescription Relationships: requires(Pharmacy, Patient Prescription, G8), constraint(Patient, CIA, Pharmacy), constraint(Pharmacy, correct credentials, Patient), constraint(Pharmacy, access not compromised, Patient) dependency(Patient, G8, Pharmacy) Service Model: SaaS Deployment Model: Public

Figure 4.9: Example of two atomic cloud services in the cloud service output during Activity 2 in the secure cloud process.

enhanced model, generated based on the cloud services shown in Figure 4.9 and Figure 4.10. In this model the cloud requirements engineer has indicated the satisfaction of the security constraint *"Correct credentials"* restricting the cloud service *"E-presription"*, through the enforcement of a security objective *"User-group credentials"*. The threat *"Customer-data manipulation"* posed by the malicious actor exploits the vulnerability *"Insecure interface and APIs"* and is mitigated by *"Correct credentials"*. Additionally *"User-group credentials"* implements the security mechanism *"IAM Policies"*, which protects against *"Insecure interface and APIs"*. Therefore the developer is able to visualise the security challenges and mitigation techniques by constructing a model of the system-under-design, using concepts and relationships from our cloud modelling language.

Step 3.1: Organisation Concepts

This sub-step provides a fine-grained method of adding properties specific to an organisational context. The output model is then taken as input in the next sub-step, in order to build a complete view of the cloud system under design. Specifically the developer focuses on the following range of concepts and relationships: *goal*, *cloud service*, *actor*, *cloud actor*, *virtual resource*, *security constraint*, *restricts*, *owns*, *manages*, *requires* and *dependency*. The initial step of this activity is to identify and instantiate the stakeholders involved, which is based on the organisational model generated in Activity 1. Alternatively the developer can also input stakeholders based on existing models or their own knowledge. The actors are represented in the

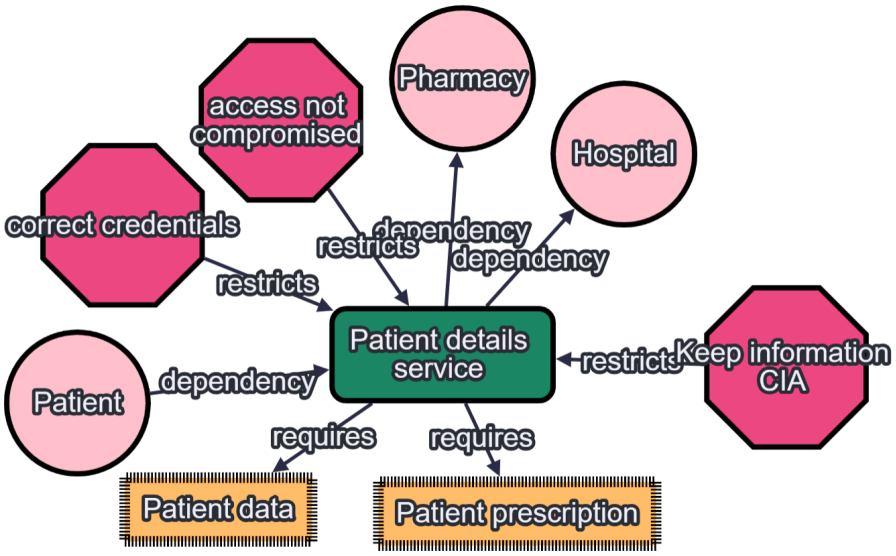

Figure 4.10: Example of one composite cloud service in the organisational cloud service view during Activity 3 in the secure cloud process.

model using the concrete notation of the actor concept, where the developer inputs the name of each actor corresponding to each instance.

The developer then populates the model with the list of identified cloud services, which can either be the output from Activity 2 or generated by the developer in previous work. Each unique cloud service is instantiated using the concrete notation of the cloud service concept. where the developer inputs the name of each cloud service. The specific configurations of each cloud service is then defined through properties; namely the label of the cloud service through *description*, a textual description of the capability in the content of the cloud system through *Capability*, the type of deployment model through *DeploymentModel* and the type of service model through *ServiceModel*. Any assets required by a cloud service is visually indicated by creating the *requires* relationship from a *cloud service* to a *resource* or *virtual resource*. *Security constraints* is placed on a *cloud service*, *resource* or *virtual resource* using the *restricts* relationship. A cloud actor has responsibility for a cloud service or resource, indicated through the *owns* and *manages* relationships. As an example, a cloud actor of type *csp* can posses intellectual and legal ownership over a cloud service, indicated through the *owns* relationship. Consumers of cloud services, indicated through the type *cu*, can be developers of cloud services where

they are responsible for *managing* the configuration and development of a cloud service. Data ownership is indicated using the *owns* relationship from an actor to a virtual resource. These concepts and relationships over goals, cloud services and resources captures the high-level organisational needs in a cloud computing environment.

Step 3.2: Application Concepts

This sub-step focuses on expressing concepts orientated around software properties, specifically the relationships revolving around cloud services. Again the output is a model of the cloud system, the *CEM*, which is taken as input during the next sub-step. The developer focuses on the following range of concepts and relationships: *cloud actor*, *cloud service*, *virtual resource*, *owns*, *manages*, *threat*, *vulnerability*, *security constraint*, *security objective*, *security mechanism*, *restricts*re*satisfies*, *implements*, *mitigates*, *protects*, *exploits*, *impacts* and *requires*. In particular the developer specifies the configuration and details of applications during this stage, by inputting and refining the properties of concepts such as cloud services and virtual resources. As an example, the *ResourceType* narrows down the category of a virtual resource, while the *Vendor*, *Version* and *Type* of a virtual resource is used in the semi-automated reasoning techniques to identify vulnerabilities affecting specific iterations of a product or application according to expert knowledge in publicly available vulnerability databases.

Step 3.3: Infrastructure Concepts

The third sub-step focuses on the tangible aspects of cloud computing, where the supporting infrastructure and the components related to hosting cloud services are modelled. This is crucial as it provides a view of entities and models the relationship between physical properties and their interaction with virtual entities such as cloud services and data, which enables us to express cloud properties such as multi-tenancy and track stakeholders involved. Specifically the developer focuses on the following range of concepts and relationships: *infrastructure node*, *physical infrastructure*, *virtual resource*, *composition*, *permeates*, *cloud actor*, *cloud service*, *owns*, *manages* and the range of security concepts from the modelling language. In particular, the

multi-tenancy cloud characteristic is expressed through the *Tenancy* property in the infrastructure node concept. This property can be used to specify constraints on the infrastructure node concept based on tenancy, limiting the number of relations with specific concepts. As an example, an infrastructure node with the value *"single"* in the *Tenancy* property has the constraint where it cannot be the target of more than one *permeates* relationship. This can be further refined to check if the source of the *permeates* relationships is associated with more than one cloud actor, given an *owns* relationship.

4.5 Activity 4: Cloud Analysis

The purpose of this activity is to identify and model security components such as vulnerabilities, threats and security constraints impacting cloud services and their relationship with other components found at various levels of abstraction within the cloud environment model. This activity can be carried out by cloud system experts or organisational experts with little to no expertise in software security, as the formal framework provides semi-automated guidance towards the identification of security issues and provides a selection of security mitigation techniques.

In order to demonstrate the impact of cloud-specific threats and vulnerabilities and the mitigation strategies, we have identified in the related work a baseline of cloud security issues to guide our assessment of threat and vulnerabilities in our cloud computing models. Our formal framework in allows the sourcing of known cloud computing and software security issues from publicly available databases, which is encoded into an expert knowledge-base. Thus the framework performs a series of queries in order to identify security issues and determine mitigation techniques based on the existing knowledge, thus providing an extensible framework for performing cloud security analysis.

Step 4.1: Cloud Security Analysis

The goal of this sub-step is to take as input, the cloud environment model produced in Activity 3, and perform security analysis to identify vulnerabilities and threats on the system-under-design. During the vulnerability analysis, the specific security

properties identified are dependent on the examined layer or level of abstraction. For example on the organisational layer, we model social attacks at a fine-grained level, because the model provides the expressiveness required to capture social concepts such as stakeholders, strategic needs and security requirements as security constraints on cloud services. Any vulnerabilities identified from the security requirements by the cloud security expert can be added to the cloud environment model manually. Semi-automated assistance is provided through the formal framework described in Chapter 5, which systematically identifies cloud services in the working cloud model and queries the knowledge base in order to generate vulnerabilities based on the attributes of each cloud service. Tool-support is also available for developers, using the SectroCloud module described in Section 5.4 of Chapter 5 to create graphical models of the system-under-design.

Based on security issues of deployment and service models identified in the literature review chapter, we are then able to apply security properties to models through pattern matching. For example given a known cloud threat which targets a specific cloud service model, the analysis will identify cloud services in the cloud environment model matching these specific properties and assign the cloud threat to the marked cloud services. Consider the cloud vulnerability "V06 vulnerability in hypervisors" (Hughes & Cybenko, 2014) which affects cloud services deployed through the Infrastructure as a Service (IaaS) service model, this will automatically generate the vulnerability "V06 vulnerability in hypervisors" and link all known cloud services in the cloud environment model with the IaaS attribute to the vulnerability through the "Affects" relationship.

Step 4.2: Security Mitigation Analysis

The goal of this sub-step is to guide the developers through the process of creating a mitigation strategy, in order to address threats and vulnerabilities identified in system-under-design. We take as input the cloud environment model, with the assumption that there exists one or more threats or vulnerabilities. In the case where there exists no threats or vulnerabilities in the system-under-design, performing this analysis will not produce any new knowledge. By default, a security constraint is unsatisfied given that there are no incoming relationships of type *"implements"* from

one or more security objectives. Therefore unsatisfied security constraints can be enforced through security objectives, where each security objective associated with a security constraint through the *"implements"* relationship is said to satisfy the respective security constraint. In addition, any security properties in a security objective would address one or more of the corresponding security properties in the security constraint. For example a security objective with the security property *"Integrity"* will satisfy a security constraint with the security property *"Integrity"*, given that the *"implements"* relationship exists from the security objective to the security constraint. Thus when all security properties in a security constraint has been fully addressed by one or more security objectives with corresponding security properties, the security constraint is therefore said to be satisfied. Security objectives are realised through the implementation of security mechanisms and security controls, where a security objective is said to be implemented given that one or more security mechanisms with the *"implements"* relationship is associated with the respective security objective. Similar to the satisfaction of security constraints, a vulnerability is said to be unsatisfied given that it has no incoming relationships of type *"protects"* from one or more security mechanisms. Therefore unsatisfied vulnerabilities can be protected through security mechanisms, where each security mechanisms would address one or more vulnerabilities. In the general case, the mitigation check is evaluated given that there exists at least one security mechanism with is associated to a vulnerability through the *"protects"*, thus addressing and mitigating the vulnerability from an abstract point of view. From a technical perspective, this method does not take into account the degree of satisfaction provided by multiple potential solutions implemented through different security mechanisms. The general case also does not provide a guarantee that the implementation of the security mechanism will fully mitigate the vulnerability. Therefore the next step required to provide fine-grained mitigation methods is through pattern-matching and querying expert knowledge from a global security vulnerabilities database. The formal framework defined in Section 5.1 of Chapter 5 supports the querying of a knowledge base containing known cloud vulnerabilities and mitigation techniques, which can be realised through the unification of a vulnerability with security mechanisms representing high-level implementations of mitigation techniques.

Step 4.3: Transparency Analysis

In this sub-step, the cloud security requirements of the system-under-design is summarised from the cloud environment model, focusing on the cloud actors responsible for compromising or protecting the system-under-design. Taking as input the cloud environment model, we focus on enumerating the security constraints defined in the system, the set of vulnerabilities and threats identified, mitigation strategy to address identified issues and the cloud actors responsible for managing these issues. The details of this sub-step is expanded in in Section 5.3 of Chapter 5, specifically referring to the security analysis techniques.

4.6 Chapter summary

In this chapter the activities for eliciting the security requirements of a secure cloud computing systems is described, taking into account the steps a security requirements engineer should follow when applying concepts from the cloud modelling language. The key feature of the secure cloud process is to systematically define a model of the cloud computing system using the concepts of cloud services, virtual resources, infrastructure nodes, physical infrastructures and relationships between cloud actors and their assets. To understand cloud security requirements, the model of the cloud computing environment is examined from three perspectives, in terms of the organisational, application and infrastructure aspects.

Chapter 5

Semi-Automated Reasoning Support

Software systems in a cloud environment are complex and involve multiple actors, each with their own individual goals, assets and security needs. In such cases ensuring transparency in the cloud environment requires identifying and satisfying security needs, where issues such as unsatisfied or conflicting security requirements and ambiguous delegation of responsibility need to be resolved. However when developers apply the Secure Tropos methodology and by extension, our secure cloud process, it is a manual task with multiple complex steps. Without automated reasoning support, the developer risks introducing issues such as missing system components or creating conflicting requirements due to human error, especially as the modelled system scale upwards in complexity. For example in a cloud system with five cloud actors and thirty infrastructure nodes over ten cloud services, a developer needs to examine the details of each infrastructure node to identify threats and vulnerabilities, trace the responsibility of the cloud actors through the cloud services and determine the mitigation strategy to satisfy security needs.

Therefore in this chapter we describe three analysis techniques providing reasoning support through the process of deriving security requirements and cloud security requirements from cloud models. Our contributions in this chapter enhances the Secure Tropos methodology through the addition of semi-automated cloud security analysis. The analysis provides developers with decision support throughout the

process of modelling and analysing cloud environments, to assist them in eliciting cloud security requirements. For example given a cloud model describing the security needs, assets and relationships of cloud actors in a cloud environment, we propose the following set of analysis techniques: (i) the cloud security analysis identifies cloud threats and security vulnerabilities, based on knowledge derived from public security databases, (ii) the security mitigation analysis identifies and validates security constraints, objectives and mechanisms, proposing alternatives for mitigating issues identified in the cloud security analysis, and (iii) is the transparency analysis to identify the parties responsible for ensuring the satisfaction and implementation of cloud security needs within the cloud environment. In Section 5.1 we unambiguously define concepts and relationships of the cloud analysis using the instance syntax defined previously in Chapter 3.5. In Section 5.2 we describe the concepts behind our security knowledge-base and how we extract and map information from various vulnerability and cloud security controls data sources to our cloud modelling language, in order to enhance the automated reasoning through domain-specific expert knowledge. In Section 5.3 we present our three analysis techniques and illustrate the process of the security analysis, outlining each step of the analysis with expected input and output.

5.1 Formal Analysis Concepts

This section introduces the syntax and semantics required to support semi-automated cloud security analysis, building upon the instance syntax of the Secure Cloud Language described in Section 3.5. In order to perform cloud security analysis, we take as input cloud models constructed from our cloud modelling language. However for the analysis to process the information in cloud models without ambiguity, we need to first formally define the semantics of the modelling language. That is we take the instance syntax which specifies the concepts and relationships of cloud models as input for the analysis. In order to demonstrate our formal concepts, we only present the full description of each instance using the instance syntax where needed. As an example the instance syntax describing an instance of an infrastructure node will be shortened to *IN1*, unless the properties within the instance is required to understand an example, in which case the full description is given as

IN1(VMWare,storage,multiple).

We extend a selection of the formal concepts presented by Paja (Paja, 2014), which describes socio-technical systems (STS) from a requirements engineering perspective, where the concept of actors, relationships and technological components are formalised in order to provide formal analysis and reasoning for conflicting requirements. Cloud computing systems share common characteristics with STS, for example in a social environment where actors interact with software systems and the relationships between technical components such as software and infrastructure nodes also require contextual descriptions. Therefore this provides a foundational base to extend the security requirements engineering domain with cloud computing concepts and enrich our research through formal analysis, in order to deduce cloud security requirements.

Definition 5.1.1. (**Resource knowledge base**) There exists different types of assets in a cloud computing environment, namely the physical, virtual resources and their relationships. We define the concept of a resource knowledge base to consolidate all known resources within the scope of the given cloud model, thus providing the means for automated analysis techniques to parse the required data.

The resource knowledge base is a tuple RKB where:

$$RKB = \langle VR, PI, IN, VPIR \rangle$$

given VR is a set of virtual resources , PI is a set of physical infrastructure, IN is a set of Infrastructure nodes, and $VPIR$ is a set of relationships formalising the *composition* and *permeates* relationships defined in Chapter 3.3. The *composition* relationship indicates the geographical grouping of individual infrastructure nodes over a physical infrastructure. The *permeates* relationship indicates the flow of data between resources, which enables the propagation of security needs. We define the composition and permeates relationships as follows:

- *composition(IN, PI)*: infrastructure node IN is part of the physical infrastructure PI;
- *permeates(VR, IN)*: virtual resource VR permeates the infrastructure node IN;

For example given a scenario where the resource knowledge base contains the hierarchical components of a data-centre, a query can be performed to identify all virtual data of user records residing on a specific infrastructure node. Figure 5.1 illustrates a case where a cloud service requires virtual data permeating separate instances across two geographically disparate physical infrastructure. In this case given the graphical model shown in Figure 5.1, we obtain the resource knowledge base:

$$RKB = \Big\langle \{VR1, VR2\}, \{PI1, PI2\}, \{IN1, IN2\}, \{permeates(VR1, IN1), permeates(VR1, IN2), permeates(VR2, IN2), composition(IN1, PI1), composition(IN2, PI2)\}\Big\rangle$$

Thus this format of representing information enables analysis techniques to infer that the infrastructure node *IN2* is part of the composition per *composition(IN2,PI2)*, and that the physical infrastructure *PI2* is geographically based in the US from *PI2(PI2,US)*. Additionally the permeates relationships on virtual resources *VR1* and *VR2* in *permeates(VR1,IN2)* and *permeates(VR2,IN2)* indicates that the infrastructure node *IN2* hosts both virtual resources.

We extend the concepts of the "informational knowledge base" proposed by Paja in (Paja, 2014) because concepts from the social domain in STS is crucial for capturing stakeholder interactions and relationships in cloud computing systems. However the information found in cloud systems also extends into the physical domain, in the form of hardware components and resources. Therefore we use the concepts from the application cloud model in 3.6.2 and the infrastructure cloud model in 3.6.3 to formalise the relationships between the physical components such as servers, networks and storage and virtual resources such as customer data and virtual machines.

Definition 5.1.2. (**Intentional Relationship**) The intentional relationship defines a relationship within the scope of an individual Actor A, describing their responsibilities over the set of resources R and set of cloud services CS in a cloud environment. This definition formalises the *owns*, *manages* and *poses* (poses is defined in Figure 3.1) relationships from actors to components defined in Section 3.2. Specifically the *owns* relationship determines an actors ownership over resources, where an actor

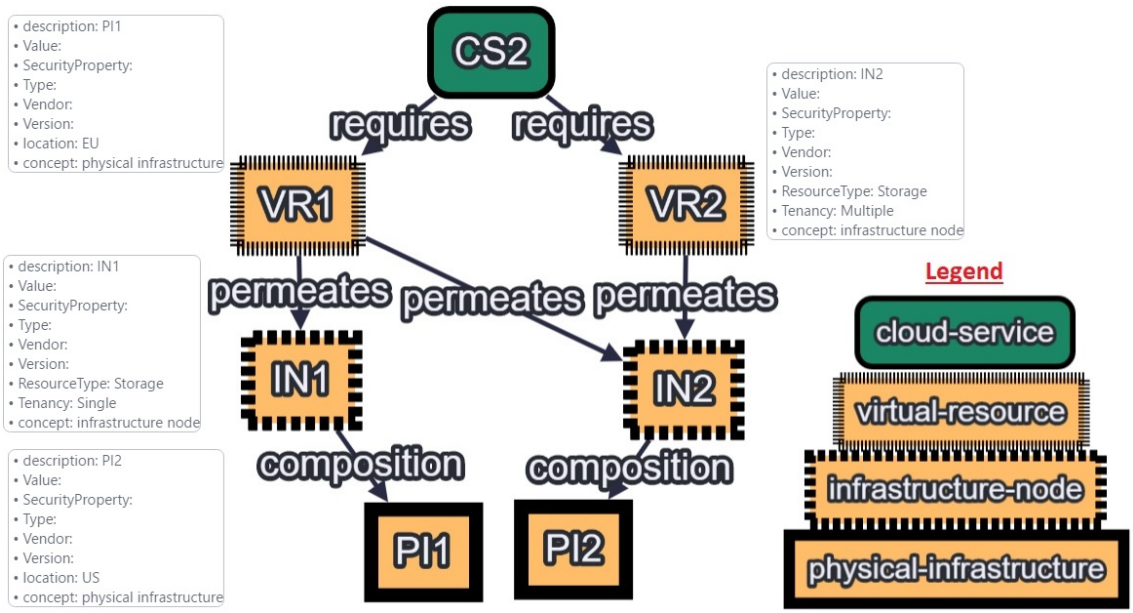

Figure 5.1: Graphical example of a cloud service, the resources required and infrastructure.

may own one or more cloud service, virtual resource, physical infrastructure or infrastructure nodes. The *manages* relationship identifies the parties responsible for managing security controls and configuration of cloud services. The *poses* relationship indicates a malicious actors intent to cause harm to the system through threats. We define the relationships as follows:

- *owns(A, R)*: actor A owns the set of resources R, where R=$\{V1,...,Vn,PI1,...,PIn,IN1,...,INn\}$;
- *owns(A, CS)*: actor A owns the set of cloud services CS, where CS=$\{CS1,...,CSn\}$;
- *manages(A, CS)*: actor A manages the set of cloud services CS, where CS=$\{CS1, ...,CSn\}$;
- *poses(MA, T)*: malicious actor MA poses the set of threats T, where T=$\{T1,...,Tn\}$.

An actor may own zero or more resources, which can be of the type virtual resource, physical infrastructure or infrastructure node. This represents the ownership of tangible assets such as physical servers, data-centres or networks and the ownership

of virtual resources such as personal data, software and computational data. For example in Figure 5.2 the cloud actor *CSP1* owns the physical infrastructure *PI1* and the infrastructure node *IN1*, indicated formally as:

$$owns(CSP1, \{PI1, IN1\})$$

Thus an analysis technique can use this information to determine the cloud actor *CSP1* as responsible for enforcing security measures to protect *PI1* and *IN1*. The cloud actor *Developer* manages the cloud service *CS2*, indicated formally as:

$$manages(Developer, \{CS2\})$$

This indicates the cloud actors responsibility to ensure any security needs placed on the cloud service also extends to the resources required by the cloud service, such as *VR1*. Finally the actor *Malicious Actor* represents an actor with malicious intent to compromise the system, where they pose the threat *Shared technology vulnerabilities* to exploit the infrastructure node *IN1*, which is indicated formally as:

$$poses(MaliciousActor, \{Sharedtechnologyvulnerabilities\})$$

This relationship captures the actors that poses threats on the system and traces the impact of the threat to the specific components, allowing developers to model the impact of a given threat on assets and identify the threatening party.

We now define the actor model to describe the assets and their relationships with an actor within the scope of a single actor instance.

Definition 5.1.3. (**Actor model**) An actor model AM is a tuple, where

$$AM = \big\langle A, CS, R, IRL, T \big\rangle$$

given that A is an actor, CS is a set of cloud services, R is a set of resources, IRL is a set containing the intentional relationships of actor A, and T is a set containing

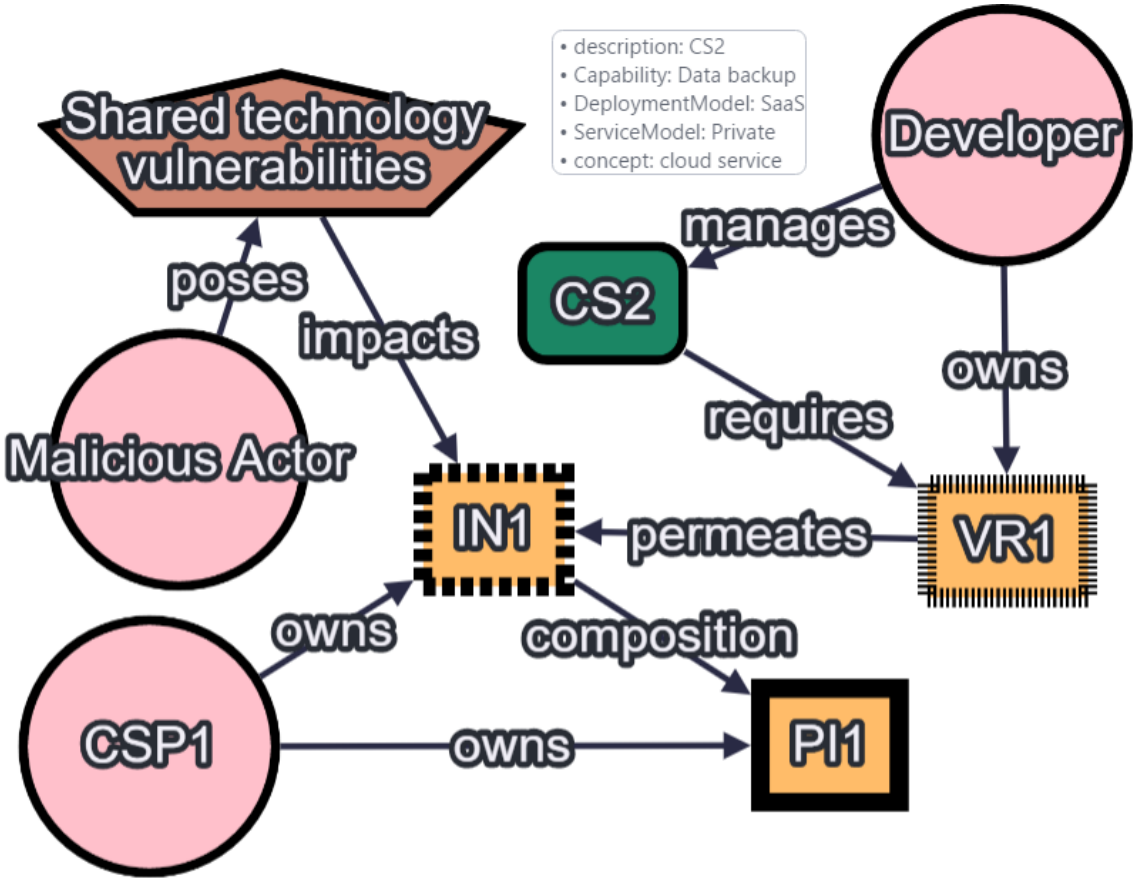

Figure 5.2: Graphical example of the owns, manages and poses relationships.

the types of the cloud actor.

The actor model describes an instance of the actor concept, within the scope of its relationships in a cloud environment. This definition encapsulates an actor instance, cloud services associated with the actor, resources associated with the actor, a set of intentional relationships to the actor and the set of cloud actor types. Thus each instance of an actor model captures the specific resources required from virtual resource, infrastructure node and physical infrastructure, the manages, owns and poses (poses is defined in Figure 3.1) relationships and the types of the cloud actor from Chapter 3.4.1. For example the actor model for the cloud actor *Developer* in Figure 5.2 is as follows:

$$AM = \Big\langle Developer, \{CS2\}, \{VR1, IN1\}, \{owns(Developer, VR1),$$
$$manages(Developer, CS2), permeates(VR1, IN1)\}, \{Cloud\ Service\ Provider\}\Big\rangle$$

Definition 5.1.4. (**Organisational relationship**) The organisational relationship defines a relationship of one or more actors in a cloud computing system within the scope of an organisational context, given a resource, security need or dependency. We now formally define the relationships presented in the previous chapters, specifically the Secure Tropos *dependency* and *secure dependency* in Section 2.3.2 and the *restricts* and *requires* relationships in Section 3.3:

- *dependency(A1, D, A2)*: dependee actor *A1* depends on depender actor *A2* for dependum *D*, where *D* can be a resource *R* or goal *G*;
- *restrictsresource(SC, R, A)*: security constraint *SC* restricts the resource *R* and is imposed to the actor *A*;
- *restrictsgoal(SC, G, A)*: security constraint *SC* restricts the goal *G* and is imposed to the actor *A*;
- *securedependency(SC1, SC2, dependency(A1, D, A2))*: given a dependency relationship *dependency(A1, D, A2)*, *SC1* is a set containing *restrictsresource(SC, R, A)* or *restrictsgoal(SC, G, A)* relationships where zero or more security constraints *SC* restricts the dependum *D* and is imposed to the actor *A1*, *SC2* is a set containing *restrictsresource(SC, R, A)* or *restrictsgoal(SC, G, A)* relationships where zero or more security constraints *SC* restricts the dependum *D* and is imposed to the actor *A2*;
- *requiresgoal(G, R)*: goal *G* requires the set of resources *R*, where *R*={ *V1,...,Vn, PI1,...,PIn,IN1,...,INn*};
- *requiresresource(R1, R2)*: resource *R1* requires the set of resources *R2*, where *R2*={ *V1,...,Vn,PI1,...,PIn,IN1,...,INn*};
- *requiresgoalcs(G, CS)*: goal *G* requires the set of cloud services *CS*, where *CS*={ *CS1, ...,CSn*};

- *requiresresourcecs(R, CS)*: resource R requires the set of cloud services CS, where CS=$\{CS1,...,CSn\}$;

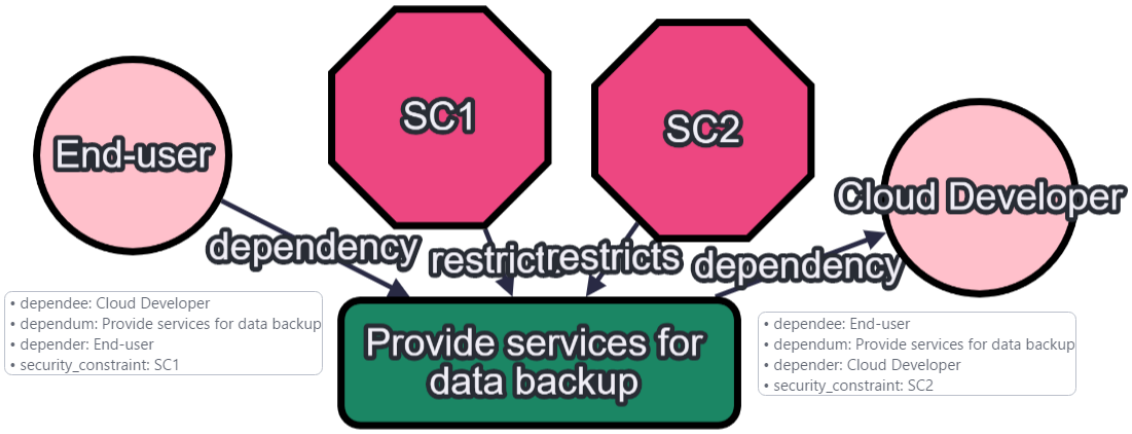

Figure 5.3: Graphical example of a secure dependency relationship.

The organisational relationship defines the validity of dependencies and security needs of actors from an organisational perspective, with respect to the application and infrastructure concepts. This enumerates and formally defines the dependency, restricts and secure dependency relationships in Section 2.3.2. Therefore we are able to enumerate dependencies between actors given a resource of type virtual resource, infrastructure node or physical infrastructure, and a goal or cloud service. Security needs on assets and the actor responsible for satisfying these needs is described through the restricts relationship, where a security constraint restricts a resource or goal and is imposed to an actor. We specify two types of the restricted relationship dependent on the restrained entity, in order to differentiate between constraints on resources and goals. The secure dependency relationship builds upon the dependency and restricts relationships to define the relationship between two actors in order to identify the security needs placed on resources or goals.

We also define the requires relationship through several specific definitions, in order to capture all possible cases. Recall that the resource concept has three specialisations; virtual resource, physical infrastructure and infrastructure node, therefore a resource can be one of the three types defined. Additionally recall the goal concept has the specialisation cloud service. Specifically to formalise all cases of the requires

relationship, we define the derivations where a goal requires a resource, a resource requires another resource, a goal requires a cloud service and a resource requires a cloud service.

Figure 5.3 illustrates a secure dependency relationship between two cloud actors, with a cloud service dependum and two security constraints. The cloud actor *"End-user"* is the dependee in the dependency relationship with the cloud actor *"Cloud Developer"* as the depender and the goal *"Provide services for data backup"* as the dependum. This dependency relationship is enumerated as follows: *dependency(End-user, Provide services for data backup, Cloud Developer).* Both security constraints in Figure 5.3 restrict the goal, which represents the security needs. This is enumerated as:

$$restrictsgoal(SC1, Provideservicesfordatabackup, CloudDeveloper) and$$
$$restrictsgoal(SC2, Provideservicesfordatabackup, End-User)$$

Finally the security dependency relationship is enumerated as:

$$securedependency(SC1, SC2, dependency(End-user,$$
$$Provideservicesfordatabackup, CloudDeveloper))$$

where the set *SC1* in the security dependency relationship is as follows (Note: The SC1 inside the set is the instance SC1, not the set SC1 again):

$$SC1 = \{restrictsgoal(SC1, Provideservicesfordatabackup, CloudDeveloper)\}$$

and the set *SC2* in the security dependency relationship is as follows (Note: The SC2 inside the set is the instance SC2, not the set SC2 again.):

$$SC2 = \{restrictsgoal(SC2, Provideservicesfordatabackup, End-User)\}$$

Definition 5.1.5. (**Cloud Service Model**) A cloud system consists of one or more cloud services, where each cloud service describes their service and deployment model types and a set of required resources. Encoding this information through a cloud

service model allows the developer to perform automated reasoning on specific cloud service instances, within the scope of the system. We define a cloud service model *CSM* as a tuple:

$$CSM = \langle CS, DM, SM, R, OR \rangle$$

given that *CS* is an instance of a cloud service, *DM* is a cloud deployment model of type *public, private, hybrid or community*, *SM* is a set consisting of one or more unique cloud service model types *SM*={*SaaS, PaaS, IaaS*}, and *R* is a set of resources *R*={ *VR1,...,VRn,PI1,...,PIn,IN1,...,INn*} where $|R| >= 1$ and *OR* is a set containing the organisational relationships between the cloud service and associated resources or goals.

The cloud service model is encoded for each cloud service identified in the software system, where instances of the cloud service model aggregates attributes, properties and relationships connected to each cloud service. The service model of the cloud service indicates the level of abstraction in terms of resource allocation and delegation of responsibilities. For instance a cloud service with a service level of *IaaS* denotes the allocation of physical resources such as infrastructure nodes, where the actors responsible for the cloud service and its components are indicated through relationships such as manages, cloud dependency and owns.

The cloud service model *SM* is represented through a set consisting of one or more unique types of cloud services; Software-as-a-service(SaaS), Platform-as-a-Service(PaaS), Infrastructure-as-a-Service(IaaS).

The type of cloud service model associated with a cloud service determines the category of threat and vulnerabilities impacting the specific instances of a cloud service.

For example the *CSM* for the cloud service *CS2* in Figure 5.2 previously is as follows:

$$CSM = \langle CS2, private, \{SaaS\}, \{VR1\}, \{\} \rangle$$

Definition 5.1.6. Cloud vulnerability model We define a cloud vulnerability

model to describe the instances of vulnerabilities affecting assets and cloud services in a cloud system. This model is defined in order to facilitate the construction of analysis techniques, which require details of the vulnerabilities affecting specific instances of the cloud service and resource concepts in a cloud model. The cloud vulnerability model *CVM* is defined as a tuple

$$CVM = \langle V, SM, CS, R, Rel \rangle$$

where V is a set of vulnerabilities affecting goals and resources through the *affects* relationship and $|V| >= 0$, SM is a set of security mechanisms protecting vulnerabilities through the *protects* relationship and $|SM| >= 0$, CS is a set consisting of goals and cloud services where $|CS| >= 0$, R is a set consisting of *virtual resources*, *infrastructure nodes* and *physical infrastructures* where $|R| >= 0$ and Rel is a set of relationships between these concepts. We now define the following relationships based on the *affects* and *protects* relationships in the fragment of the Secure Tropos metamodel shown in Figure 3.1 of Section 3.2:

- *affects(Vul, R)*: the vulnerability *Vul* affects the resource R, where R can be of type virtual resource *VR*, infrastructure node *IN* or physical infrastructure *PI*.

- *affects(Vul, G)*: the vulnerability *Vul* affects the goal G, which can be a cloud service *CS*.

- *protects(SecMech, Vul)*: the vulnerability *Vul* is protected by the set *SecMech*, which contains one or more security mechanisms where *SecMech*={*SecMech1,... ,SecMechn*}.

The cloud vulnerability model describes vulnerabilities which affect an instance of a cloud service, goal, resource, virtual resource, infrastructure node or physical infrastructure. The security mechanisms required to protect against specific vulnerabilities are also described in the cloud vulnerability model. As an example Figure 5.4 illustrates a generic case where a vulnerability *V1* affects a cloud service *CS1* and is protected by the security mechanism *SM1*. Another vulnerability *V2* affects a virtual resource *VR1*. In this case the relationships are enumerated

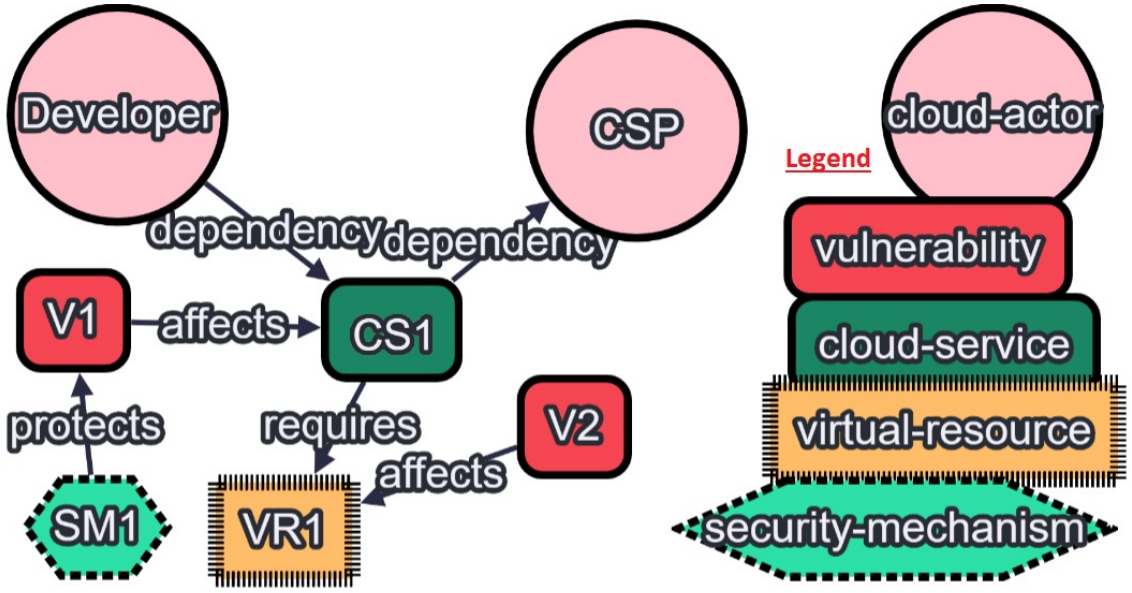

Figure 5.4: Graphical example of relationships in a cloud vulnerability model.

through the cloud vulnerability model: $CVM = \langle \{V1,\ V2\}, \{SM1\}, \{CS1\}, \{VR1\}, \{affects(V1,CS1), affects(V2,VR1),\ protects(\{SM1\}, V1)\} \rangle$.

Definition 5.1.7. Cloud threat model The cloud threat model captures the threats impacting cloud services or resources in a cloud system. This enumerates the threats targeting specific cloud services or resources in order to assist the developer in identifying threats given a list of assets. The cloud threat model CTM is defined as a tuple:

$$CTM = \langle T, V, SO, SM, SC, CS, R, Rel \rangle$$

where T is a set of threats exploiting vulnerabilities through the *exploits* relationship and $|T| >= 0$, V is a set of goals and resources impacted by a threat through the *impacts* relationship and $|V| >= 0$, SO is a set of security objectives satisfying security constraints through the *satisfies* relationship and $|SO| >= 0$, SM is a set of security objectives implementing security mechanisms through the *implements* relationship and $|SM| >= 0$, SC is a set of security constraints mitigating threats through the *mitigates* relationship and $|SC| >= 0$, CS is a set consisting of goals and cloud services where $|CS| >= 0$, R is a set consisting of *virtual resources*, *infrastructure nodes* and *physical infrastructures* where $|R| >= 0$ and Rel is a set

of relationships between these concepts. We now define the following relationships based on the *exploits*, *impacts*, *satisfies*, *implements* and *mitigates* relationships in Section 3.3:

- *exploits(Thr, Vul)*: the threat *Thr* exploits the vulnerability *Vul*.
- *impacts(Thr, G)*: the threat *Thr* impacts the goal *G*, which can be also be the cloud service specialisation.
- *impacts(Thr, R)*: the threat *Thr* impacts the resource *R*, which can be of type virtual resource, infrastructure node or physical infrastructure.
- *satisfies(SecObj, SecCo)*: the security objective *SecObj* satisfies the each element in the set of security constraints *SecCo*, given a set of security constraints *SecCo1*=$\{$*SecCo1,...,SecCon*$\}$ where $|SecCo| >= 0$.
- *implements(SecObj, SecMech)*: the security objective *SecObj* implements the each element in the set of security mechanisms *SecMech*, given a set of security mechanisms *SecMech*=$\{$*SecMech1,...,SecMechn*$\}$ where $|SecMech| >= 0$.
- *mitigates(SecCo,Thr)*: the security constraint *SecCo* mitigates the threat *Thr*.

The cloud threat model focus on threats which exploit identified vulnerabilities, how to satisfy security constraints using security objectives and which security mechanisms are implemented. Referring to Figure 5.5, the example demonstrates the *exploits*, *impacts*, *satisfies* and *implements* relationships. Specifically a threat *T1* exploits a vulnerability *V1*, given the relationship *exploits(T1,V1)*. *T1* also impacts the cloud service *CS1*, captured through the relationship *impacts(T1,CS1)*. The security objective *SO1* satisfies a security constraint *SC1* using the relationship *satisfies(SO1,*$\{$*SC1*$\}$*)*. Finally the security objectives *SO2* implements the security mechanism *SM2*, described through the relationship *implements(SO2,*$\{$*SM2*$\}$*)*. The information visualised in Figure 5.5 is enumerated through the cloud threat model as follows: $CTM = \langle$ $\{$*T1*$\}$,$\{$*V1*$\}$,$\{$*SO1,SO2*$\}$,$\{$*SM1,SM2*$\}$,$\{$*SC1*$\}$,$\{$*CS1*$\}$,$\{\}$, $\{$*exploits(T1,V1), impacts(T1,CS1), satisfies(SO1,*$\{$*SC1*$\}$*), implements(SO2,*$\{$*SM2*$\}$*), mitigates(SC1,T1)*$\}$ $\rangle$.

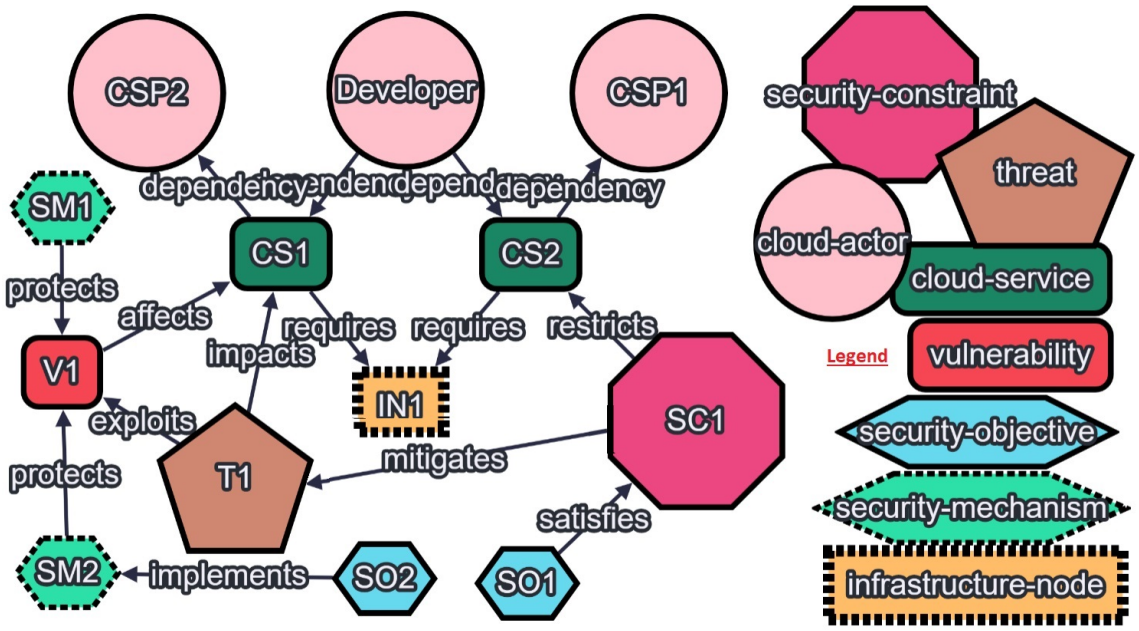

Figure 5.5: Graphical example of relationships in a cloud threat model.

Definition 5.1.8. Cloud environment model

The information captured in a cloud environment is enumerated in a single conceptual model in our framework, from which the organisational, application and infrastructure views are derived to present information at a lower level of granularity. By aggregating information into a single model, we ensure that any changes to the system is consistent across different views. Each model defined in this section is constructed according to the information given in the model definition, using the instances of the system-under-design. That is, a model will use information given from the system-under-design and ensure that consistency between each model is maintained by only populating the defined set of values specific for each model. As an example, a *CVM* will hold information extracted from the system-under-design consisting of the following; a set of vulnerabilities, a set of security mechanisms, a set of cloud services and goals, a set of consisting of resources, virtual resources, infrastructure nodes, physical infrastructures and a set of relationships such as *affects* and *protects*. Furthermore, tool support will provide a semi-automated way to ensure that a consistent model for the cloud system is produced when these models are grouped together. This can be achieved by defining a schema in the tool cor-

responding to the definitions provided for each model, which will ensure that only the concepts associated with each model will be accepted. That is, the user of the tool will not be allowed to create relationships or instances of concepts which are not defined in the schema.

Therefore in order to maintain consistency between the models and relationships defined in this section, we define the cloud environment model as a super-set containing the resource knowledge base, actor model, organisational relationships, cloud service model, cloud vulnerability model and cloud threat model. For example any changes made to a subset of the cloud model are reflected in the super-set. We now define the cloud environment model *CEM* as follows:

$$CEM = \langle RKB, AM, OR, CSM, CVM, CTM \rangle$$

where *RKB* is the resource knowledge base, *AM* is a set of actor models, *OR* is a set of organisational relationships, *CSM* is a set of cloud service models, *CVM* is a set of cloud vulnerability models and *CTM* is a set of cloud threat models.

5.2 Security Knowledge

The primary users of our cloud security framework are developers of cloud computing systems, with expertise in cloud security and requirements engineering. However the framework also supports developers without domain-specific knowledge in cloud security, specifically regarding technical vulnerabilities, mitigation and implementation-level details. This contributes towards a decision-support framework allowing developers without in-depth knowledge of cloud security to model and analyse cloud systems, increasing the accessibility and uptake of our work. This is achieved through consulting publicly available sources of domain-specific knowledge, such as expert databases on security vulnerabilities or cloud security controls.

In this subsection we describe how we examine and extract data from vulnerability databases and cloud security controls, in order to enhance our automated reasoning techniques with expert knowledge. We present our techniques for extracting data from the sources described and then define transformation rules mapped

to our cloud modelling language and formal analysis concepts.

5.2.1 Common Weakness Enumeration

The Common Weakness Enumeration(CWE) is a "formal directory of common software weaknesses that can occur in software's architecture, design, code or implementation that can lead to exploitable security vulnerabilities" [1]. CWE is a free open standard maintained by the MITRE Corporation, a government-funded organisation focusing on providing standards for the information security community. The CWE define software weaknesses as " flaws, faults, bugs, vulnerabilities, and other errors in software implementation, code, design, or architecture that if left unaddressed could result in systems and networks being vulnerable to attack.". Based on this definitions, they define software weaknesses as errors that can lead to software vulnerabilities. Thus the CWE provides a common baseline for the identification of weaknesses, mitigation and prevention in the scope of code security assessment. That is, the CWE focuses on the underlying vulnerability, as opposed to an instance within a specific product or system.

CWE also provide external groupings such as Top-N lists from OWASP, which are expressed through subsets of entries. First we align the definitions from CWE to our concepts, in order for our security analysis to express the required concepts based on our definitions.

The term "Weakness" in CWE is "a type of mistake in software that, in proper conditions, could contribute to the introduction of vulnerabilities within that software.". Thus a weakness in CWE corresponds to our concept of an *Attack Method*, which is a property of the *Vulnerability* concept.

The term "Vulnerability" in CWE is "an occurrence of a weakness (or multiple weaknesses) within software, in which the weakness can be used by a party to cause the software to modify or access unintended data, interrupt proper execution, or perform incorrect actions that were not specifically granted to the party who uses the weakness.". In this case our definition of a vulnerability coincides with the CWE definition, so references to a vulnerability in CWE corresponds to a vulnerability from our concepts.

[1] https://cwe.mitre.org/about/faq.html

The term “Resource” in CWE is “a vulnerability theory term for an object or entity that is accessed or modified within the operation of the software, such as memory, CPU, files, or sockets. Resources can be system-level (memory or CPU), code-level (function or variable), or application-level (cookie or message).”. Thus our concept of a resource is a generalisation, where our virtual resource concept is a specialisation of resource, which represents code-level and application level resources. Our infrastructure node concept is a specialisation of resource, which represents system-level resources. In addition our physical infrastructure concept is a specialisation of resource, which represents a tangible resource in the operational environment and aggregates one or more infrastructure nodes.

Another method for extracting data is to obtain the latest core content in a schema such as XSD (XML Schema Definition), where we define rules for extraction based on information stored in a pair consisting of a tag and value.

The term “tag” refers to the HTML definition: “Tags contain elements which provide instructions for how information will be processed or displayed.”. A tag consists of a pair of <> symbols, where the name of the tag is enclosed within each pair. In this case a tag is opened with a <> symbol, followed by one or more elements and closed with the < / >, as shown in the following code extract:

$$< tag > information\ or\ instructions\ enclosed\ here < /tag >$$

The term “value” refers to a description, parameter or variable enclosed between a pair of tags, as shown in the following code extract where a pair of tags are used to enumerate the threat ID 124:

$$< ThreatID > 124 < /ThreatID >$$

Alternatively a tag can also appear as a single entity, consisting of the < / > symbol and one or more variable declarations enclosed within the tag to enumerate values assigned to a variable using the = symbol, as shown in the following code extract where a tag describing a threat contains two variables enumerating the threat

ID and threat level:

$$< Threat\ ThreatID = \text{“}124\text{”}\ ThreatLevel = \text{“}High\text{”} / >$$

We now explain the tags and values relevant to our extraction process, using examples performed on a small set of data extracted from version *2.10* of the CWE core content, available at *https://cwe.mitre.org/data/* using a XSD file.

Weakness: this tag begins the description of a weakness in the core content of CWE.

- *“ID”* parameter defines the CWE ID of the weakness.
- *“Name”* parameter defines the full name of the weakness.
- *“Weakness_Abstraction”* parameter defines the the level of abstraction of the weakness.
- *“Status”* parameter defines if the weakness is currently usable.

For example the following code provides the weakness entry for CWE 119:

```
<Weakness ID="119"
    Name="Improper Restriction of Operations within the Bounds of a
    Memory Buffer"
    Weakness_Abstraction="Class"
    Status="Usable">
```

Relationship: this tag defines a relationship between the weakness and classes, bases, categories, chains, variants and views. With the following tags:

- *Relationship_Views* tag with *Relationship_View_ID* parameter determines the ID of the view.
- *Relationship_Chains* tag with *Relationship_Chain_ID* parameter determines the ID of the chain.
- *Relationship_Target_Form* tag determines the type of relationship.

- *Relationship_Nature* tag determines the nature of the relationship from the current weakness to a target weakness.
- *Relationship_Target_ID* tag determines the ID of the target weakness.

For example this extract of code defines the relationships between this CWE and other entries:

```
<Relationship>
    <Relationship_Views>
      <Relationship_View_ID>1000</Relationship_View_ID>
    </Relationship_Views>
    <Relationship_Chains>
      <Relationship_Chain_ID>680</Relationship_Chain_ID>
    </Relationship_Chains>
    <Relationship_Target_Form>Weakness</Relationship_Target_Form>
    <Relationship_Nature>CanPrecede</Relationship_Nature>
    <Relationship_Target_ID>119</Relationship_Target_ID>
    <!--Improper Restriction of Operations within the Bounds
    of a Memory Buffer--></Relationship>
```

Applicable_Platforms: this tag defines the platforms applicable with the weakness.

- *Languages* tag with *Prevalence* parameter defines the prevalence of languages affected.
- *Languages* tag with *Language_Name* parameter defines the name of the language affected.
- *Language_Class* tag with *Language_Class_Description* parameter defines describes general classes of languages affected by the weakness.

For example this extract of code describes the applicable platforms:

```
<Applicable_Platforms>
    <Languages>
       <Language Prevalence="Often" Language_Name="C"/>
       <Language Prevalence="Often" Language_Name="C++"/>
       <Language Language_Name="Assembly"/>
       <Language_Class Language_Class_Description="Languages without
       memory management support"/>
    </Languages>
```

Common_Consequences: this tag describes the security property goals affected by the weakness.

- *Consequence_Scope* tag indicates the scope of the security property affected by the weakness.
- *Consequence_Technical_Impact* tag indicates the technical impact of the weakness.

For example this block of code describes the common consequences on security properties:

```
<Common_Consequences>
    <Common_Consequence>
       <Consequence_Scope>Integrity</Consequence_Scope>
       <Consequence_Scope>Confidentiality</Consequence_Scope>
       <Consequence_Scope>Availability</Consequence_Scope>
       <Consequence_Technical_Impact>Execute unauthorised code or
        commands</Consequence_Technical_Impact>
       <Consequence_Technical_Impact>Modify memory</Consequence_
       Technical_Impact>
```

5.2.2 Common Vulnerabilities and Exposures

The Common Vulnerabilities and Exposures(CVE) is a "dictionary of common names (i.e., CVE Identifiers) for publicly known cybersecurity vulnerabilities". The purpose of CVE is to provide an industry standard for vulnerability and exposure

names. Each CVE identifier includes an unique CVE identifier number i.e. "CVE-2017-0038", a brief description of the security vulnerability or exposure and any pertinent references (i.e., vulnerability reports and advisories). CVE is an international cybersecurity community effort, launched in 1999 by the MITRE Corporation who are responsible for maintaining CVE and the public website. In comparison to the CWE which focuses on underlying vulnerabilities, the CVE describes specific instances within a product or system.

In order to extract information associated with vulnerabilities, we define the following fields to transfer entries from CVE to our knowledge-base in a specific format to support automated analysis. The field *"CVE-ID"* provides an unique identifier for each specific vulnerability, through the format "CVE-" followed by a four digit year indicating which year the CVE-ID was created or allocated, followed by a four digit ID. The field *"Vendor"* indicates the actor that provides the product or service. The field "Product" provides the name of the product or service. The field *"Version"* indicates the version of the product or service, where the value is a list containing zero or more values.

For example, given the XML file *"allitems-cvrf-year-2017.xml"* containing CVE entries from 2017, the following tags are examined to extract data from the file:
Vulnerability: The relevant information we are interested in are enclosed in the *"vulnerability"* tag. The *"title"* tag is a vulnerability entry corresponding to our concept of a vulnerability instance in the model, for example *"CVE-2009-4344"*. The *"note"* tag provides textual information of the vulnerability, where the *"note type"* tag of type *"description"* indicates the description, version and attack details, for example *"Cross-site scripting (XSS) vulnerability in the ZID Linkliste (zid_linklist) extension 1.0.0 for TYPO3 allows remote attackers to inject arbitrary web script or HTML via unspecified vectors."*. The *"note type"* tag of type *"other"* with the tag "title" with "published" indicates the date the vulnerability was published, for example "2009-12-17". The *"CVE"* tag such as *"CVE-2009-0057"* is used to provide the CVE entry of the vulnerability, with the *"references"* tag enclosing references describing mitigation strategies from external sources, for example an external link to the URL *"http://typo3.org/teams/security/security-bulletins/typo3-sa-2009-020"* with the *"description"* tag *"CONFIRM: http://typo3.org/teams/security/security-bulletins/typo3-sa-2009-020"*.

5.2.3 National Vulnerability Database

The National Vulnerabilities Database(NVD) is an U.S. government repository of standards-based vulnerability management data. NVD is a product of the NIST Computer Security Division, Information Technology Laboratory and is sponsored by the Department of Homeland Security's National Cyber Security Division. The NVD is a superset of CVE and thus is synchronised such that any updates to CVE appear immediately in the NVD. NVD integrates information from CWE into the scoring of CVE vulnerabilities by providing a cross section of the overall CWE structure. NVD analysts score CVEs using CWEs from different levels of the hierarchical structure.

We use the NVD CVE XML 2.0 schema in our work because this version includes Common Weakness Enumeration (CWE) identifier(s) for each CVE, allowing us to map CWE-IDs to the MITRE CWE List. The CVE provides a list of properties which subsequently identifies specific vulnerabilities given an asset with the matching properties. The CWE provides a list of mitigation strategies which maps to our concepts of the security objective and security mechanism given a specific instance of a vulnerability. Thus by mapping the information in the CVE to the CWE based on cross-referencing entries in the NVD, we are able to extract the vulnerability, properties of resources affected by said vulnerability, mitigation strategies for the vulnerability as security objectives and security mechanisms and the consequences on security properties, through security constraints.

For example the following wrapper tag in version 2.0 of the NVD XML Schema describes the vulnerable version of the product, following the same tag and value definitions described in Section 5.2.1:

```
<vuln:product>cpe:/o:apple:mac_os_x:10.12.0</vuln:product>
```

Additional information is provided in version 1.2.1 of the NVD XML Schema:

```
<vuln_soft>
    <prod name="mac_os_x" vendor="apple">
      <vers num="10.12.0" prev="1"/>
    </prod>
  </vuln_soft>
```

In this case the *"name"* attribute in the *"prod"* wrapper corresponds to the *Product* property in our resource concept, the *"vendor"* attribute corresponds to our *Vendor* property and the *"num"* attribute in the *"vers"* wrapper corresponds to our *Version* property. The *"prev"* attribute indicates that versions previous to this version number are also affected by this vulnerability, which is a Boolean. Therefore given a resource instance with the following properties: $Vendor$: $apple, Product$: $mac\_os\_x, Version$: 10.12.0, we are able to detect and assign the vulnerability *"CVE-2016-4671"*(and twenty three other matching records) via consulting the NVD feed.

Using the Common Vulnerability Scoring System (CVSS) we support the usage of metrics to determine impact on security properties, specifically providing metrics for the CVSS Severity using version 3.0 or 2.0. In particular we capture the impact of vectors in confidentiality, integrity and availability, ranked from none, partial and complete. We then refer to CWE in order to model the threats, where several levels of abstraction can be defined. For example given the vulnerability *"CVE-2016-4671"*, consulting the weakness entry *CWE-787* in CWE defines at the high level the weakness class *"CWE-119: Improper Restriction of Operations within the Bounds of a Memory Buffer"* from the following block:

```
<Relationship_Target_Form>Weakness</Relationship_Target_Form>
<Relationship_Nature>ChildOf</Relationship_Nature>
<Relationship_Target_ID>119</Relationship_Target_ID>
```

At lower levels the variants *"CWE-121: Stack-based Buffer Overflow"* and *"CWE-122: Heap-based Buffer Overflow"*(This is found by searching for CWE-787 and tracing all instances of relationships with other entries, such as parent of or child of). For example finding a reference to *CWE-787* and the relationship nature *ChildOf*, then finding the weakness id of the entry:

```
<Relationship_Target_Form>Weakness</Relationship_Target_Form>
<Relationship_Nature>ChildOf</Relationship_Nature>
<Relationship_Target_ID>787</Relationship_Target_ID>
```

After identifying the vulnerability, threat and potential impact on security constraints, we define the mitigation strategy by generating security objectives and

mechanisms through consulting the CWE. In some cases a vulnerability has to be decomposed to child or parent vulnerabilities, where details for mitigation are provided. For example given the weakness entry *CWE-787*, we examine the mitigation strategies for its parent class *CWE-119* and variants *CWE-121*.

We categorise the level of abstraction according to the phase of mitigation, thus modelling security objectives as higher level concepts and security mechanisms for lower levels of abstraction. The level of categories can also be defined by the developer, though we will provide a default listing as follows: for phase requirements, architecture and design the mitigation falls under security objectives. For the phase Build and Compilation, Implementation and Operation, these are security mechanisms.

For example given *CWE-119*, the Mitigation ID *MIT-3* in phase *Requirements* and strategy *Language Selection* defines the textual description: *"Use a language that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, many languages that perform their own memory management, such as Java and Perl, are not subject to buffer overflows. Other languages, such as Ada and C#, typically provide overflow protection, but the protection can be disabled by the programmer. Be wary that a language's interface to native code may still be subject to overflows, even if the language itself is theoretically safe."*. Thus this generates a security objective instance named *MIT-3: Language Selection* with the description following the text block.

5.2.4 Cloud Control Matrix

We extract information from the top ten threats of 2013 defined by CSA (Los et al., 2013), specifically the threat name, control IDs from *CCM v3.0.1* (Saxena, 2013), applicable service models, CSA security guidance reference and threat analysis (STRIDE).

We then align each threat with recommendations from the CCM, in order to generate a list of applicable scopes and security objectives. For example given the threat **Insecure Interfaces and APIs,** the applicable security properties is confidentiality, integrity and authenticity (Yahya, Walters & Wills, 2015). The extracted list of recommended controls are as follows:

AIS-01: Application & Interface Security – Application Security

- Architectural relevance: compute, storage, app, data
- Cloud service model applicability: SPI
- Security objective: Applications and programming interfaces (APIs) shall be designed, developed, deployed, and tested in accordance with leading industry standards (e.g., OWASP for web applications) and adhere to applicable legal, statutory, or regulatory compliance obligations.
- Supplier relationship: CSP

AIS-04: Application & Interface Security – Data Security/Integrity

- Architectural relevance: network, compute, storage, app, data
- Cloud service model applicability: SPI
- Security objective: Policies and procedures shall be established and maintained in support of data security to include (confidentiality, integrity, and availability) across multiple system interfaces, jurisdictions, and business functions to prevent improper disclosure, alteration, or destruction.
- Supplier relationship: CSP

IAM-08: Identity & Access Management – Trusted Sources

- Architectural relevance: data
- Cloud service model applicability: SP
- Security objective: Policies and procedures are established for permissible storage and access of identities used for authentication to ensure identities are only accessible based on rules of least privilege and replication limitation only to users explicitly defined as business necessary.
- Supplier relationship: CSP, CU

IAM-09: Identity & Access Management – User Access Authorisation

- Architectural relevance: network, compute, storage, app, data
- Cloud service model applicability: SPI
- Security objective: Provisioning user access (e.g., employees, contractors, customers (tenants), business partners, and/or supplier relationships) to data and organisationally-owned or managed (physical and virtual) applications, infrastructure systems, and network components shall be authorised by the organisation's management prior to access being granted and appropriately restricted as per established policies and procedures. Upon request, provider shall inform customer (tenant) of this user access, especially if customer (tenant) data is used as part the service and/or customer (tenant) has some shared responsibility over implementation of control.
- Supplier relationship: CSP, CU

In this case the architectural relevance applies to the virtual resources of the same type, for example a virtual resource instance with the type storage will match the relevant control i.e. AIS-01, AIS-04 and IAM-09. The cloud service model applicability will search for if one or more cloud services with the *requires* relationship and listed service models in the property are associated with the resource instance. The security objective generates a security objective instance with the details linked to one or more security constraints with the security properties associated with the threat. Finally the supplier relationship indicates the actors responsible for achieving the security objective.

So far we have defined the formal analysis concepts based on the cloud modelling language in Section 5.1 and in this section we introduced how we extract information from expert databases to enhance our automated reasoning support. In the next section we define three analysis techniques using our formal analysis concepts and how the information from the expert data-bases are utilised in the analysis.

5.3 Cloud Analysis Techniques

In this section we define security analysis techniques to support the developer through the process of identifying threats and vulnerabilities in a cloud environment,

proposing alternative mitigation measures and validating cloud security needs. For example the threat analysis identifies any threats impacting cloud assets, where the developer is able to determine the impact of a threat in terms of the value metric associated with the targeted assets. Following the security mitigation analysis, the developer is able to determine the cost metric associated with implementing security measures, such as any security mechanisms required mitigate threats or when validating the satisfaction of security properties against user-defined rules.

Figure 5.6 illustrates the three supported cloud analysis techniques, with the input and output of each analysis technique indicating which sub-set of the cloud environment model(*CEM*) is used as input and updated respectively. The *CEM* is used as input for all three analysis techniques, where the *CEM* describes the system-under-design using the instance syntax introduced previously in Chapter 3. The order of the analysis is interchangeable depending on the developer needs and richness of the cloud environment model The analysis techniques can be performed independently, therefore the developer is able to perform analysis techniques in the order of their choosing. For example the developer can perform the security mitigation analysis on a cloud environment model containing pre-existing threats and vulnerabilities. On the other hand a cloud environment model with an empty resource knowledge base i.e. no resources in the system, will not produce any vulnerabilities or threats because there is no resource to analyse.

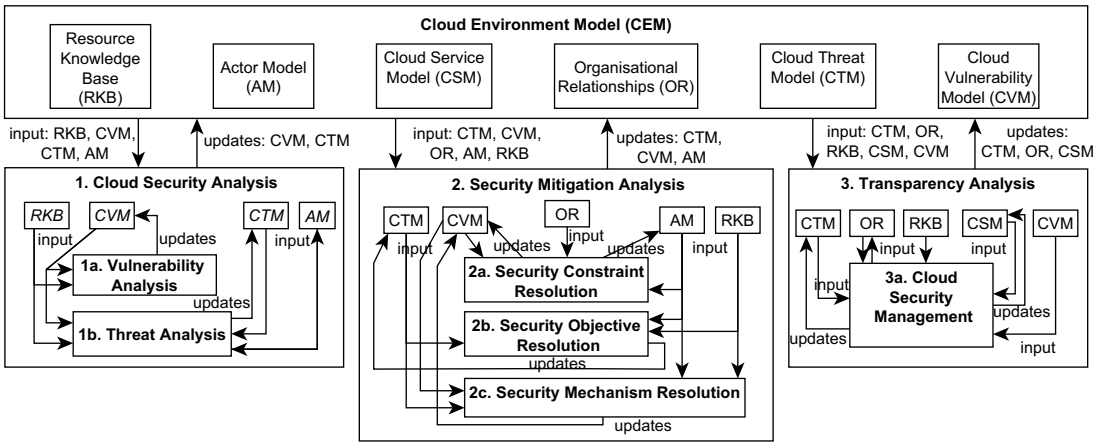

Figure 5.6: Process illustrating three types of cloud analysis techniques with inputs from and updates to the cloud environment model.

We now summarise each analysis, describing the specific details of each technique and how this contribute towards addressing the research question RQ3.

- **Cloud Security Analysis 1a. Vulnerability analysis:** We identify vulnerabilities affecting resources in the system-under-design, represented through the cloud environment model(CEM), by consulting the resource knowledge base(RKB). We refine the CEM by enumerating specific vulnerabilities which affects resources in the system, updating the cloud vulnerability model(CVM). Thus the CVM contains information on which resources are exploitable and the specific vulnerabilities affecting the system-under-design.

- **Cloud Security Analysis 1b. Threat analysis:** We identify threats which exploit vulnerabilities in the system. According to the vulnerabilities exploited, we refine the CEM to enumerate in the cloud threat model(CTM), which goals or resources are impacted by specific threats. Thus the CTM provides information on which resources or goals are at risk as a result of vulnerabilities identified in the CVM, and the specific threat exploiting the associated vulnerabilities which impacts the security properties of resources of goals in the system. The actor model(AM) identifies any malicious actors posing a threat on the system.

- **Security Mitigation Analysis 2a Security constraint resolution:** We identify security constraints by consulting the organisation relationships(OR), given an AM, in order to mitigate threats in the system. The purpose of a security constraint is to address the specific security property compromised by threats identified in the CVM. We consult the CEM and update the AM to show security constraints placed on the system as a result of identified threats. Thus the AM provides information on what security constraints or security needs need to be satisfied, in order to mitigate threats on the system-under-design.

- **Security Mitigation Analysis 2b. Security objective resolution:** We identify security objectives by consulting the CTM, in order to determine if all the security constraints placed on the system are satisfied. We refine the CEM to enumerate in the CTM security objectives and which security properties

they satisfy in relation to security constraints found in the *AM*. Thus the *CTM* advises the developer of the specific security objectives which should be implemented, in order to address the security constraints placed on the system-under-design.

- **Security Mitigation Analysis 2c. Security mechanism resolution:** We identify security mechanisms by consulting the *CVM* in order to protect against the exploitation of vulnerabilities in the system-under-design. Security mechanisms are implemented through security objectives, thus the *CEM* is refined through the *CVM* and *CTM* to enumerate how vulnerabilities are addressed by security mechanisms and which security objectives implements the defined countermeasures respectively.

- **Transparency Analysis 3a. Cloud Security Management** This analysis identifies the roles of cloud actors responsible for the management or ownership of physical and virtual assets, by consulting the *AM* and cross-referencing the organisational relationships(*OR*) with resources and goals found in the *RKB*. The identified relationships enumerating these roles are reflected in the *OR* and updates the *CEM* with any modifications made. The cloud actors responsible for managing and implementing security measures are also identified by consulting the *CVM*, *CTM* and *AM*. This analysis allows developers to summarise the delegation of responsibilities in order to ensure the cloud security requirements of the system is satisfied, updating the *CTM* and *CSM*.

The following subsections describes the steps of each stage in further detail, providing a systematic process for developers to carry out cloud security analysis. Note that the notation of the process is described using the instance syntax defined earlier in Section 5.1, where capital letters in italic indicates variables holding values i.e. *X, Y*, and the equals sign is the assignment operator i.e. $X{=}Y$ assigns the value in the variable Y to X.

5.3.1 Cloud Security Analysis

The first analysis takes as input a cloud model with the minimal concepts consisting of goals, actors and resources as shown in Figure 5.7. Goals are required in order to

capture the general needs of the system. Cloud services are a specialisation of the goal concept, in order to describe the cloud computing needs of the system-under-design and stakeholders involved. Resources describe the components the system and processes require in order to fulfil their needs, which includes specialisations of the resource concept describing cloud computing assets; virtual resource representing intangible resources, infrastructure node representing tangible resources and physical infrastructure representing boundary-specific components in scope of the system. This analysis consists of two techniques; the vulnerability analysis and threat analysis.

In order to map security knowledge to our cloud models, vulnerability, threat and mitigation information is extracted from the NVD CVE XML 2.0 schema. We define NVD as a set and say that a quadruple *(cvename, product, vendor, version)* is in NVD if a NVD entry describes *cvename* in the *entry* element with *name* as the attribute, *product* in the *prod* element with *product* as the *name* attribute, *vendor* as the *vendor* attribute and *version* in the *vers* element with *version* as the *num* attribute.

Vulnerability Analysis

Given a cloud environment model *CEM*, the vulnerability analysis examines the *RKB* of the *CEM* and searches the properties of assets within *RKB*. Specifically this analysis examines the properties of the set virtual resources *VR*, the set physical infrastructure *PI*, the set infrastructure node *IN* and their relationships *VPIR* in the *RKB*. Vulnerabilities are identified based on security knowledge and the *CVM* is updated. The following definition describes the procedure in order to perform the vulnerability analysis, where the *NVD* described in Section 5.2.3 and Section 5.3.1 is used to identify and generate vulnerabilities affecting assets in a cloud model:

Definition 5.3.1. Vulnerability analysis For each (product, version, vendor) in a virtual resource, physical infrastructure or infrastructure node of *RKB* if (product, version, vendor) is in *NVD* then update the *CVM* with a new instance of the vulnerability concept with the *affects* relationship to the virtual resource, physical infrastructure or infrastructure node. The new vulnerability instance has the description *cvename* where *cvename* is the matching NVD entry given the quadruple (cvename,

product, vendor, version).

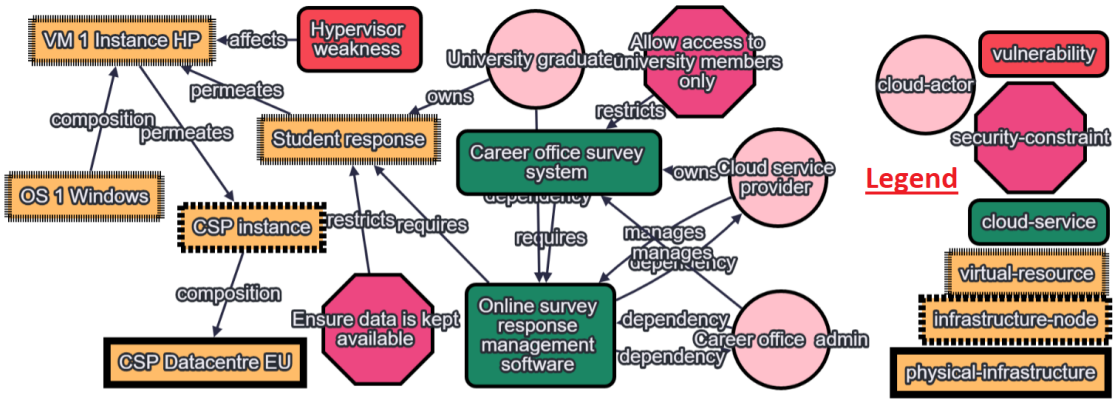

Figure 5.7: A cloud model as input for the vulnerability analysis.

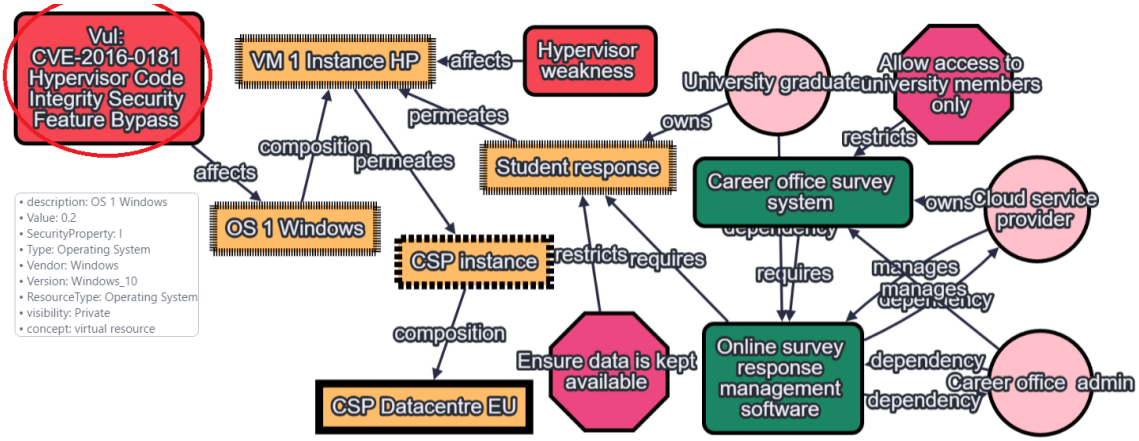

Figure 5.8: A new vulnerability is identified and generated from the vulnerability analysis.

For example given the cloud environment model shown in Figure 5.7, there are five entities in the *RKB* of the *CEM*; *VR1 Student response, VM 1 Instance HP, CSP instance, CSP Datacentre and OS 1 Windows*. The *CVM* has one existing vulnerability entry, enumerated as: *CVM(VM_1_Instance_HP,Hypervisor_weakness, affects(VM_1_Instance_HP,Hypervisor_weakness))*. When performing the vulnerability analysis, the properties in *VR1 Student response, VM 1 Instance HP, CSP instance and CSP Datacentre* do not match any entries in *NVD*. However the virtual resource *OS 1 Windows*, which is enumerated as *OS_1_ Windows(Windows,Microsoft,*

Windows_10), matches an entry in the *NVD* represented as a quadruple *(CVE-2016-0181,Windows,Microsoft,Windows_10)*. Therefore a new instance of the vulnerability concept was added to the *CVM* with the description *CVE-2016-0181* and the relationship *affects(CVE-2016-0181,OS_1_ Windows)*. Figure 5.8 illustrates the new vulnerability in the cloud model, after performing the vulnerability analysis. The *CVM* is now enumerated as: *CVM((VM_1_Instance_HP,OS_1_ Windows),(Hypervisor_weakness,CVE-2016-0181),(affects(VM_1_Instance_HP, Hypervisor_weakness),affects(CVE-2016-0181,OS_1_ Windows)))*

Threat Analysis

The threat analysis is the second analysis technique in the cloud security analysis, taking as input a *CEM*. Given a set of vulnerabilities in the *CVM*, these vulnerabilities can be exploited through threats posed by malicious actors in the *AM* to compromise the confidentiality, integrity or availability of resources and processes in the system. Therefore this analysis examines the *CVM* for vulnerabilities and existing threats in the *CTM*, generating new threats in the *CTM* to indicate the type of security property each threat impacts.

Definition 5.3.2. Threat analysis For each instance of a vulnerability V in the *CVM* of the *CEM* with the relationship *affects(V,R)* or *affects(V,G)*, if the description of V is in *NVD* then create a new instance of a threat T in the *CTM* with the following two properties; (i) the description *"Impact on (C,I,A)"* where (C,I,A) is a triple containing values from the corresponding *NVD* entry with *CVSS_vector* in the *entry* element and *C,I,A* as the attributes, (ii) the security property (C,I,A) where (C,I,A) is a triple containing values from the *NVD* entry with *CVSS_vector* in the *entry* element and *C,I,A* as the attributes.

The new instance of the threat T has the following relationships; (i) the *exploits* relationship to the corresponding vulnerability as *exploits(T,V)*, (ii) the *impacts* relationship to the resource R or goal G as *impacts(T,R)* or *impacts(T,G)*.

The Common Vulnerability Scoring System(CVSS) is a free and open industry standard for quantifying the severity of security vulnerabilities. The CVSS defines the impact of an exploit on a system using three security properties: "The confidentiality (C) metric describes the impact on the confidentiality of data processed

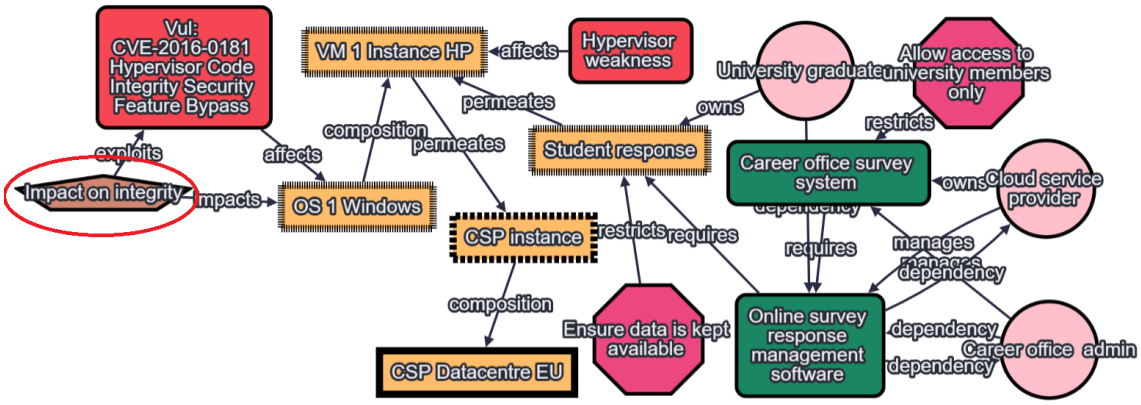

Figure 5.9: Identifying a threat exploiting a vulnerability and impacting a resource.

by the system.", "The Integrity (I) metric describes the impact on the integrity of the exploited system." and "The availability (A) metric describes the impact on the availability of the target system. Attacks that consume network bandwidth, processor cycles, memory or any other resources affect the availability of a system.". The current version of CVSS (CVSS v3.0) was released in June 2015.

Therefore we align the impact of exploiting a vulnerability in CVSS as the impact of a threat on a resource and the exploitation of a vulnerability in our cloud modelling language. Specifically we take the three security properties defined in CVSS; confidentiality, integrity and availability to correspond to values defining security needs in the security property of our threat concept. That is we map on an one-to-one basis given the exploitation of a vulnerability in CVSS, the impact on one or more of the three security properties corresponds to a threat enumerating these security properties.

For example Figure 5.9 illustrates the *exploits* relationship of a threat on a vulnerability, where the threat also *impacts* a resource. In this example the vulnerability entry with the CVE identifier *CVE-2016-0181* is found in the NVD vulnerability data feed, where the following CVSS information is extracted for that entry; $< entryCVSS_vector = "(C : N/I : P/A : N) >$. This extract from the CVSS indicates that there is no impact on the confidentiality property of resources affected by the vulnerability if exploited, partial impact on the integrity property of associated resources and no impact on the availability property. Therefore the threat with the name *"Impact on integrity"* with the set of security property $[I : P]$ is generated

in the model, indicating the threat of partially impacting integrity if the associated vulnerability is exploited.

5.3.2 Security Mitigation Analysis

The second type of analysis supported is the security mitigation analysis, where the purpose is to examine threats and vulnerabilities from the *CVM* and *CTM* in order to propose potential approaches for mitigation. This analysis takes a *CEM* as input, given that there exists vulnerabilities in the *CVM* and threats in the *CTM*.

Security Constraint Resolution

Security constraints represent the security needs of the system on assets or processes, where a security constraint needs to be satisfied in order to mitigate a threat. A security constraint mitigates a threat, indicated by the *mitigates(SC,T)* relationship in the *CTM*. A security constraint *SC* restricts a resource *R* and its specialisations or a goal *G* and the cloud service specialisation, indicated through the relationship *restrictsresource(SC,R,A)* and *restrictsggoal(SC,G,A)* in the *OR*. We define this technique to identify security constraints, create the *mitigates* relationship from a security constraint to a threat *T* and create the *restricts* relationship from a security constraint to a resource *R* or goal *G*. In the scope of this technique, *R* and *G* refers to the entry in the *impacts(T,R)* or *impacts(T,G)* relationships of the *CTM*. Therefore this analysis identifies the security needs of the system through security constraints, in order to address threats impacting resources and goals.

Definition 5.3.3. Security constraint resolution For each threat *T* in the *CTM* of the *CEM* where *mitigates(SC,T)* does not exist, then create an entry *mitigates(SC,T)* in the *CTM* where *SC* has the following properties; (i) the description *"Protect (C,I,A) of (R—G)"* where (C,I,A) is a triple in the security property of *T* and *R—G* is a single in the description of *R* or *G*, (ii) the security property *(C,I,A)* where (C,I,A) is a triple in the security property of *T*.

For each *impacts(T,R)* or *impacts(T,G)* in the *CTM* where there does not exist a *restricts(SC,R)* or *restricts(SC,G)*, create the following relationship with the security constraint *SC*; (i) *restricts(SC,R)* or *restricts(SC,G)* where *R* or *G* corresponds to the entry found in the *impacts(T,R)* or *impacts(T,G)* relationships of the *CTM*.

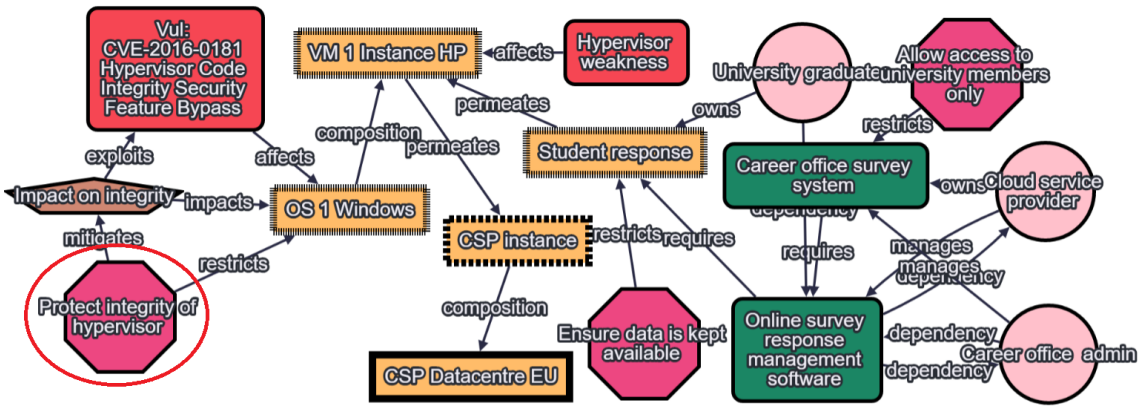

Figure 5.10: Creating a security constraint which mitigates a threat and restricts a resource.

For example Figure 5.10 shows a security constraint created in order to mitigate a threat, while restricting a resource. Following the threat analysis, we identify the threat named *Impact on integrity* in the cloud model, with the security property containing the set of values *I*. Taking the name of the threat as *Impact on integrity*, the target of the impacts relationship from the threat is identified as *OS 1 Windows*, which is assigned to *A*. The security constraint with the name *Protect integrity* is created, with the security property *I*. A mitigates relationship from the security constraint *Protect integrity* to the threat *Impact on integrity* is then created. Finally a restricts relationship from the security constraint *Protect integrity* to the resource *OS 1 Windows* is created.

After performing this analysis we have created a security constraint with the security property *I*, which has the mitigates relationship to the threat *Impact on integrity* and the restricts relationship to the resource *OS 1 Windows*. This security constraint therefore represents the security need which has to be satisfied to mitigate the indicated threat, where the security need is the security property integrity. The restricts relationship from the security constraint to the resource tells us which component in the system has security needs, specifically it tells us which security property needs to be satisfied.

Security Objective Resolution

The second part of the security mitigation analysis takes as input the *CTM*, *AM* and *RKB*. The purpose of this analysis is to cross-reference CVE and CWE entries between the NVD vulnerability data feed in Section 5.2.3 and the CWE core content database in Section 5.2.1, in order to identify and create security objective entries in the *CTM*. Therefore in this technique security objectives are identified to satisfy security constraints through the *satisfies* relationship, where properties of the security objectives are extracted from entries in the NVD and CWE.

Definition 5.3.4. Security objective resolution For each security constraint *SC* in the *CTM* of the *CEM* where *satisfies(SO,SC)* does not exist, then create an entry *satisfies(SO,SC)* in the *CTM* where the security objective *SO* has the following properties; (i) the description *(M)* where (M) is a single that exists in the CWE core content database with the *Mitigation_Strategy* element and *M* attribute corresponding to the CWE entry, (ii) the security property *(C,I,A)* where (C,I,A) is a triple that exists in the CWE core content database which describes (C,I,A) in the *Consequence_Scope* attribute of the *Common_Consequence* element given a corresponding CWE entry.

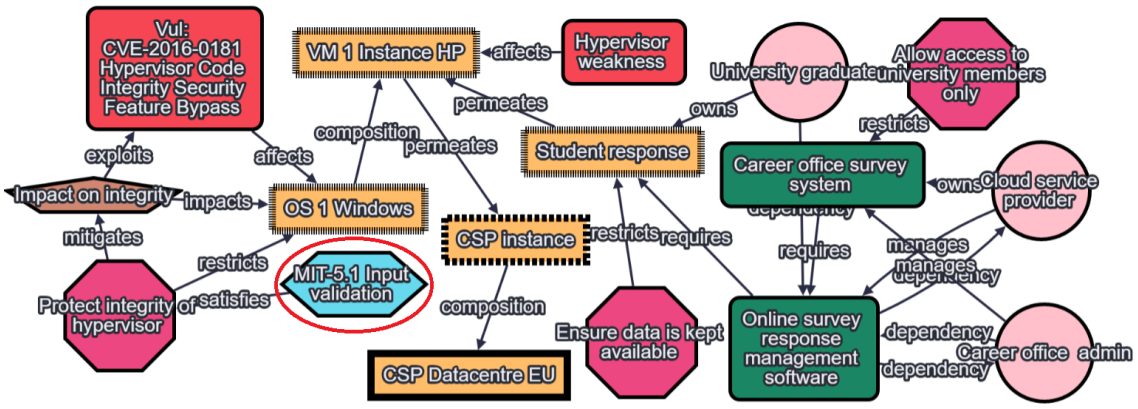

Figure 5.11: Creating a security objective which satisfies a security constraint.

Figure 5.11 shows the identification of a security objective as a result of consulting the CVE during the security objective resolution. In this case the security objective has the *securityProperty* of value *I* and the relationship *satisfies(MIT-*

5.1_Input_validation,Protect_integrity_of_hypervsior), which corresponds to the *securityProperty I* in *Protect_integrity_of_hypervsior*.

Security Mechanism Resolution

The final stage of the security mitigation analysis as input the *CVM*, *CTM*, *AM* and *RKB* from the *CEM*. The purpose of this technique is to identify the security mechanisms required to address vulnerabilities in the system-under-design. Specifically the CVE entry of vulnerabilities are cross-referenced through the NVD and CWE, where matching entries specify the mitigation method through the *protects(SM,V)* relationship.

Definition 5.3.5. Security mechanism resolution For each vulnerability *V* in the *CVM* of the *CEM* where *protects(SM,V)* does not exist, then create an entry *protects(SM,V)* in the *CVM* where the security mechanism *SM* has the following properties; (i) the description *(M)* where (M) is a single that exists in the CWE core content database with the *Mitigation_Description* element and *M* attribute corresponding to the CWE entry, (ii) the security property *(C,I,A)* where (C,I,A) is a triple that exists in the CWE core content database which describes (C,I,A) in the *Consequence_Scope* attribute of the *Common_Consequence* element given a corresponding CWE entry.

For each *protects(SM,V)* in the *CTM* where there does not exist an *implements(SO,SM)* relationship, create the following relationships with the security objective *SO*; (i) an *implements(SO,SM)* where the responsibility property of *implements* is the actor *A*.

Figure 5.12 shows a security mechanism which implements a security objective. In this case the security mechanism indicates how the security objective can be implemented, not the technical implementation instance or method. This limitation is due to the level of information provided in the security sources selected in our security analysis process. We argue that in the scope of security requirements engineering, providing exact technical implementation details during the early requirements stage is difficult due to the abstraction of components and concepts. That is during the design of the system-to-be, the developer does not posses enough

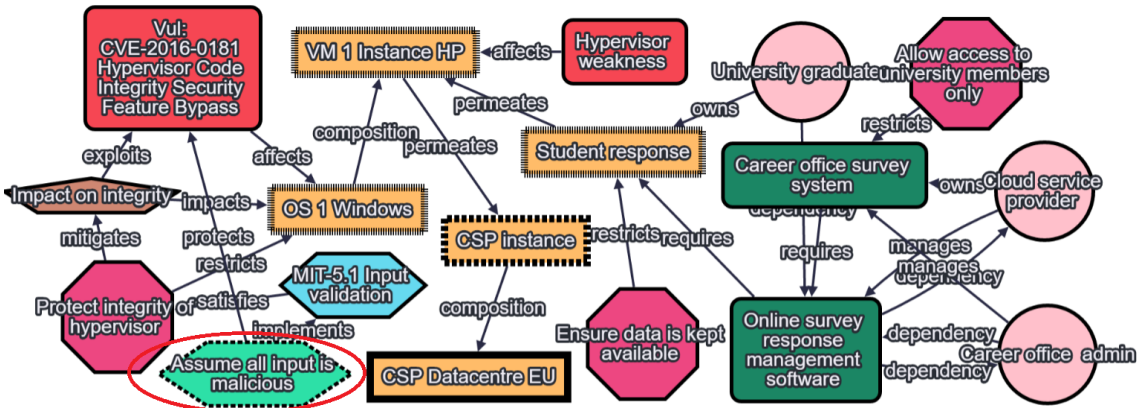

Figure 5.12: Creating a security mechanism which implements a security objective and protects a vulnerability.

information to be able to provide or make use of precise technical implementation level details.

5.3.3 Transparency Analysis

The analysis techniques defined previously have dealt with identifying security issues and solutions in a cloud computing environment, based on the properties of concepts such as cloud services and resources in the *CEM*. In order for developers to understand how cloud computing characteristics impacts the security needs in a cloud computing system, the responsibilities of cloud actors should be made clear. In this case there are several factors which determine a cloud actors scope of security responsibilities in a cloud computing environment. The primary concern is to determine the ownership and management of processes, data and infrastructure in a cloud computing system. Therefore by associating cloud actors with concepts in the system-under-design, the tractability of security responsibilities can be determined. As an example, given a cloud service with the SaaS cloud service model, the cloud service provider will be responsible for managing the infrastructure and applications required by the cloud service. Therefore the security implications in this example is that the cloud service provider will be responsible for addressing any security challenges that impact the infrastructure and applications associated with the cloud service.

Cloud Security Management

The purpose of this analysis is to examine the security needs of a cloud computing system, in order to identify cloud actors responsible for satisfying these security needs. The analysis takes as input the *CVM*, *CTM*, *OR*, *RKB* and *CSM*. Specifically the analysis examines the *affects*, *exploits*, *impacts* and *restricts* relationships in the *CVM*, *CTM* and *OR* to determine the security challenges imposed on goals and resources. The *protects*, *satisfies*, *implements* and *mitigates* relationships in the *CVM* and *CTM* determine how to address security challenges in the system-under-design. The *owns*, *manages* and *poses* relationships in the *AM* indicates the association between cloud actors and the processes, data and infrastructure in the cloud computing environment.

Definition 5.3.6. Cloud security management For each security constraint *SC* of *CEM*, if *SC* is in *OR* where *restrictsresource(SC,R,A)* or *restrictsgoal(SC,G,A)* and *requiresgoal(G,R)* exists then each *SC* has the following values appended to its *SecurityRequirement* property;

- append *"The cloud actor A is responsible for SC on G R: "* where the properties of *A*, *SC*, *G* and *R* is in the *restrictsresource(SC,R,A)* or *restrictsgoal(SC,G,A)* of *OR*
- for each *affects(V,R)*, *affects(V,G)* and *protects(SM,V)* in *CVM* where *R* or *G* is the same scope of *SC* then append *"Security mechanism SM to protect against vulnerability V "*
- for each *exploits(T,V)* and *mitigates(SC,T)* in *CTM* where *T* and *SC* is in the same scope of *SC* then append *"Security constraint SC mitigates the threat T "*
- for each *satisfies(SO,SC)* and *implements(SO,SM)* in the *CTM* where *SC* and *SM* is in the same scope of *SC* then append *"Security objective SO satisfies security constraint SC and implements security mechanism SM"*

Figure 5.13 illustrates a cloud environment model consisting of the following concepts: cloud actor *Career office admin* and *CSP1*, cloud service *Survey man-*

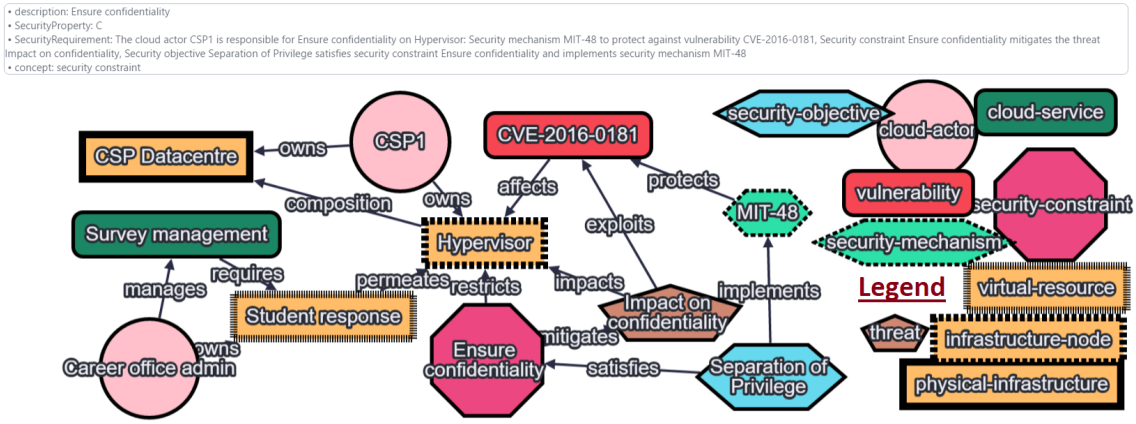

Figure 5.13: Cloud environment model showing the security needs of a system-under-design.

agement, virtual resource *Student response*, infrastructure node *Hypervisor*, physical infrastructure *CSP Datacentre*, vulnerability *CVE-2016-0181*, threat *Impact on confidentiality*, security constraint *Ensure confidentiality*, security objective *Separation of Privilege* and security mechanism *MIT-48*. These concepts are formally represented in

$$CEM = \langle RKB, AM, OR, CSM, CVM, CTM \rangle$$

where relationships such as *restrictsresource(Ensure confidentiality,Hypervisor,CSP1)* in the *OR*, *protects(MIT-48,CVE-2016-0181)* in the *CVM* and *satisfies(Separation of Privilege,Ensure confidentiality)* in the *CTM* are described. These relationships are examined when performing the cloud security management analysis, describing the security needs of the system. After performing this analysis, the security constraint *Ensure confidentiality* in *CEM* has the following *securityRequirement* property representing the security responsibility of a cloud actor:
"The cloud actor CSP1 is responsible for Ensure confidentiality on Hypervisor: Security mechanism MIT-48 to protect against vulnerability CVE-2016-0181, Security constraint Ensure confidentiality mitigates the threat Impact on confidentiality, Security objective Separation of Privilege satisfies security constraint Ensure confidentiality and implements security mechanism MIT-48".

In summary, after performing the analysis techniques in the security analysis process, a security-enhanced cloud model is produced by semi-automatically enriching the model with security concepts such as vulnerabilities, threats, security constraints, security objectives and security mechanisms. Specially the security constraints tells us which resources and goals in the system is impacted by threats or affected by vulnerabilities, and how they are restricted in terms of the security properties that need to be protected. The security objectives provide high level descriptions of what needs to be done in order to satisfy the security properties outlined in security constraints. The security mechanisms describe at a lower level, the method or technique which need to be carried out in order to implement security objectives.

While the granularity of the proposed concepts and mitigation strategy do not provide a direct way for developers to proceed to the implementation stage through coding guidelines, we argue that this is a limitation given the level of information provided from the security sources. Rather our security analysis approach guides the developer of cloud systems in analysing the goals and resources of a system, drawing from industry standard security sources to create and illustrate the relationships and concepts to help developers understand the security needs of the system-under-design.

5.4 Tool Support for the Secure Cloud Environment Framework

Apparatus Software Tool(ASTo) is an open source graphical security analysis tool for Internet of Things(IoT) networks, providing tool support for the Apparatus security framework (Mavropoulos, Mouratidis, Fish & Panaousis, 2017). In order to provide developers with semi-automated tool support when applying our secure cloud environment framework, our contribution is the SectroCloud module, which extends ASTo with cloud security requirements concepts. The automated reasoning support includes a graphical interface improving quality-of-life through task automation, the means to create visual models and perform semi-automated security analysis. The contributions of the SectroCloud module is as follows:

- Visualisation of the concepts and relationships defined from Section 3.2 of our

cloud modelling language

- Graphical representation of the conceptual cloud environment model, with three views focusing on the organisational, application and infrastructure level of concepts
- Provides semi-automated reasoning based on the three cloud analysis techniques defined in Section 5.3
- Quality-of-life utility and interface usability to support developers during the modelling process, such as concept and relationship filters, baseline security analysis techniques, data overview and model categorisation.

ASTo with the SectroCloud module is available for download from the authors online repository [2]. The instructions for installing the tool are provided in the online repository. We now document the features offered through the SectroCloud module, from a practitioners perspective.

5.4.1 SectroCloud Interface

ASTo is modular by design, which currently supports modules for modelling security in IoT networks and cloud computing environments. We focus on the interface for the SectroCloud module in this subsection. Figure 5.14 shows the main interface of the SectroCloud module. The left sidebar contains the following buttons from top to bottom:

- **Add node:** expands on hover to offer a list of all supported nodes. Clicking on an item in the list creates an instance of the selected concept in the model, with the corresponding visual representation.
- **Add edge:** expands on hover to offer a list of all supported edges. In order to add an edge, the source node should be selected first, followed by selecting the target node and clicking on an item in the add edge list. This generates an edge from the source node to the target node. Note that the connection item

[2]https://github.com/NOMNUDS/apparatus

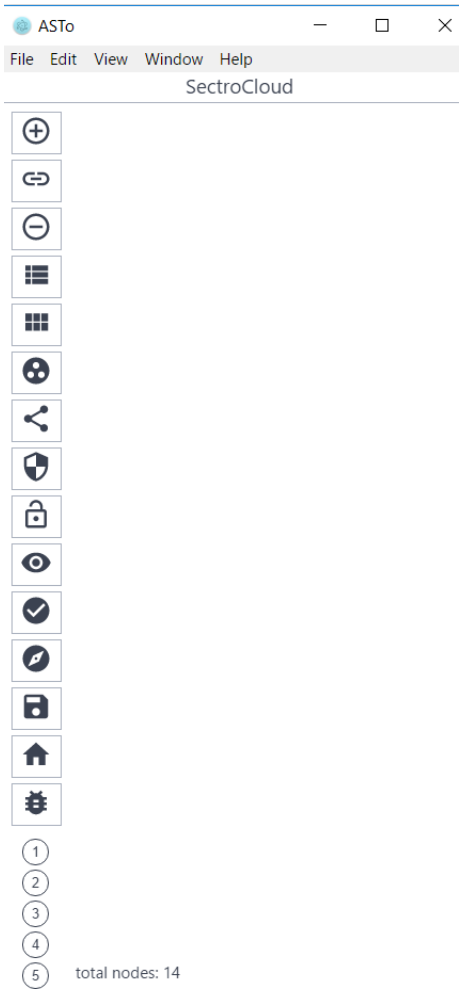

Figure 5.14: The main *SectroCloud* interface in ASTo.

in the list enumerates edges that are supported by default, while the other items in the list are specific edges to avoid ambiguous conflicts.

- **Remove:** removes the current node or edge that has been selected. Removing a node that is the target of multiple edges will also remove the corresponding edges.

- **Filter by concept:** visually highlights all nodes in the model corresponding to the type of concept selected, by applying a faded visual effect on all other nodes and edges.

- **Filter by group:** visually highlights all nodes in the model corresponding to the type of view selected, similar to the filter by concept feature.
- **Layout option:** offers a selection of model layouts such as breadth-first, circle and grid allowing developers to specify the spatial arrangement of the visual model.
- **Show neighbours:** highlights all edges and connected nodes of the selected node.
- **Threat verification:** performs a check to determine and visually highlight the total number of threats in the model and the number of threats that are mitigated through security constraints.
- **Vulnerability verification:** similar to the threat verification, this performs a check on the model to highlight and indicate the total number of vulnerabilities in the model and the number of mitigated vulnerabilities through security mechanisms.
- **Vulnerability identification:** examines the properties of resource and cloud service nodes, comparing specific attributes against known vulnerabilities stored in an external file and textually outputs the CVE details of identified vulnerabilities.
- **Model validation:** checks the validity of relationships and properties of nodes, according to a schema based on the cloud metamodel.
- **Overview:** generates a list summarising the total number of nodes in the model, the concepts used and the number of nodes instantiated from each concept.
- **Save model:** generates a json or js file containing data of the current working model.
- **Module selection:** allows the user to switch to a different module in ASTo.
- **Numerical buttons:** the first button hides the labels of nodes and edges, the second button shows the labels of nodes and edges, the third button shows

only the labels on nodes, the fourth button displays the id of nodes and the fifth button shows the description of nodes as the label.

5.4.2 Visualisation of Concepts and Relationships

In this subsection we present how the concepts, relationships and properties from our cloud metamodel are visualised using nodes and edges through the tool. Nodes are visually represented as a shape with a textual label, where each node corresponds to an instance of a concept. Edges are visually represented as a line connecting two edges together, where each edge represents a relationship. From this point on the term node is interchangeable with concept, and the term edge is interchangeable with relationship when referred to.

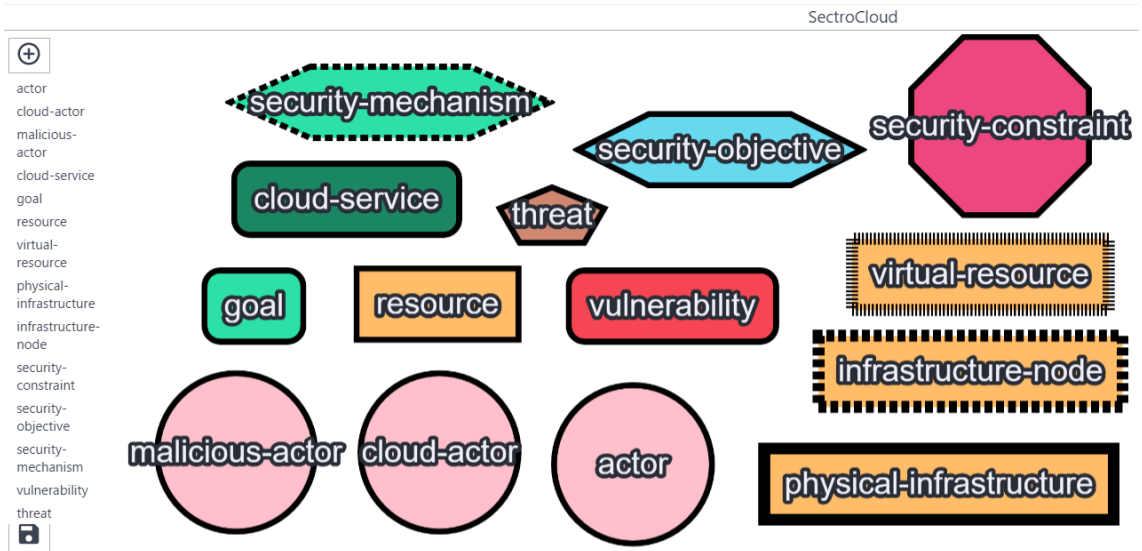

Figure 5.15: Adding concepts to the model in SectroCloud as nodes.

Figure 5.15 shows the options in the first button on the sidebar to add concepts to the model. All the concepts in the cloud metamodel in Section 3.2 can be visualised as nodes. The shape and colour of each concept is inspired by the visual theme employed in the Secure Tropos methodology.

Figure 5.16 shows the options in the second button on the sidebar to add relationships to existing nodes found on the model. All the relationships in the cloud metamodel are visualised as edges between nodes, with the label on the edge in-

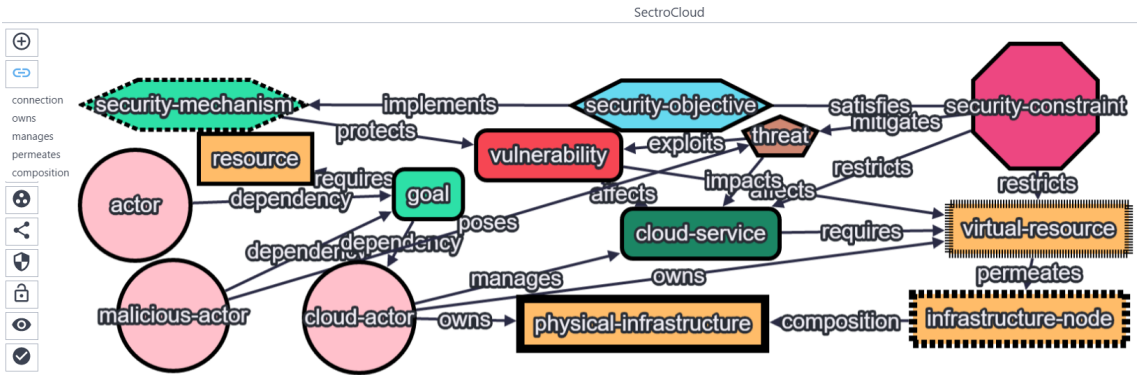

Figure 5.16: Adding relationships between nodes in SectroCloud as edges.

dicating the name of the relationship and the arrow indicating the direction of the relationship.

5.4.3 Properties of Concepts and Relationships

Recall in the cloud metamodel where concepts and relationships have properties specifying attributes unique to each instance.

Figure 5.17: Examining the properties of a node.

Figure 5.17 shows the properties of a cloud service instance, which is displayed in a pop-up tool-tip when hovering the cursor over a concept. Each property of the concept is indicated by a separate bullet-pointed line, where the property is described first, followed by a colon and the attribute of the property. For example the cloud-service concept has five properties, where the first property is the *description* with the *Amazon EC2* attribute.

Figure 5.18 shows the case where the properties of a relationship is displayed in the pop-up tool-tip, when hovering the cursor over a relationship. The format

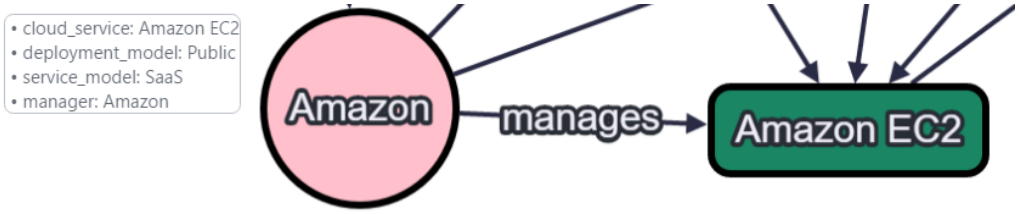

Figure 5.18: Examining the properties of an edge.

of the properties belonging to a relationship follows the same format as presented in properties belonging to concepts. For example the *manages* relationship from *Amazon* to *Amazon EC2* has four properties; the first property *cloud_service* with the attribute *Amazon EC2*, the second property *deployment_model* with the attribute *Public*, the third property *service_model* with the attribute *SaaS* and the forth property *manager* with the attribute *Amazon*.

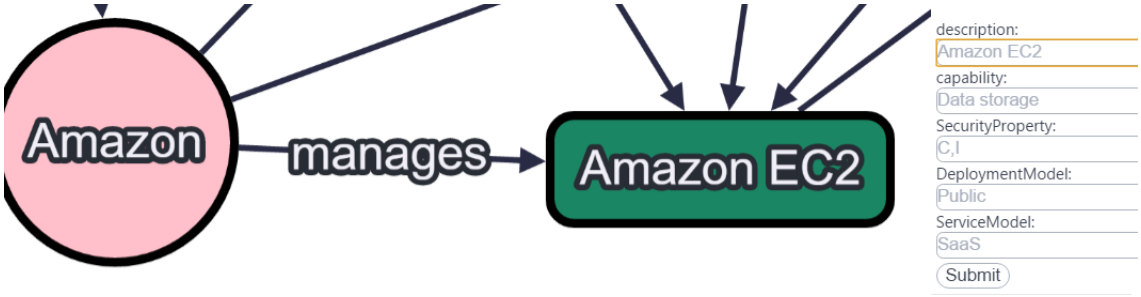

Figure 5.19: Editing the properties of a node.

Additionally the developer is able to edit the properties of nodes and edges in a model, by performing a right-click on the node or edge they wish to edit. Figure 5.19 shows an example where the developer has performed a right-click action on the *Amazon EC2* node, bringing up a form with the current properties of the selected item. The developer is then able to edit the attributes of each property and save changes by clicking the *Submit* button, committing the change to the model and closing the form.

5.4.4 Views and Filters

One of the main issues in modelling complex systems in a cloud environment is the scalability and visual presentation of data, when developers need to model cloud

systems with hundreds of nodes and relationships. With tool support, features such as the visualisation and grouping of concepts within a specific scope helps developers to uncouple complex models and understand the workings of the system under design. In this subsection we present features of SectroCloud which presents data within a specific scope.

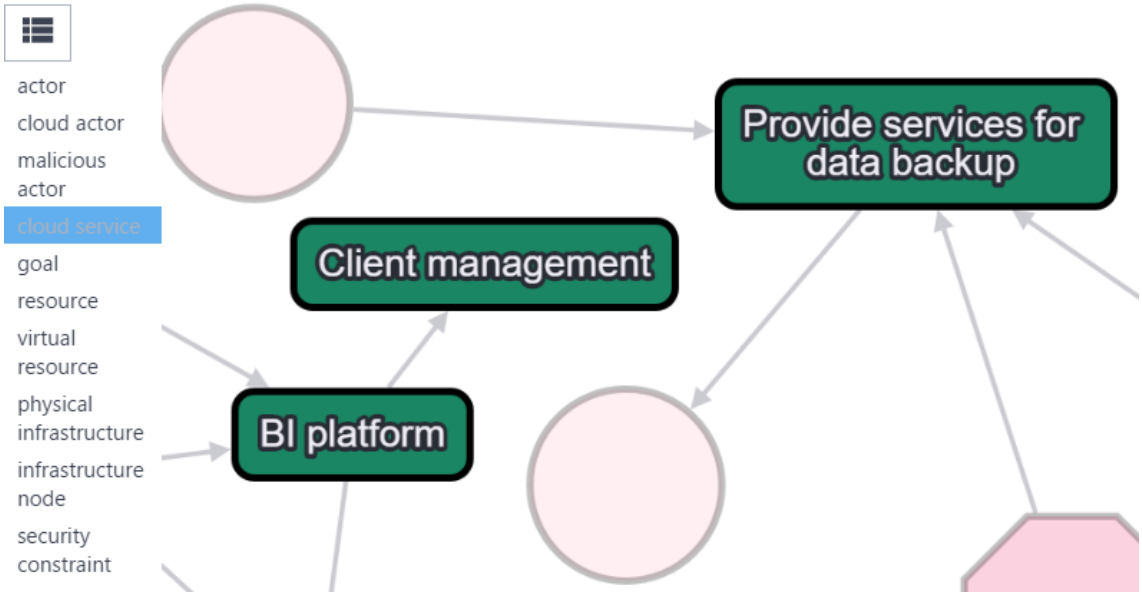

Figure 5.20: Filtering the model using the cloud-service concept.

The filter by concept feature allows developers to focus only on instances of the selected concept in the model. For example in Figure 5.20 the cloud-service concept is selected in the filter by concept menu, which visually focuses on all nodes belonging to the cloud-service concept while creating a faded out effect for the rest of the nodes. This feature is helpful for identifying and examining specific concepts in large and complex models.

The filter by group feature works similarly to the filter by concept feature, allowing the developer to focus on groups of concepts in a predetermined scope. For example in the schema we have defined the organisational, application, infrastructure, security and management views, which groups specific concepts from the cloud metamodel. In Figure 5.21 the application option has been selected from the *filter by group* menu, which focuses on the nodes that belong to the application view group, while excluded nodes are visually faded out. This feature allows developers to focus

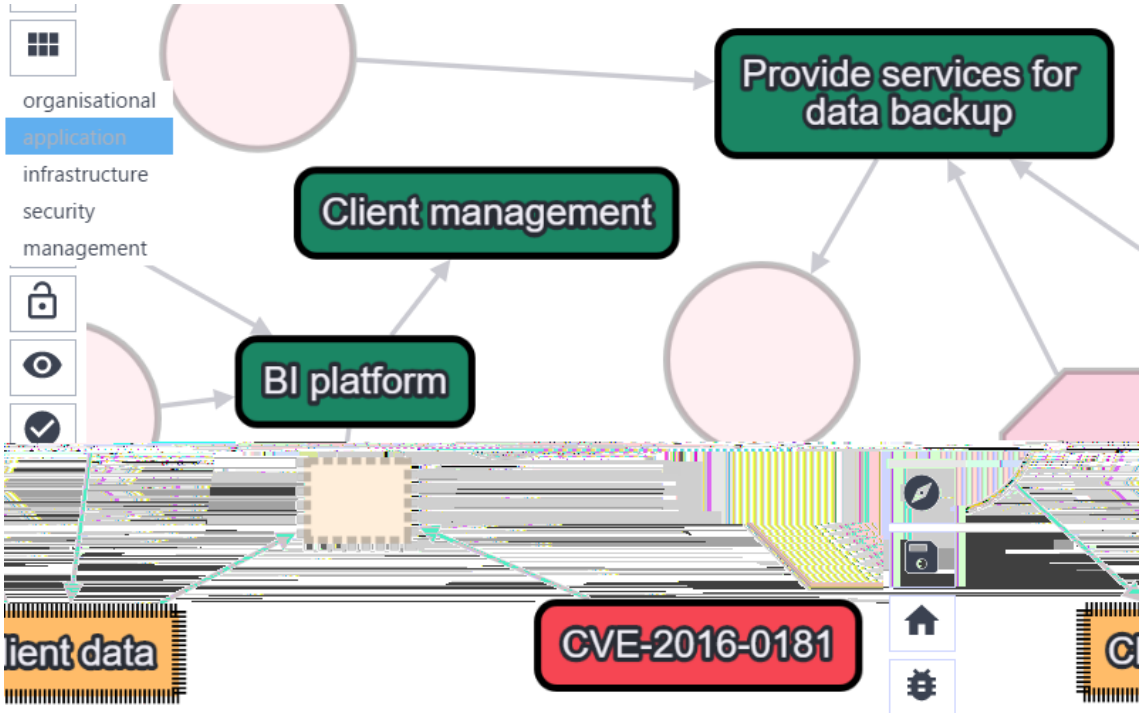

Figure 5.21: Filtering the model using the application view.

on a grouping of concepts in a specific scope, for example when they are interested in application-specific aspects of a cloud system.

5.5 Chapter summary

This chapter presents formal concepts to support semi-automating reasoning for cloud computing systems. To achieve this the instance syntax of the modelling language defines, without ambiguity, the information in the cloud environment model. Specifically the formal concepts define the following: a resource knowledge base specifying cloud assets in a cloud environment model, the actor model with intentional relationships and organisational relationships capturing the scope of cloud actor interactions, the cloud service model describing instances of the cloud service concept, the cloud vulnerability and cloud threat model enumerating security issues on assets and finally the formal definition of the cloud environment model. The procedure for extracting and mapping known vulnerabilities, threats and mitiga-

tion techniques from external security databases to our formal concepts is described through the security knowledge section. Three cloud analysis techniques have been presented, the cloud security analysis to identify vulnerabilities and threats, the security mitigation analysis to address identified security issues and the transparency analysis to highlight the security responsibility of cloud actors in the cloud system. To provide semi-automated reasoning, the functionality of ASTo and the proposed SectroCloud module was described. The tool provides a visual modelling approach for developers to create cloud environment models using the modelling language, process and analysis techniques proposed in this thesis.

Chapter 6

Evaluation of the Secure Cloud Environment Framework

In this chapter we evaluate the theoretical and practical components of the Secure Cloud Environment Framework, in order to prove that our proposed approach is a valid contribution in identifying, modelling and resolving cloud security requirements. We apply our framework through three use-case scenarios via self-evaluation and one case study. The objective of self-evaluation is to demonstrate the expressiveness of the modelling language, how the process is applied and the effectiveness of the semi-automated reasoning techniques. We now evaluate the cloud environment framework through three use-case scenarios and a case study.

6.1 Use-case Scenarios

Recall in Chapter 3 the cloud modelling language which defines the cloud computing concepts, relationships and properties for describing cloud computing environments. In order to demonstrate how the language is able to capture cloud computing characteristics and how effective the reasoning techniques are when identifying and specifying cloud security needs, we construct use-case scenarios based on real-world systems and security challenges. These scenarios capture the various security issues in cloud computing environments, from social interactions, to software configurations and the properties of physical components, in order to assist developers in

modelling and understanding cloud security requirements. In each of the following use-case scenarios, we apply the Secure Cloud Process from a practitioners perspective, with the SectroCloud providing semi-automated tool-support in graphical modelling and analysis.

Recall the three cloud service models according to the NIST definition;

- **Software as a Service(SaaS):** In the SaaS model, software applications are offered as a service to cloud users. The offered software applications can be hosted and delivered on-premise from cloud computing infrastructure owned by cloud service providers, or it may be deployed from infrastructure offered by third party vendors.

- **Platform as a Service(PaaS):** The PaaS model allows cloud users to build their own applications on top of the platform provided by a cloud service provider or third party vendor.

- **Infrastructure as a Service(IaaS):** Cloud users at the IaaS model can configure and manage virtual machines which are executed on hypervisors, which resides on physical infrastructure provided by cloud service providers or third party vendors.

We now demonstrate how our framework is applied within the scope of the three cloud service models, identifying different entry points into cloud systems through vulnerabilities and threats, how to address issues in security design through security mechanisms and security objectives and how to specify cloud security requirements through security constraints.

6.1.1 Software as a Service

In the SaaS cloud service model, the data of cloud users are stored on the data centre belonging to the cloud service provider, or on the cloud computing infrastructure of third party providers. The cloud user depends on the cloud service provider to ensure appropriate measures are taken to keep the users data and processes secure. In the case where the service provider leverages the cloud service from publicly available sources, the same physical infrastructure may be shared with multiple disparate

users and providers. Therefore in order to address the lack of control and knowledge over how the cloud users data is stored and secured in the SaaS model, cloud service providers need to provide assurance that data and processes entrusted to them are kept secure. Specifically developers should be able to identify the security needs of the cloud computing system, focusing on the security responsibilities of cloud service providers in multi-tenant environments. This relies on capturing ownership and management relationships between cloud actors and resources in the system.

Data Security

In order to discuss data security in SaaS cloud service model deployments, we will first construct an use-case loosely modelled on the media reports of an issue with BitDefender in 2015 (Fox-Brewster, 2015). Based on the use-case we demonstrate how our visual models could be used to mitigate common issues identified as the following vulnerabilities:

- Cross-site request forgery
- Insecure storage
- CVE-2011-1576
- CWE-119: Buffer Errors

This use-case is based on an incident in July 2015, where BitDefender, an anti-virus firm, had an undisclosed number of un-encrypted customer user-names and passwords stolen due to a security vulnerability in its public cloud application, which was hosted on Amazon Web Services(AWS). In this incident "DetoxRansome", the hacker responsible for the attack on the public cloud application, demanded a ransom of 15,000 US dollars in exchange for a portion of the customer user-names and passwords. As the specification and configuration of the infrastructure involved in this use-case isn't publicly disclosed, we model this scenario using high level concepts for the components of BitDefender and AWS. This approach was taken in order to demonstrate and propose a solution which identifies the security requirements that would have prevented such an attack. Figure 6.1 illustrates the cloud environment model of this scenario using the SectroCloud tool-support. In

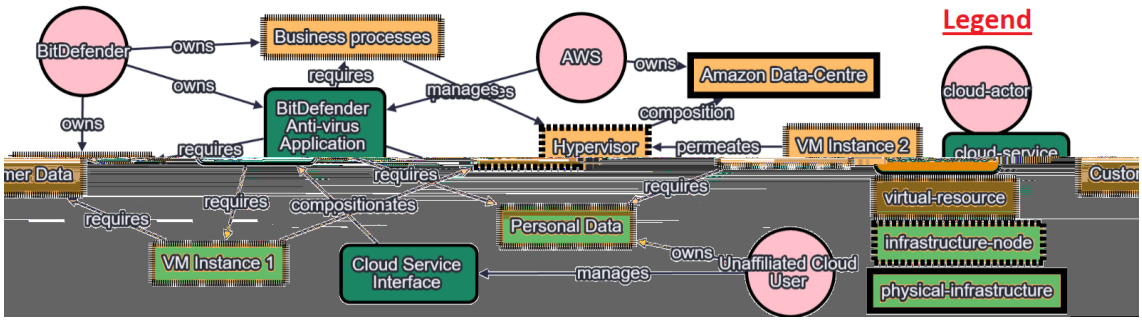

Figure 6.1: Cloud Environment model for the BitDefender use-case scenario.

the scope of the scenario, the cloud actor *"BitDefender"* has hosted their anti-virus cloud application on infrastructure provided by a third party provider, in this case *"AWS"*. The *"BitDefender Anti-virus Application"* cloud service in this scenario is owned by *"BitDefender"* and managed by the cloud service provider *"AWS"*. The *"BitDefender Anti-virus Application"* cloud service is composed of another cloud service, the *"Cloud Service Interface"* interface which users such as *"BitDefender"* and *Unaffiliated Cloud User* are able to access the *"BitDefender Anti-virus Application"* cloud service by providing a set of valid credentials. The *"BitDefender Anti-virus Application"* cloud service requires the virtual resource *"VM Instance 1"*, representing an instance of the service running on a virtual machine dedicated to the *"BitDefender"*. However the *"Customer Data"* and *"Business processes"* virtual resources permeates the same *"Hypervisor"* infrastructure node and physical infrastructure *"Amazon Data-Centre"* as the virtual resource *"Personal Data"* of another unaffiliated cloud user *"Tenant B"*.

In order to perform analysis on the BitDefender scenario shown in Figure 6.1, a summary of the *CEM* formally describing the system-under-design is provided to highlight key relationships and properties given that $CEM = \langle RKB,\ AM,\ OR,\ CSM,\ CVM,\ CTM \rangle$:
The $RKB = \langle \{Business\_processes, Customer\_Data, VM\_Instance\_1, Personal\_Data, VM\_Instance\_2\}, \{Hypervisor\}, \{Amazon\_Data\text{-}Centre\}, \{permeates(VM\_Instance\_1, Hypervisor),\ permeates(VM\_Instance\_2, Hypervisor),\ composition(Hypervisor, Amazon\_Data\text{-}Centre)\} \rangle$, the $AM = \langle BitDefender, \{BitDefender\_Anti\text{-}virus\_Application\}, \{Business\_processes, Customer\_Data\}, \{owns(BitDefender, Business\_processes),\ owns(BitDefender, Customer\_Data), \{CU\}\} \rangle$, the $CSM =$

$\langle BitDefender\_Anti\text{-}virus\_Application(Public,\{SaaS\},\{Business\_processes,Customer\_Data\})\rangle$. At this stage the *CVM* and *CTM* are empty because no security concepts such as threats or vulnerabilities have been identified.

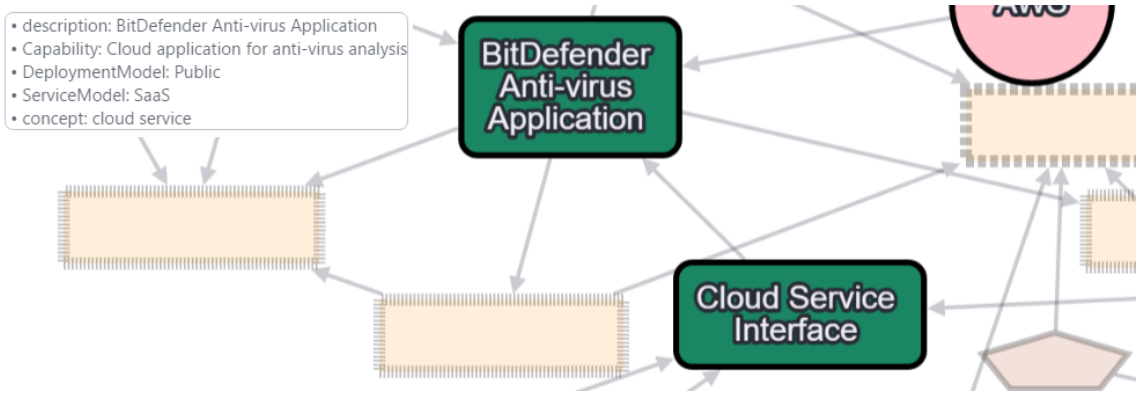

Figure 6.2: Organisational view of the cloud model.

The organisational view shown in Figure 6.2 highlights the cloud actors and the cloud services they manage or own. Here we examine the property of the cloud service *"BitDefender Anti-virus Application"*, where key properties include the *SaaS* service model and *public* deployment model. These properties determine the scope of security issues and management, focusing on public SaaS cloud service deployments.

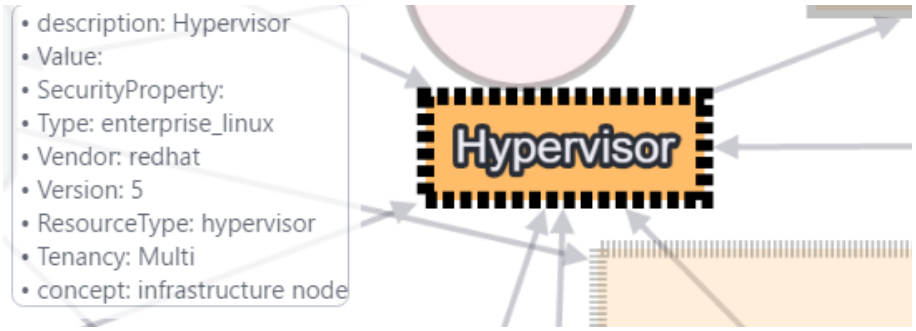

Figure 6.3: Properties of the *"Hypervisor"* infrastructure node instance.

We now describe the results after performing the semi-automated reasoning techniques on the cloud model, detailing how the physical, virtual and personnel security needs are transferred to the SaaS provider, *AWS*, when outsourcing applications from a traditional on-premise model to a cloud environment. Therefore the SaaS provider is responsible for ensuring security measures, such as access control policies and data

encryption techniques are implemented and enforced. In this case a malicious actor can exploit weaknesses at the physical and logical levels, gaining access to sensitive organisational data and processes. For example according to the Top 10 2013: Cross-site Request Forgery(CSRF) by OWASP (Wichers, 2013) the *"Cloud Service Interface"* is vulnerable to *"Cross-site request forgery"* if any links and forms lack an unpredictable cross-site request forgery token, which is exploited by the threat of *"Social Engineering"* when the attacker is able to trick the authenticated victim into submitting a forged request.

By examining the properties of *"Hypervisor"*, the type, vendor and version attributes shown in Figure 6.3 is used to identify matching vulnerabilities in expert databases from sources such as the National Vulnerability Database(NVD). In this case the properties type *"enterprise_linux"*, vendor *"redhat"* and version *"5"* of "Hypervisor" matches the vulnerability *"CVE-2011-1576"* from the NVD. The "Vulnerability identification" analysis generates an instance of a vulnerability in the cloud model with the description *"CVE-2011-1576"* and relationship *"affecting"* the "Hypervisor" node. In addition according to the entry data in the NVD, the *"CVE-2011-1576"* vulnerability is associated to the vulnerability type *"CWE-119: Buffer Errors"*, which is generated in the cloud model as *"CWE-119: Buffer Errors"* and is composed of the originally identified vulnerability *"CVE-2011-1576"*. The *"Malware"* threat exploits the "Insecure storage" vulnerability affecting the *"Hypervisor"* infrastructure node.

After identifying the vulnerabilities and creating instances in the cloud model, the next step is to determine security mechanisms suitable for addressing these vulnerabilities. According to the secure cloud process, this is achieved either through personal expert knowledge on the part of the developer, or through tool-assisted analysis using security knowledge-bases for known mitigation strategies, such as the NVD. For example the developer can refer to the entry for Cross-site request forgery in the OWASP Top 10 Critical Web Application Security Risks (Wichers, 2013) and find the security mechanism "Use of unique token in a hidden field" to protect against the "Cross-site request forgery" vulnerability. The high level vulnerability "Insecure storage" can be addressed by cryptography using security mechanisms such as "Encryption" (Los et al., 2013) on passwords or data-sets. The security mechanism resolution also identifies two security mechanisms; *"Patch: RHSA-2011:0927-1"*

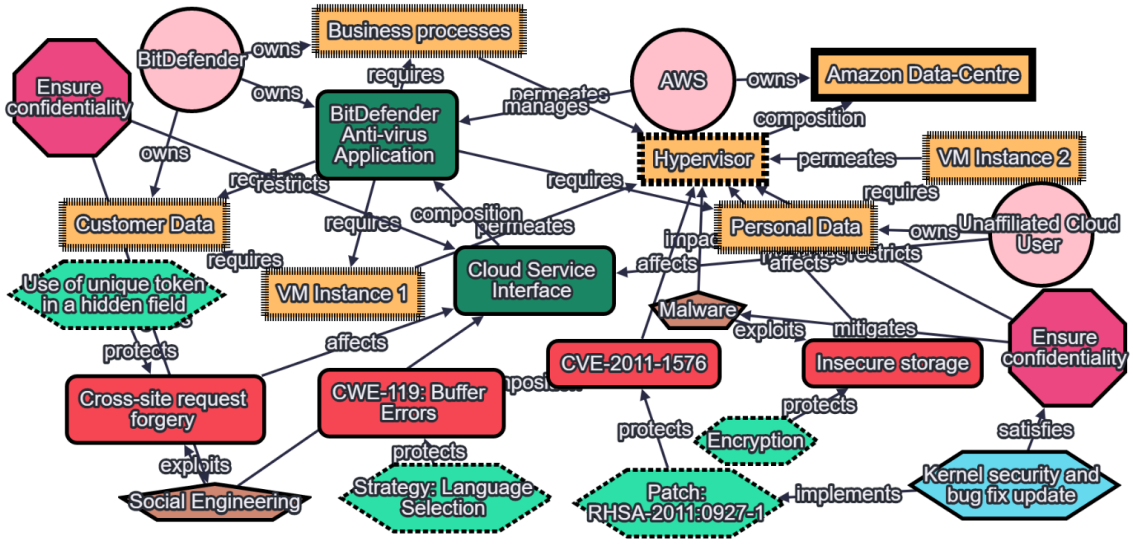

Figure 6.4: The cloud environment model after performing semi-automated reasoning techniques.

and *"Strategy: Language Selection"*, in order to address the *"CVE-2011-1576"* and *"CWE-119: Buffer Errors"* vulnerabilities respectively. These security mechanisms were identified and modelled based on the mitigation details described in the vulnerabilities entries, when referring to the expert knowledge in the NVD.

Figure 6.4 shows a visual representation of the cloud environment model, after performing the cloud security analysis, security mitigation analysis and the transparency analysis. The security responsibilities are enumerated through the two security constraints:

- *restrictsgoal(Ensure_confidentiality, Cloud_Service_Interface, BitDefender)* where *Ensure_confidentiality* has the *securityRequirements* property *"The cloud actor BitDefender is responsible for Ensure_confidentiality on Cloud_Service_Interface: Security mechanism Use_of_unique_token_in_a_hidden_field to protect against vulnerability Cross-site_request_forgery, Security constraint Ensure_confidentiality mitigates the threat Social_Engineering"*

- *restrictsresource(Ensure_confidentiality, Hypervisor, AWS)* where *Ensure_confidentiality* has the *securityRequirements* property *"The cloud actor AWS is responsible for Ensure_confidentiality on Hypervisor: Security mech-*

anism Strategy:_Language Selection to protect against vulnerability CWE-119: _Buffer_Errors, Security mechanism Patch:_RHSA-2011:0927-1 to protect against vulnerability CVE-2011-1576, Security mechanism Encryption to protect against vulnerability Insecure_storage, Security constraint Ensure_confidentiality mitigates the threat Malware, Security objective Kernel_security_and_bug_fix_update satisfies security constraint Ensure_confidentiality and implements security mechanism Patch:_RHSA-2011:0927-1"

In summary *BitDefender* could have prevented the attack described in this scenario, given that developers of the system is able to identify the security responsibilities in this scenario.

6.1.2 Platform as a Service

In the context of a PaaS service model, the data and processes of the cloud user needs to be obtained from their on-premise infrastructure, transmitted to the PaaS application for processing and stored on the infrastructure of the PaaS provider. This data transmission process is carried out through multiple network boundaries, from the cloud user to the cloud service provider. Therefore all data-flow over these networks need to be secured to prevent data-leakage. However the initial step is to identify the points at which data flows between different boundaries, to ensure appropriate measures are taken to address security according to the jurisdiction and identify the parties responsible.

Figure 6.5 illustrates a scenario where the *Business Data* of an organisation *Business Owner* is permeating from their on-premise network to the infrastructure of their cloud service provider *IaaS Provider*. At this stage it is possible to identify potential vulnerabilities affecting the network connection, when data is passed from point to point, which may impact the security of the transmitted data or processes on the end-point connection.

Network Security

We now discuss network security in the PaaS cloud service model and demonstrate how our visual models could be used to mitigate the following vulnerability:

- Insecure SSL Trust configuration

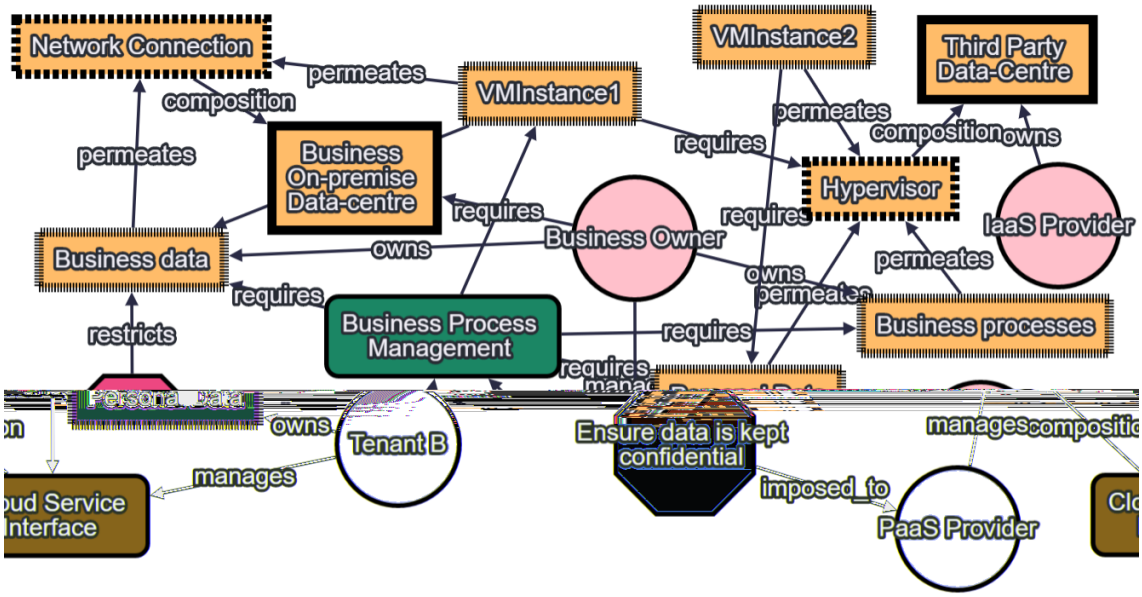

Figure 6.5: Cloud environment model of the network security scenario.

In this scenario the *"Business Data"* permeates the infrastructure node *"Network connection"*, which is part of the physical infrastructure *"Business On-premise Data-centre"*. In turn the *"Network connection"* has the permeates relationship to *"VMInstance1"*, indicating that the two nodes are connected. A summary of the *CEM* formally describing the system-under-design is provided to highlight key relationships and properties given that $CEM = \langle RKB,\ AM,\ OR,\ CSM,\ CVM,\ CTM \rangle$:

The $RKB = \langle \{Business\_data, Customer\_Data, VMInstance1, Personal\_Data, VMInstance2, Business\_processes\}, \{Hypervisor, Network\_Connection\}, \{Business\_On\text{-}premise\_Data\text{-}centre, Third\_Party\_Data\text{-}Centre,\}, \{permeates(VMInstance1, Network\_Connection), permeates(VMInstance2, Hypervisor), permeates(Business\_processes, Hypervisor), permeates(Business\_data, Network\_Connection), permeates($

Personal_Data,Hypervisor),composition(Hypervisor,Third_Party_Data-Centre)
,composition(Network_Connection,Business_On-premise_Data-centre)} $\rangle$, the AM = $\langle$*Business_Owner,*{*Cloud_Service_Interface*}, {*Business_processes,Business_data,*
Business_On-premise_Data-centre}, {*owns(Business_Owner,Business_processes),*
owns(Business_Owner,Customer_Data),owns(Business_Owner,On-premise_Data-
centre),manages(Business_Owner,Cloud_Service_Interface){*CU*}$\rangle$, the CSM = $\langle$*{Cloud_Service_Interface(Public,{SaaS},{}),Business_Process_*
Management(Public,{PaaS},{Business_data,Personal_data,VMInstance1})}$\rangle$. At this stage the CVM and CTM are empty because no security concepts such as threats or vulnerabilities have been identified.

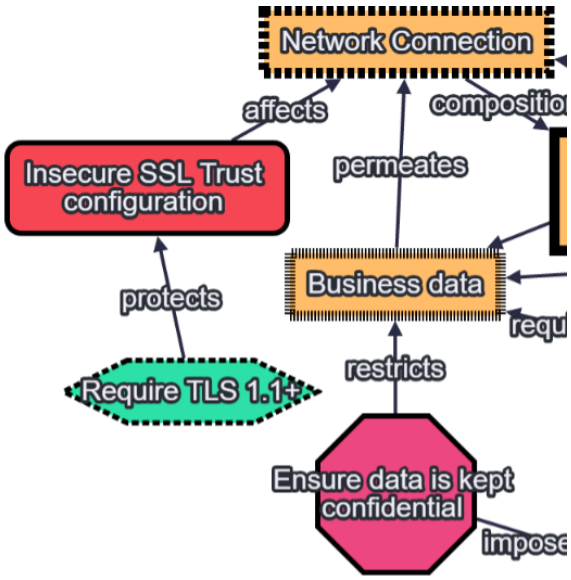

Figure 6.6: Identified vulnerability and security mechanism in the network security scenario.

During the vulnerability analysis the *RKB* is examined, where *"Network connection"* has the attribute *ResourceType* which contains *"network"*, indicating through the NVD that vulnerabilities such as *"Insecure SSL Trust configuration"* affects the node *"Network connection"*. According to OWASP (Wichers, 2013), *"Require TLS 1.1+"* is one of the potential security mechanisms which can be used to mitigate the *"Insecure SSL Trust configuration"* vulnerability. Figure 6.6 illustrates the cloud environment model after performing the transparency analysis which shows, through

the neighbouring connections between the network node and the physical infrastructure, that the vulnerability *"Insecure SSL Trust configuration"* should be addressed by the actor responsible for the physical infrastructure. In the transparency analysis, this responsibility is inferred from the composition relationship between the infrastructure node *"Network Connection"* to the physical infrastructure *"Business On-premise Data-centre"*, where the *"Business On-premise Data-centre"* is owned by the cloud user *"Business Owner"*. The developer, with guidance from our visual models, is therefore able to identify the potential security mechanisms for mitigating the vulnerability *"Insecure SSL Trust configuration"* and the actors responsible for ensuring the security constraint *"Ensure data is kept confidential"* is satisfied. In this scenario the security responsibility is enumerated through the security constraint:

- *restrictsresource(Ensure_data_is_kept_confidential,Business_data,PaaS_Provider)* where *Ensure_data_is_kept_confidential* has the *securityRequirements* property *"The cloud actor PaaS_Provider is responsible for Ensure_data_is_kept_confidential on Business_data: Security mechanism Require_TLS_1.1+ to protect against vulnerability Insecure_SSL_Trust_configuration"*

6.1.3 Infrastructure as a Service

In the context of an IaaS service model, the cloud users share joint responsibility with cloud service providers for the management of their security needs. While the degree and scope of the responsibilities shared is vendor dependent, the cloud user is responsible for configuring and managing the security controls and policies of the applications they deploy. The cloud service provider is therefore responsible for the security of the physical assets they own, for instance the cloud infrastructure and underlying components such as networks and infrastructure nodes.

Therefore in order to understand and provide transparency on the security needs of cloud users, the developers of a cloud system need to define the assets belonging to each cloud user, the expected security challenges and how to mitigate such challenges. To demonstrate this, we now describe a scenario where a developer identifies a vulnerability affecting their processes, how it impacts their application and how they are able to address these issues. We then outline how the visual model shown in Figure 6.7 helps developers describe the cloud security requirements to address

identified issues in this scenario.

Security Responsibilities

We now discuss the security responsibilities of cloud users and cloud service providers in the IaaS cloud service model, in order to demonstrate how our visual models could be used to mitigate the following vulnerability:

- Virtualization vulnerability

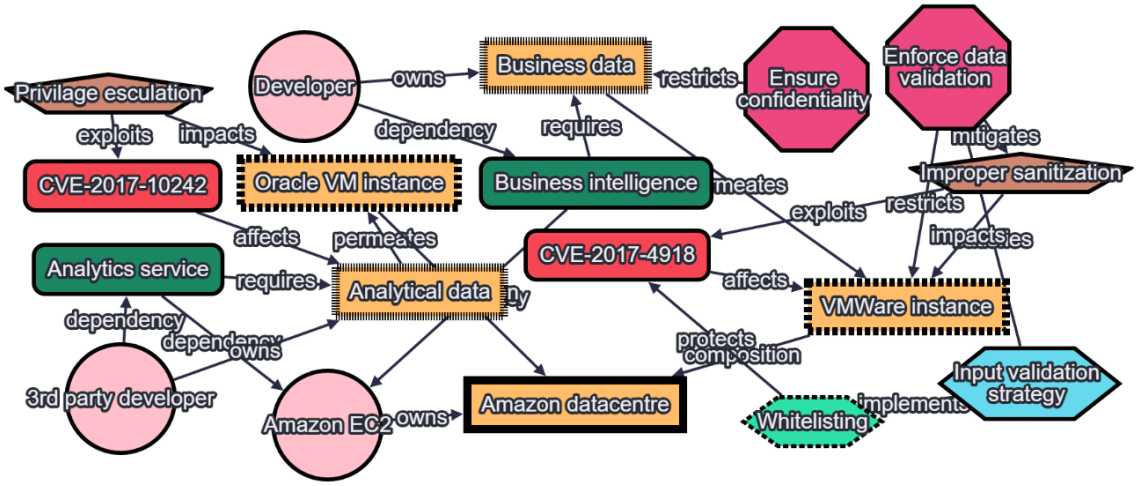

Figure 6.7: Cloud environment model showing how to mitigate the virtualisation vulnerability affecting an infrastructure node.

The cloud environment model illustrated in Figure 6.7 was modelled from the perspective of the cloud actor *"Developer"*, where their goal is to understand the security needs of their cloud system-under-design and to provide transparency in the security responsibilities. The *"Developer"* owns and deploys the cloud service *"Business intelligence"* through the IaaS cloud service model, which is hosted on the infrastructure owned by the cloud service provider *"Amazon EC2"*. The *"Developer"* owns and is responsible for configuring the infrastructure node *"VMWare instance"*, which represents a virtual machine instance. The specific properties of this instance is shown on the top left of Figure 6.7.

Using the vulnerability analysis from Section 5.3.1, the *"Developer"* identifies, based on the vendor, version and type properties of *"VMWare instance"*, the vulnerability *"Virtualisation vulnerability"*. Specifically the details of the vulnerability

is enumerated in the properties shown on the left of the Figure 6.7 where the description property contains the value *"CVE-2017-4918"*, representing the CVE entry matching the product affected in *"VMware instance"*.

We then perform the threat analysis defined in Section 5.3.1, identifying the threat *"Improper sanitization"* from CWE, specifically the entry *"CWE–77: Improper Sanitization of Special Elements used in a Command ("Command Injection")"*. The security constraint *"Enforce data validation"* is identified, using the security mitigation analysis in Section 5.3.2. The security objective *"Input validation strategy"* implementing the security mechanism *"Whitelisting"* are then identified based on the CWE entry, as part of the security mitigation analysis. Therefore the security challenges have been identified in the model, along with the mitigation strategy.

The next step is to determine the scope of responsibility in order to address the identified security issues through the enforcement of security controls. The infrastructure node *"VMWare instance"* targeted in this model is part of the physical infrastructure *"Amazon datacentre"*, which is owned by the cloud service provider *"Amazon EC2"*. Therefore though the *"owns"* relationship, it is the responsibility of *"Amazon EC2"* to ensure all identified vulnerabilities and threats associated through the *"composition"* relationship with the object of ownership, *"Amazon datacentre"*, is addressed. In this instance, the mitigation strategy is the identified security constraint *"Enforce data validation"*, security objective *"Input validation strategy"* and the security mechanism *"Whitelisting"*. Therefore in this scenario the security responsibility is enumerated through the security constraint:

- *restrictsresource(Enforce_data_validation,VMWare_instance,Amazon_EC2)* where *Enforce_data_validation* has the *securityRequirements* property *"The cloud actor Amazon_EC2 is responsible for Enforce_data_validation on VMWare_instance: Security mechanism Whitelisting to protect against vulnerability CVE-2017-4918, Security constraint Enforce_data_validation mitigates the threat improper_sanitization, Security objective input_validation_strategy satisfies security constraint Enforce_data_validation and implements security mechanism Whitelisting"*

In summary the developer has identified the vulnerability *"Virtualisation vulner-*

ability" and threat *"Improper sanitization"*, which targets the infrastructure node *"VMWare instance"*. The mitigation strategy to address this has been identified as the security objective *"Input validation strategy"* and the security mechanism *"Whitelisting"*. The physical infrastructure *"Amazon datacentre"*, owned by the cloud service provider *"Amazon EC2"*, is composed of the infrastructure node *"VMWare instance"*. Therefore in order to provide transparency, the security responsibilities of the cloud service provider *"Amazon EC2"* is to enforce the security objective *"Input validation strategy"*, the security mechanism *"Whitelisting"* and the security constraint *"Enforce data validation"*.

6.2 VisiOn Case Study: Municipality of Athens

In this section we evaluate our Secure Cloud Environment Framework using a scenario from the Visual Privacy Management in User Centric Open Environment(VisiOn)[1] European project case study. VisiOn is an integrated tool solution that organisations can connect to part or all of their systems within their infrastructure. The VisiOn tool is accessed through a web interface by users such as citizens, allowing them to interact with the public administration(PA) in order to search for information, updates and requirements on the management of their data. This case study was selected because of the relevancy between the VisiOn project and the cloud computing needs of their stakeholders, specifically in scenarios where the cloud security requirements of stakeholders need to be understood as they transition from traditional systems to a cloud computing environment.

The scenario from this case study was selected from one of three pilot scenarios proposed by a local Greek government development company named "City of Athens IT Company" (DAEM) for the VisiOn project. In this scenario an Athenian city resident, George, submits a request for an annual membership at a public swimming pool in Athens, with a disability discount. In order to verify his current residence and medical situation to be eligible for specific city services, the swimming pool faculty is responsible for verifying the birth and medical certificates of the applicant, in order to approve the citizenship status and medical circumstances for a discount.

[1] http://www.visionprivacyplatform.eu

The swimming pool administration team will outsource their computing needs to a third party cloud computing provider, where the business data and processes of their system is stored and processed externally in a cloud computing environment. The municipality of Athens is responsible for providing copies of various certificates of citizens, where their data is accessed and stored by certified governmental cloud computing providers. The clinic in this scenario stores the medical information and certificate of citizens on certified health-care cloud computing infrastructure. Therefore in this scenario the personal data of citizens is exchanged between the public authority from the municipality of Athens, personnel at a clinic and the swimming pool administration staff. This scenario is examined from the perspective of DAEM, where they are responsible for determining the cloud security requirements of the stakeholders in this scenario, and disseminating the identified requirements to the appropriate parties.

The stakeholders in this case study are the representatives of the public authority (Municipality of Athens), swimming pool administrator and Athenian citizens (George), where their requirements are represented by DAEM.

The description and security requirements of the stakeholders in the Municipality of Athens scenario were obtained as part of the deliverables in the VisiOn project, where the document is accessible from [2]. The security requirements is as follows:

- SR1 The actor *SP Information System* require *Badge* to authenticate in order to achieve goal *Badge issued*.
- SR2 The actor *SP Information System* require unlinkability on *Badge* in order to achieve goal *Badge issued*.
- SR3 The actor *Municipality of Athens* shall ensure the integrity of the required resource *Medical certificate* permeating to *SP information system* is preserved.
- SR4 The actor *Municipality of Athens* shall ensure the confidentiality of the required resource *Medical certificate* permeating to *SP information system* is preserved.

[2]http://www.visioneuproject.eu/wp-content/uploads/2016/03/2015-VSN-RP-053-D2.2-Citizens-and-Public-Administrations-Privacy-Requirements.pdf

- SR5 The actor *Municipality of Athens* shall ensure the integrity of the required resource *Birth certificate* permeating to *SP administrator* is preserved.
- SR6 The actor *Municipality of Athens* shall ensure the confidentiality of the required resource *Birth certificate* permeating to *SP administrator* is preserved.
- SR7 The actor *SP Information System* shall ensure the confidentiality of the required resource *Bank details* is preserved.
- SR8 The actor *SP Information System* shall ensure the integrity of the required resource *Bank details* is preserved.
- SR9 The actor *Clinic* shall ensure the integrity of the required resource *Medical certificate* is preserved when received by actor *SP information system.*
- SR10 The actor *Clinic* shall ensure the confidentiality of the required resource *Medical certificate* is preserved when received by actor *SP information system.*

Taking these requirements into account, the aim of this scenario is to realise these requirements in a cloud computing environment. This is achieved by applying the Secure Cloud Environment Framework to refine cloud models and perform security analysis in order to obtain cloud security requirements.

6.2.1 Application and Outcome of the Secure Cloud Environment Framework

In this subsection we report on the outcomes after applying the Secure Cloud Environment Framework, mentioned in Chapter 4.1, to the Municipality of Athens scenario. The application of the framework was carried out by the author, based on information provided through requirements specification documents describing the context and contents of the Municipality of Athens scenario in the VisiOn European project. Here we outline the outcomes of each activity following the application of the Secure Cloud Process, focusing on the output in terms of producing and refining cloud models throughout the modelling and security analysis activities without detailing each step in full technical detail. We also describe the steps involved in the application of the SectroCloud tool in a narrative format.

Organisational Goal Model

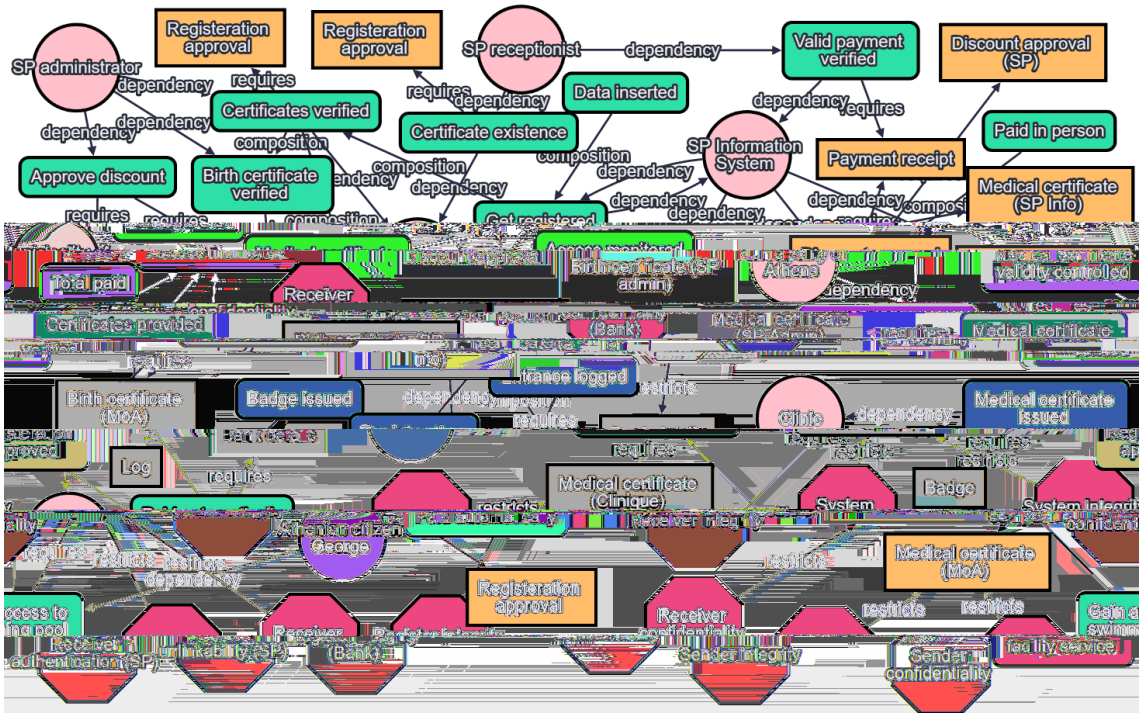

Figure 6.8: Organisational goal model of the VisiOn Municipality of Athens scenario.

The Municipality of Athens scenario describes a situation where the primary goal of an Athenian citizens is a request for an annual membership for a local swimming pool in Athens, which is processed by the administration staff. In the process of determining the validity for a disability discount, the administration staff at the swimming pool requests access to copies of the citizens birth and medical certificates. This is carried out through communication with public administration staff, represented as the actors *Municipality of Athens* and medical personnel at a local clinic. Figure 6.8 shows a graphical representation of an organisational goal model in the Municipality of Athens scenario, generated using the SectroCloud tool. The organisational goal model was constructed by the author based on the information found in stakeholder requirements and specification documents available from the VisiOn European project. As our work builds upon and extends the Secure Tropos methodology, the SectroCloud is fully capable of creating the organisational goal model following the same notation and relationships defined in Secure Tropos.

We now describe the steps taken using the SectroCloud tool in order to produce the an organisational goal model shown in Figure 6.8. The following concepts are used in the creation of the organisational goal model; *actor*, *resource*, *goal* and *security constraint*. The following relationships are used in the organisational goal model; *dependency*, *requires*, *restricts* and *composition*. Note that when the user selects two nodes and clicks the *Add Edge* button, the SectroCloud tool automatically assigns the edge according to the rules of our proposed cloud metamodel. As an example, if the user selects an *Actor* node and a *Goal* node, clicking on the *Add Edge* button will create an edge called *dependency* from the first node selected to the second node selected. If the two selected nodes has the option of selecting more than one type of edge between them, the user is then required to select from a selection of edges, such as *owns*, *manages* or *permeates*. With this set of concepts and relationships, the user will create the organisational goal model by adding the nodes using the buttons corresponding to the required concepts, populating the model with *actor*, *goal*, *resource* and *security constraint* instances. Each instance can be renamed by right clicking on the node and editing the properties. The user will then create the relationships between nodes by selecting a pair of nodes and adding edges between them using the "Add Edge" button. The completion of these two activities using the SectroCloud tool results in the creation of the organisational goal model shown in Figure 6.8.

The actors in this scenario are; *Athenian citizen George*, *SP Information System*, *Municipality of Athens*, *SP administrator*, *Clinic* and *SP receptionist*. The primary goal of the actor *Athenian citizen George* in the scenario is *Gain access to swimming pool facility service*. This is composed of the sub-goals *Medical certificate issued* and *Get registered*. The model has six conceptual boundaries corresponding to each actor, where each bounded area includes the goals and resources of each actor. Several goals depend on the goals located in cross-boundary areas, denoting the exchange of data outside the scope of individual organisations or stakeholders. For example the *Municipality of Athens* depends on the *SP administration* to ensure the medical certificate is confidential during transmission.

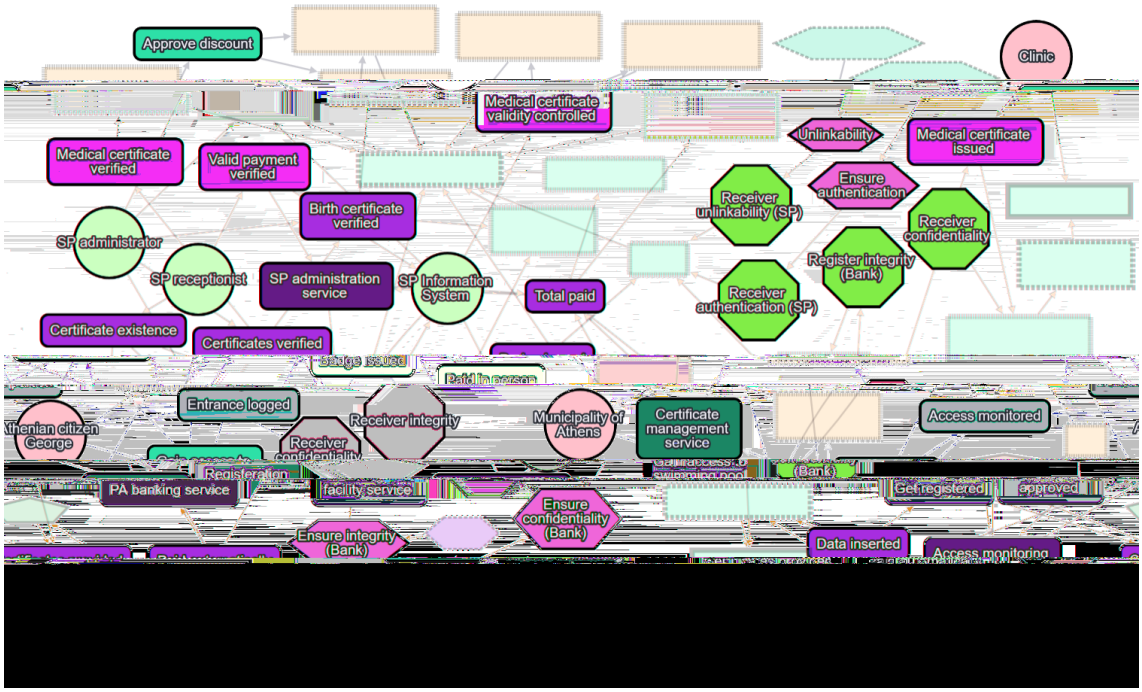

Figure 6.9: View of the model using the organisation filter.

Organisational Cloud System

Following the creation of the organisational goal model, the next activity focuses on the identification and creation of cloud services, corresponding to the goals of the system and stakeholders. As this activity is performed from the perspective of DAEM, we identify the cloud services required in order to transmit, access and store data towards achieving the primary goal of *Gain access to swimming pool facility service* by the actor *Athenian citizen George*.

Therefore the following concepts are used in the creation of the organisational cloud system; *actor*, *cloud actor*, *goal*, *cloud service*, *security objective* and *security constraint*. The following relationships are used in the organisational cloud system; *dependency*, *requires*, *restricts*, *composition* and *mitigates*. With this set of concepts and relationships, the user will create the organisational goal model by adding the nodes using the buttons corresponding to the required concepts, populating the model with instances of the *actor*, *cloud actor*, *goal*, *cloud service*, *security objective* and *security constraint* concepts. The relationships between each node is added in the same way described in the SectroCloud tool, by using the *Add Edge* button.

The completion of this activity using the SectroCloud tool results in the creation of the organisational cloud system shown in Figure 6.8.

Thus the following instances of cloud services have been identified during this activity; *SP payment service*, *Access monitoring service*, *Certificate management service*, *PA banking service*. The *SP administration service* represents a cloud service which acts as a gateway to business processes essential for the operation of the swimming pool. The *Access monitoring service* represents a cloud service which is part of the *SP administration service*, specialising in capabilities within the scope of monitoring membership access. The *Certificate management service* represents a cloud service for managing the access, verification and issuing of medical and citizenship certificates. The *PA banking service* represents a cloud service used by public administration for banking services, such as accessing, withdrawing and setting up payments for bank accounts.

The cloud service filter shown in Figure 6.9 showcases how the graphical highlighting of a specific concept helps developers narrow the scope of a cloud model to cloud services. This filter can be accessed in the SectroCloud tool by clicking on the *organisational* button in the toolbar. Thus given a large model composed of complex relationships and concepts, developers are able to spot patterns and manipulate the graphical representation of the cloud model to further their understanding. This was useful in this scenario due to the requirements for modelling and visualising the satisfaction of security properties, such as confidentiality and integrity on components, which involves understanding visually complex chains of relationships and properties. We now discuss how the relationships and properties of the cloud services identified in this activity are refined in the following activity.

Holistic Cloud Model

The holistic cloud model is refined through organisational concepts, application concepts and infrastructure concepts, in order to output the cloud environment model in this activity. Building upon the organisational cloud system produced in the previous stage, the activity begins by focusing on refining the organisational concepts of the cloud computing environment. Specifically the following concepts are used during this stage; *goal*, *cloud service*, *actor*, *cloud actor*, *virtual resource*, *se-*

Figure 6.10: Visualisation of the holistic cloud model.

curity constraint. In addition, the following relationships are used; *restricts*, *owns*, *manages*, *requires* and *dependency*. Most crucially during the refinement of the organisational concepts, the properties of the cloud services identified are populated by the developer. This is done in the SectroCloud tool by right clicking on a cloud service instance to bring up the properties for that particular instance, where the user is able to enter the following details; name of the cloud instance, capability, security property, deployment model and service model. The *manages* and *owns* relationship between *cloud actors* and *cloud services* are also created during this stage, using the *Add Edge* button. The next stage focuses on refining the application concepts of the cloud computing environment. Specifically the following concepts are used during this stage; *goal*, *cloud service*, *actor*, *cloud actor*, *virtual resource*, *security constraint*, *threat*, *vulnerability*, *security mechanism*, *security objective*. In addition, the following relationships are used; *restricts*, *owns*, *manages*, *requires*, *satisfies*, *implements*, *mitigates*, *protects*, *exploits*, *impacts* and *dependency*. In particular, the properties of virtual resource instances are populated here, by right clicking on a virtual resource instance to bring up the properties for that particular instance, where the user is able to enter the following details; description, value, security property, product, vendor, version, type and visibility. The third and final stage focuses on refining the infrastructure concepts of the cloud computing environment. Specifically the following concepts are used during this stage; *goal*, *cloud service*, *actor*, *cloud actor*, *virtual resource*, *infrastructure node*, *physical infrastructure*, *security constraint*, *threat*, *vulnerability*, *security mechanism*, *security objective*. In addition, the following relationships are used; *restricts*, *owns*, *manages*, *requires*, *satisfies*, *implements*, *mitigates*, *protects*, *exploits*, *impacts*, *permeates* and *dependency*. In this instance, the properties of infrastructure node and physical infrastructure instances are populated, again by right clicking on the instance to bring up the properties for that particular instance.

In order to address the ten security requirements on the cloud system, represented through security constraints, the framework guides the developer through the process of devising the mitigation strategy. Therefore during the organisational modelling step, six security objectives were added to address the existing security constraints. Nine security mechanisms were proposed to implement all identified security objectives. Specifically the security constraints *Receiver authentication (SP)*

and *Receiver unlinkability (SP)* are satisfied by the security objectives *Ensure authentication* and *Unlinkability*, which are implemented by the security mechanisms *Username/password* and *Pseudonymizer tools* respectively. The security constraints *Receiver confidentiality*, *Receiver confidentiality (Bank)*, *Receiver integrity* and *Receiver integrity (Bank)* are satisfied by the security objectives *Ensure confidentiality* and *Ensure integrity*, which are implemented by the security mechanisms *Encryption*, *Host IDS* and *Network IDS* respectively. Finally the last set of security constraints *Sender integrity* and *Sender confidentiality* are satisfied by the security objectives *Ensure confidentiality* and *Ensure integrity*, and is implemented by the security mechanisms *Encryption* and *Mirroring* respectively. These security measures have been identified manually by the author and represents at a high level, a mitigation strategy for the satisfaction of the cloud security needs.

Lessons Learned

In this sub-section we discuss the lessons learned after the application of our work in the VisiOn case study. During the collaborative work with the requirement engineers in the application of the tool, we have observed issues in the scalability of models and visual presentation. Specifically the model in question was created from requirements specifying over eighty nodes, where several nodes had a high concentration of incoming and outgoing edges. Nodes with multiple edges would partially obscure or overlap with neighbouring nodes and edges, which resulted in models with sections that were visually difficult to understand. This issue with visual presentation of highly concentrated models was exacerbated in the figures produced in this thesis, primary due to the limitation of presenting readable figures which still fits within an A4 page. However in the models created by requirements engineers during the collaborative phase, the spacing was more relaxed to allow for visualising systems with higher complexity and density. This phenomenon was attributed to the fact that in a practical work space, the practitioner focuses on fragments of a system. For example Figure 6.11 shows a fragment of the cloud environment model generated for the VisiOn case study, where the spacing is relaxed. As such, they are not bound by the limitation of presenting a visual model which scales down to fit an A4 page.

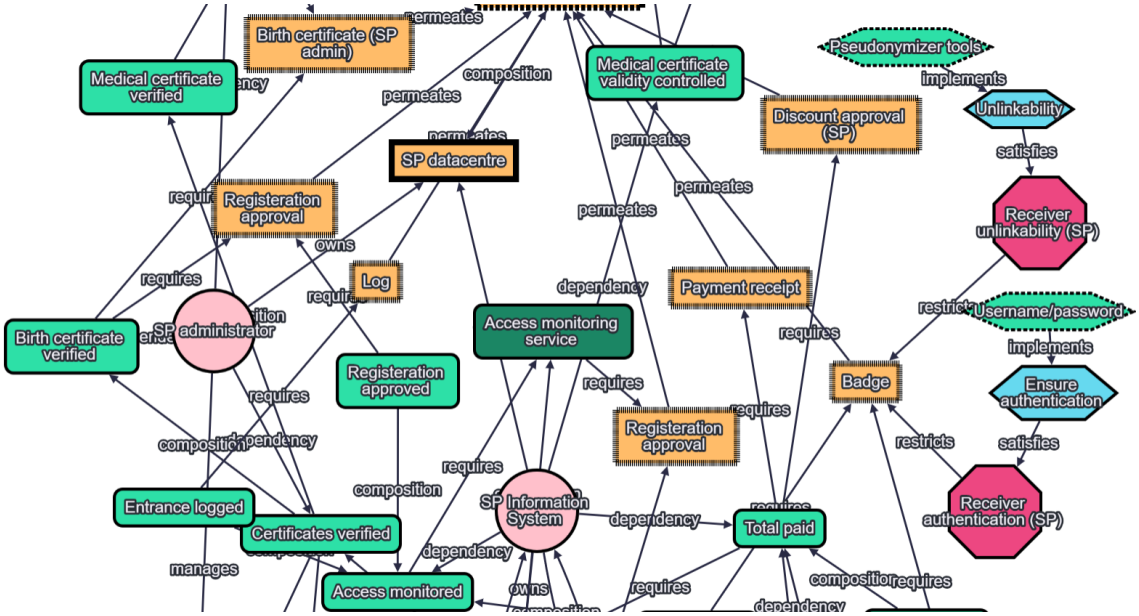

Figure 6.11: Fragment of the model in the VisiOn case study.

Therefore from this observation we conclude that from a practitioners perspective, the process of creating and editing models using the SectroCloud tool-support is manageable for systems containing up to eighty nodes. In terms of scalability and the impact on the visual presentation of models, nodes with multiple incoming and outgoing edges are partially or fully obscured when located in areas with a high density of nodes. This is further exacerbated when attempting to present the entirety of the model in one view, in particular when scaling the image to an A4 page. This can be partially avoided by allocating additional work space and providing adequate spacing between each node, at the cost of creating models which are not rendered readable when scaled down to an A4 sized page. Therefore with this limitation in mind, we have proposed as future work to provide the functionality of resizable nodes, in order to reshape nodes which has a high number of incoming or outgoing edges, such that edges and labels no longer overlap. A functionality should also be added to support modifications to the anchor points of edges, where the path of edges can be manipulated by the developer.

Another observation was regarding the usability of the tool in terms of the graphical notation and user interface. The participants involved in the development process have provided comments on the usability of the tool, noting in comparison to

sketching models on papers and boards, the tool-assisted approach was more user friendly. Specifically in one case a participant was involved in a meeting where they were responsible for presenting system requirements to stakeholders. In this case they were able to demonstrate in real time, using the tool, the requirements of the system-under-design during the video conference.

6.3 Chapter summary

In this chapter the secure cloud environment framework was evaluated on three use-cases and a case study. The use-cases demonstrate how the framework was applied to model and address security issues specific to cloud service models. In each case a list of vulnerabilities have been identified through the cloud analysis, where the affected instances of cloud computing resources were highlighted. A mitigation strategy was then created using security mechanisms and objectives to address identified security issues. The results of the VisiOn case study was then reported, identifying the security requirements of the stakeholders and how these requirements are satisfied in the cloud computing system.

Chapter 7

Conclusion and Future Work

7.1 Conclusions

In this thesis we have presented the components of our security requirements engineering framework, providing developers with semi-automated decision support for the design and analysis of cloud computing environments, to elicit and understand cloud security requirements. We are able to model and address security challenges in cloud computing environments from several perspectives, such as social engineering at the social level, mis-configuration of cloud services at the application level and hypervisor vulnerabilities leading to side-channel attacks at the infrastructure level. The Cloud Security Modelling Language allows us to express components of a cloud computing system using the concepts of a cloud service, virtual resource, infrastructure node and physical infrastructure. The language describes the correlation between cloud actors and resources through the manages, owns, permeates and composition relationships. We have described properties capturing characteristics specific to cloud computing, in particular virtualisation, multi-tenancy, jurisdiction, service models and deployment models. The Secure Cloud Process defines a systematic approach for developers of cloud computing systems, describing the steps required to model the cloud computing components of the system in order to identify and address security issues. The Cloud Security Analysis proposes three analysis techniques which, given a cloud environment model, identifies vulnerabilities and threats posing a risk to the cloud computing resources, the security mechanisms,

security objectives and security constraints to address these issues and the security responsibility of cloud actors over resources they manage or own.

We now summarise the contributions made in this thesis and how they answer the research questions in Chapter 1.

7.1.1 Secure Cloud Environment Framework

The secure cloud environment framework provides semi-automated decision support for cloud developers during the process of modelling, analysing and enforcing security requirements in cloud computing systems. We argue that the security issues in a cloud computing environment are not bound to a single domain. Therefore multiple perspectives must be considered by a developer in order to capture the security needs of cloud actors. The secure cloud environment framework addresses these limitations by providing a modelling language with the expressive power needed to describe cloud security needs at a high level of abstraction, without omitting technical details. This is demonstrated through three separate views of a cloud model, representing refinements at the organisational, application and infrastructure level. Each view allows developers to focus on the concepts and relationships represented from a specific perspective, culminating in a holistic cloud model with varying levels of granularity. Therefore the security needs of stakeholders are considered from the early requirements stage, refined through each of the three views, to describe the cloud security requirements of the system.

7.1.2 Cloud Security Modelling Language

The **Cloud Security Modelling Language** presented in Chapter 3 answers the first research question *"**RQ1**: How do we describe cloud computing concepts to capture cloud systems from a security requirements engineering perspective?"*. Specifically we address the first research objective *"**RO1**: We will extend existing security requirements approaches in a way which captures the security and cloud needs of stakeholders from the early requirements stage."* in the work undertaken to extend the Secure Tropos methodology. The second research objective *"**RO2**: We will define a modelling language capable of modelling cloud computing concepts and relationships from a security requirements engineering perspective."* is addressed

through our cloud computing concepts, relationships and properties, building upon the extensions to the Secure Tropos modelling language.

We have defined the cloud security modelling language to capture cloud computing concepts and relationships, enabling the description of components required to construct conceptual models representing secure cloud computing systems. Specifically in order to describe a cloud computing system at a high level, we define the concept of a cloud service, virtual resource, physical infrastructure and infrastructure node. To represent the idea of responsibility at a social level, we propose the manages and owns relationships between actors and resources and the permeates and composition relationship between resources. We define properties such as service model, deployment model, tenancy and location in order to describe, with a finer level of granularity, the technical specifications associated with each concept.

Our cloud security modelling language extends the Secure Tropos modelling language (Mouratidis & Giorgini, 2007), incorporating a goal-oriented security requirements engineering approach in the scope of modelling secure cloud computing systems. Specifically in Chapter 2 we have reviewed the state of the art to determine the characteristics of cloud computing systems, summarised the security issues in cloud computing and presented the current limitations in security requirements engineering approaches to model secure cloud security systems. In Chapter 3 we present our arguments that the language is able to describe cloud security requirements through our definition of cloud computing concepts, relationships and properties.

7.1.3 Secure Cloud Process

The **Secure Cloud Process** presented in Chapter 4 answers the second research question *"**RQ2**: How do we define a systematic process to guide security requirements engineers in modelling cloud computing systems?"*. Specifically the third research objective *"**RO3**: We will systematically guide the developer through the process of mapping our modelling language to models of secure cloud systems."* is addressed through the five activities of the secure cloud process, where guidance is provided from early requirements to identification of cloud security requirements.

The secure cloud process provides practitioners with a comprehensive approach to the secure cloud environment framework, ensure that they are able to consist-

ently understand and apply fundamental activities throughout the cloud security requirements elicitation process. Specifically the process provides guidance from the early requirements stage to the specification of cloud security requirements, from a security requirements engineering perspective.

During the activities in the process, the developer is made aware of the input and output artefacts, as well as the tasks to be performed during each activity. The first activity is performed under the assumption that an organisational goal model representing the requirements of the system-under-design is available. This bridges the gap between existing goal-oriented security requirements engineering approaches and provides the entry point to eliciting cloud security needs. The second activity specifies the properties of cloud services, focusing on the unique offerings of cloud computing, such as service models and deployment models, which will later determine the scope of security issues and considerations. The third activity focuses on guiding the developer through the application of concepts, relationships and properties introduced in Chapter 3, in order to construct a cloud environment model to represent the system-under-design. The key contribution of this chapter is the differentiation between the organisational, application and infrastructure levels of abstraction, and the refinement of concepts within the scope of each level. This allows the developer to understand and model from different levels of abstraction, ranging from high level concepts such as delegation of responsibilities and ownership of resources, to technical specification of security mechanisms and configuration of infrastructure nodes. The forth activity guides developers through the steps of performing security analysis, focusing on three types of analysis techniques, which is fully explained in Chapter 5.

7.1.4 Semi-Automated Reasoning Support

The **Semi-Automated Reasoning Support** presented in Chapter 5 answers the third research question *"**RQ3**: What types of analysis are useful in order to identify cloud security requirements in cloud systems?"*. Specifically the forth research objective *"**RO4**: We will provide guidelines for modelling cloud systems in order to perform semi-automated analysis."* is addressed through the formal work carried out in Chapter 5.1. The fifth research objective *"**RO4.1**: The analysis will sup-*

port threat and vulnerability identification to help security requirements engineers understand security issues in cloud systems." is addressed through the SectroCloud module described in Chapter 5.4. The sixth research objective *"**RO4.2**: The analysis will propose mitigation techniques to help security requirements engineers understand how to address security issues in cloud systems."* is addressed by the cloud security analysis and security mitigation analysis techniques defined in Chapter 5.3.

Here we have introduced the formal analysis concepts, which enables the semi-automated analysis of cloud environment model constructed using our cloud security modelling language. We presented three analysis techniques; (i) *cloud security analysis*, identifying threats and security vulnerabilities, (ii) *security mitigation analysis*, identifies and validates security constraints, objectives and mechanisms, and (iii) *transparency analysis*, identifying the security responsibilities of cloud actors and understand security needs.

We have also presented our SectroCloud module, which extends the Apparatus Software Tool(ASTo) to provide a graphical interface for practitioners. The tool supports the visual modelling of cloud environment models using concepts from our cloud security modelling language, with view filters for the organisational, application and infrastructure level of abstraction. Security analysis on generated cloud environment models are supported through the integration of the formal analysis concepts and the three analysis techniques. In addition validity and well-formedness checks can be performed on models to ensure compliance with our cloud security modelling language. This provides semi-automated reasoning using existing cloud environment models, visualising the results of analysis on the models to provide feedback to the practitioner.

7.2 Future Work

Security Patterns Currently the SectroCloud module does not support importing security patterns to a cloud environment model. While initial efforts have been taken to identify patterns from several domains in the Cloud Controls Matrix (CCM) provided by the Cloud Security Alliance (CSA), the process of replicating these security patterns using tool support is manual. This involves the creation of security controls as models using concepts from our proposed language, which can be saved

as a pattern using the SectroCloud tool support. The bulk of the work is in devising a method for the automated transformation of security patterns from expert databases to our proposed language. Further work is needed to allow the importing and exporting of patterns in the tool support, for example replicating a set of security mechanisms to mitigate a specific vulnerability and generating the pattern through our concepts in an existing model. Therefore the automated transformation of cloud security controls into security patterns through tool support will provide practitioners with an extendable library of security patterns, consisting of industry standard solutions.

Scalability and Usability of SectroCloud The focus of the SectroCloud module was to provide semi-automated tool support for practitioners, allowing the visual modelling of cloud computing systems using our cloud security concepts. Further investigation of the scalability of the visual component can improve the usability of the tool, as the effect of the complexity of visual models on the decision making process has not yet been studied within the scope of this work. Two features have been proposed to address the visual clutter of complex models; (i) enabling the manual resizing and scaling of nodes, (ii) allowing the addition of anchor points on edges. Supporting the resizing of nodes would allow developers to manually adjust the scale of each node in the model, in order to reduce the amount of overlap over edges and labels in complex models. The support for manual manipulation of edges in the model would allow developers to reshape edges which overlap, further reducing visual clutter. These two proposed features is aimed at improving the usability of the tool-support, while addressing the visual component of the scalability issue.

Bibliography

Adam, I. O. & Musah, A. (2014). Small and medium enterprises (smes) in the cloud in developing countries: A synthesis of the literature and future research directions.

Ahmad, A. & Babar, M. A. (2014). A framework for architecture-driven migration of legacy systems to cloud-enabled software. In *Proceedings of the wicsa 2014 companion volume* (p. 7). ACM.

Aljawarneh, S. A., Alawneh, A. & Jaradat, R. (2017). Cloud security engineering: Early stages of sdlc. *Future Generation Computer Systems*, *74*, 385–392.

Alliance, C. (2011). Security guidance for critical areas of focus in cloud computing v3. 0. *Cloud Security Alliance*, 15.

Almorsy, M., Grundy, J. & Müller, I. (2016). An analysis of the cloud computing security problem. *arXiv preprint arXiv:1609.01107*.

Alpeyev, P., Galante, J. & Yasu, M. (2011). Amazon.com server said to have been used in sony attack. *Bloomberg, May*, *14*.

Amyot, D., Ghanavati, S., Horkoff, J., Mussbacher, G., Peyton, L. & Yu, E. (2010). Evaluating goal models within the goal-oriented requirement language. *International Journal of Intelligent Systems*, *25*(8), 841–877.

Argyropoulos, N., Shei, S., Kalloniatis, C., Mouratidis, H., Delaney, A., Fish, A. & Gritzalis, S. (2017). A semi-automatic approach for eliciting cloud security and privacy requirements. In *Proceedings of the 50th hawaii international conference on system sciences*.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... Stoica, I. et al. (2010). A view of cloud computing. *Communications of the ACM*, *53*(4), 50–58.

Barnum, S. (2012). Standardizing cyber threat intelligence information with the structured threat information expression (stix). *MITRE Corporation*, *11*, 1–22.

Barry, D. K. (2003). *Web services, service-oriented architectures, and cloud computing*. Morgan Kaufmann.

Beckers, K., Côté, I., Faßbender, S., Heisel, M. & Hofbauer, S. (2013). A pattern-based method for establishing a cloud-specific information security management system. *Requirements Engineering*, *18*(4), 343–395.

Behl, A. (2011). Emerging security challenges in cloud computing: An insight to cloud security challenges and their mitigation. In *Information and communication technologies (wict), 2011 world congress on* (pp. 217–222). IEEE.

Belaunde, M., Casanave, C., DSouza, D., Duddy, K., El Kaim, W., Kennedy, A., ... Hendryx, S. et al. (2003). Model driven architecture guide version 1.0.1.

Bergmayr, A., Bruneliere, H., Izquierdo, J. L. C., Gorronogoitia, J., Kousiouris, G., Kyriazis, D., ... Pezuela, C. et al. (2013). Migrating legacy software to the cloud with artist. In *Software maintenance and reengineering (csmr), 2013 17th european conference on* (pp. 465–468). IEEE.

Bergmayr, A., Wimmer, M., Kappel, G. & Grossniklaus, M. (2014). Cloud modeling languages by example. In *Service-oriented computing and applications (soca), 2014 ieee 7th international conference on* (pp. 137–146). IEEE.

Bhensook, N. & Senivongse, T. (2012). An assessment of security requirements compliance of cloud providers. In *Cloud computing technology and science (cloudcom), 2012 ieee 4th international conference on* (pp. 520–525). IEEE.

Bohn, R. B., Messina, J., Liu, F., Tong, J. & Mao, J. (2011). Nist cloud computing reference architecture. In *Services (services), 2011 ieee world congress on* (pp. 594–596). IEEE.

Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F. & Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, *8*(3), 203–236.

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J. & Brandic, I. (2009). Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, *25*(6), 599–616.

Center for Computational Science Research, I., Chicago. (n.d.). Open commons consortium. http://occ-data.org/. Accessed: 2017-06-07.

Cheng, B. H. & Atlee, J. M. (2007). Research directions in requirements engineering. In *2007 future of software engineering* (pp. 285–303). IEEE Computer Society.

Chou, T.-S. (2013). Security threats on cloud computing vulnerabilities. *International Journal of Computer Science & Information Technology*, *5*(3), 79.

Chow, R., Golle, P., Jakobsson, M., Shi, E., Staddon, J., Masuoka, R. & Molina, J. (2009). Controlling data in the cloud: Outsourcing computation without outsourcing control. In *Proceedings of the 2009 acm workshop on cloud computing security* (pp. 85–90). ACM.

Chung, L., Nixon, B. A., Yu, E. & Mylopoulos, J. (2012). *Non-functional requirements in software engineering*. Springer Science & Business Media.

Dalpiaz, F., Paja, E. & Giorgini, P. (2011). Security requirements engineering via commitments. In *Socio-technical aspects in security and trust (stast), 2011 1st workshop on* (pp. 1–8). IEEE.

Dardenne, A., Van Lamsweerde, A. & Fickas, S. (1993). Goal-directed requirements acquisition. *Science of computer programming*, *20*(1-2), 3–50.

Darimont, R., Delor, E., Massonet, P. & van Lamsweerde, A. (1997). Grail/kaos: An environment for goal-driven requirements engineering. In *Proceedings of the 19th international conference on software engineering* (pp. 612–613). ACM.

Depot, T. (2014). The home depot reports findings in payment data breach investigation.

Dillon, T., Wu, C. & Chang, E. (2010). Cloud computing: Issues and challenges. In *Advanced information networking and applications (aina), 2010 24th ieee international conference on* (pp. 27–33). Ieee.

Elahi, G. & Yu, E. (2007). A goal oriented approach for modeling and analyzing security trade-offs. *Conceptual Modeling-ER 2007*, 375–390.

Ellis-Braithwaite, R., Lock, R., Dawson, R. & Haque, B. (2012). Modelling the strategic alignment of software requirements using goal graphs. *arXiv preprint arXiv:1211.6258*.

Eric, S. Y. (2009). Social modeling and i. In *Conceptual modeling: Foundations and applications* (pp. 99–121). Springer.

Fabian, B., Gürses, S., Heisel, M., Santen, T. & Schmidt, H. (2010). A comparison of security requirements engineering methods. *Requirements engineering*, *15*(1), 7–40.

Faily, S. (2011). Bridging user- centered design and requirements engineering with grl and persona cases. In *Proceedings of the 5th international i* workshop*.

Fernandes, D. A., Soares, L. F., Gomes, J. V., Freire, M. M. & Inácio, P. R. (2014). Security issues in cloud environments: A survey. *International Journal of Information Security*, *13*(2), 113–170.

Ferry, N., Rossini, A., Chauvel, F., Morin, B. & Solberg, A. (2013). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *Cloud computing (cloud), 2013 ieee sixth international conference on* (pp. 887–894). IEEE.

Foster, I., Zhao, Y., Raicu, I. & Lu, S. (2008). Cloud computing and grid computing 360-degree compared. In *Grid computing environments workshop, 2008. gce'08* (pp. 1–10). Ieee.

Fox-Brewster, T. (2015). Antivirus firm bitdefender admits breach hacker claims stolen passwords are unencrypted. *Forbes*.

Frey, S. & Hasselbring, W. (2011). The cloudmig approach: Model-based migration of software systems to cloud-optimized applications. *International Journal on Advances in Software*, *4*(3 and 4), 342–353.

Grobauer, B., Walloschek, T. & Stocker, E. (2011). Understanding cloud computing vulnerabilities. *IEEE Security & Privacy*, *9*(2), 50–57.

Group, O. M. (n.d.). Object management group, cloud standards customer council. http://www.cloud-council.org/. Accessed: 2017-09-26.

Gürses, S. F. & Santen, T. (2006). Contextualizing security goals: A method for multilateral security requirements elicitation. In *Sicherheit* (Vol. 6, pp. 42–53).

Hahn, A., Thomas, R. K., Lozano, I. & Cardenas, A. (2015). A multi-layered and kill-chain based security analysis framework for cyber-physical systems. *International Journal of Critical Infrastructure Protection*, *11*, 39–50.

Hale, M. L. & Gamble, R. (2013). Building a compliance vocabulary to embed security controls in cloud slas. In *Services (services), 203 ieee ninth world congress on* (pp. 118–125). IEEE.

Hashizume, K., Rosado, D. G., Fernández-Medina, E. & Fernandez, E. B. (2013). An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, *4*(1), 5.

Hoberg, P., Wollersheim, J. & Krcmar, H. (2012). The business perspective on cloud computing-a literature review of research on cloud computing.

Howard, M. & Lipner, S. (2006). *The security development lifecycle*. Microsoft Press Redmond.

Hughes, J. & Cybenko, G. (2014). Three tenets for secure cyber-physical system design and assessment. In *Proc. of spie vol* (Vol. 9097, 90970A–1).

Iankoulova, I. & Daneva, M. (2012). Cloud computing security requirements: A systematic review. In *Research challenges in information science (rcis), 2012 sixth international conference on* (pp. 1–7). IEEE.

ISO, I. & Std, I. (2011). Iso 27005: 2011. *Information technology–Security techniques–Information security risk management. ISO*.

Jadeja, Y. & Modi, K. (2012). Cloud computing-concepts, architecture and challenges. In *Computing, electronics and electrical technologies (icceet), 2012 international conference on* (pp. 877–880). IEEE.

Jamshidi, P., Ahmad, A. & Pahl, C. (2013). Cloud migration research: A systematic review. *IEEE Transactions on Cloud Computing*, *1*(2), 142–157.

Jordan, D. & Evdemon, J. (2011). Business process model and notation (bpmn) version 2.0. object management group. *Object ManagementGroup*.

Jürjens, J. (2002). Umlsec: Extending uml for secure systems development. *≪UML≫ 2002—The Unified Modeling Language*, 1–9.

Kalloniatis, C., Kavakli, E. & Gritzalis, S. (2008). Addressing privacy requirements in system design: The pris method. *Requirements Engineering*, *13*(3), 241–255.

Kalloniatis, C., Mouratidis, H. & Islam, S. (2013). Evaluating cloud deployment scenarios based on security and privacy requirements. *Requirements Engineering*, *18*(4), 299–319.

Kanday, R. (2012). A survey on cloud computing security. In *Computing sciences (iccs), 2012 international conference on* (pp. 302–311). IEEE.

Kandukuri, B. R., Rakshit, A. et al. (2009). Cloud security issues. In *Services computing, 2009. scc'09. ieee international conference on* (pp. 517–520). IEEE.

Kao, T.-C., Mao, C.-H., Chang, C.-Y. & Chang, K.-C. (2012). Cloud ssdlc: Cloud security governance deployment framework in secure system development life cycle. In *Trust, security and privacy in computing and communications (trustcom), 2012 ieee 11th international conference on* (pp. 1143–1148). IEEE.

Karagiannis, D. & Kühn, H. (2002). Metamodelling platforms. In *Ec-web* (Vol. 2455, p. 182).

Kissel, R. L., Stine, K. M., Scholl, M. A., Rossman, H., Fahlsing, J. & Gulick, J. (2008). Security considerations in the system development life cycle. *Special Publication (NIST SP)-800-64 Rev 2*.

Kumar, S. N. & Vajpayee, A. (2016). A survey on secure cloud: Security and privacy in cloud computing. *American Journal of Systems and Software*, *4*(1), 14–26.

Lapouchnian, A. (2005). Goal-oriented requirements engineering: An overview of the current research. *University of Toronto*, 32.

Li, Y., Cuppens-Boulahia, N., Crom, J.-M., Cuppens, F. & Frey, V. (2016). Expression and enforcement of security policy for virtual resource allocation in iaas cloud. In *Ifip international information security and privacy conference* (pp. 105–118). Springer.

Liu, L., Yu, E. & Mylopoulos, J. (2003). Security and privacy requirements analysis within a social setting. In *Requirements engineering conference, 2003. proceedings. 11th ieee international* (pp. 151–161). IEEE.

Lodderstedt, T., Basin, D. & Doser, J. (2002). Secureuml: A uml-based modeling language for model-driven security. *≪UML≫ 2002—The Unified Modeling Language*, 426–441.

Lombardi, F. & Di Pietro, R. (2011). Secure virtualization for cloud computing. *Journal of Network and Computer Applications*, *34*(4), 1113–1122.

Los, R., Shackleford, D. & Sullivan, B. (2013). The notorious nine cloud computing top threats in 2013. *Cloud Security Alliance*.

Low, G., Mouratidis, H. & Henderson-Sellers, B. (2010). Using a situational method engineering approach to identify reusable method fragments from the secure tropos methodology. *Journal of Object Technology*.

Luo, S., Lin, Z., Chen, X., Yang, Z. & Chen, J. (2011). Virtualization security for cloud computing service. In *Cloud and service computing (csc), 2011 international conference on* (pp. 174–179). IEEE.

Márquez, L., Rosado, D. G., Mouratidis, H., Mellado, D. & Fernández-Medina, E. (2015). A framework for secure migration processes of legacy systems to the cloud. In *International conference on advanced information systems engineering* (pp. 507–517). Springer.

Martin, L. (2014). Cyber kill chain®. *URL: http://cyber.lockheedmartin.com/hubfs/Gaining_the_Advantage_Cyber_Kill_Chain.pdf*.

Massacci, F., Mylopoulos, J., Zannone, N. et al. (2007). An ontology for secure socio-technical systems. *Handbook of ontologies for business interaction*, *1*, 469.

Massacci, F., Mylopoulos, J. & Zannone, N. (2010). Security requirements engineering: The si* modeling language and the secure tropos methodology. *Advances in Intelligent Information Systems*, 147–174.

Mavropoulos, O., Mouratidis, H., Fish, A. & Panaousis, E. (2017). Asto: A tool for security analysis of iot systems. In *Software engineering research, management and applications (sera), 2017 ieee 15th international conference on software engineering research, management and applications (sera)* (pp. 395–400). IEEE.

Mead, N. R. & Stehney, T. (2005). *Security quality requirements engineering (square) methodology*. ACM.

Mell, P., Grance, T. et al. (2011). The nist definition of cloud computing.

Mellado, D., Blanco, C., Sánchez, L. E. & Fernández-Medina, E. (2010). A systematic review of security requirements engineering. *Computer Standards & Interfaces*, *32*(4), 153–165.

Menzel, M., Warschofsky, R., Thomas, I., Willems, C. & Meinel, C. (2010). The service security lab: A model-driven platform to compose and explore service security in the cloud. In *Services (services-1), 2010 6th world congress on* (pp. 115–122). IEEE.

Metsch, T., Edmonds, A. et al. (2010). Open cloud computing interface-infrastructure. In *Standards track, no. gfd-r in the open grid forum document series, open cloud computing interface (occi) working group, muncie (in)*.

Modi, C., Patel, D., Borisaniya, B., Patel, A. & Rajarajan, M. (2013). A survey on security issues and solutions at different layers of cloud computing. *The Journal of Supercomputing*, *63*(2), 561–592.

Moreno-Vozmediano, R., Montero, R. S. & Llorente, I. M. (2013). Key challenges in cloud computing: Enabling the future internet of services. *IEEE Internet Computing*, *17*(4), 18–25.

Mouratidis, H. (2009). Secure tropos: An agent oriented software engineering methodology for the development of health and social care information systems. *International Journal of Computer Science and Security*, *3*(3), 241–271.

Mouratidis, H. (2011). Secure software systems engineering: The secure tropos approach. *JSW*, *6*(3), 331–339.

Mouratidis, H., Argyropoulos, N. & Shei, S. (2016). Security requirements engineering for cloud computing: The secure tropos approach. In *Domain-specific conceptual modeling* (pp. 357–380). Springer.

Mouratidis, H. & Giorgini, P. (2007). Secure tropos: A security-oriented extension of the tropos methodology. *International Journal of Software Engineering and Knowledge Engineering*, *17*(02), 285–309.

Mouratidis, H., Islam, S., Kalloniatis, C. & Gritzalis, S. (2013). A framework to support selection of cloud providers based on security and privacy requirements. *Journal of Systems and Software*, *86*(9), 2276–2293.

Nhlabatsi, A., Bandara, A., Hayashi, S., Haley, C. B., Jurjens, J., Kaiya, H., ... Nuseibeh, B. et al. (2010). Security patterns: Comparing modeling approaches. *Software engineering for secure systems: Industrial and research perspectives*, 75–111.

OASIS, S. (n.d.). Topology and orchestration specification for cloud applications version 1.0. Accessed: 2017-09-28. Retrieved from http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

Paja, E. (2014). *Sts: A security requirements engineering methodology for socio-technical systems* (Doctoral dissertation, University of Trento).

Pavlidis, M., Islam, S. & Mouratidis, H. (2011). A case tool to support automated modelling and analysis of security requirements, based on secure tropos. In *Forum at the conference on advanced information systems engineering (caise)* (pp. 95–109). Springer.

Pearson, S. & Benameur, A. (2010). Privacy, security and trust issues arising from cloud computing. In *Cloud computing technology and science (cloudcom), 2010 ieee second international conference on* (pp. 693–702). IEEE.

Pohl, K. (2010). *Requirements engineering: Fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated.

Pumvarapruek, N. & Senivongse, T. (2014). Classifying cloud provider security conformance to cloud controls matrix. In *Computer science and software engineering (jcsse), 2014 11th international joint conference on* (pp. 268–273). IEEE.

Qian, L., Luo, Z., Du, Y. & Guo, L. (2009). Cloud computing: An overview. *Cloud computing*, 626–631.

Ramgovind, S., Eloff, M. M. & Smith, E. (2010). The management of security in cloud computing. In *Information security for south africa (issa), 2010* (pp. 1–7). IEEE.

Rebollo, O., Mellado, D. & Fernández-Medina, E. (2012). A systematic review of information security governance frameworks in the cloud computing environment. *J. UCS*, *18*(6), 798–815.

Rebollo, O., Mellado, D., Fernández-Medina, E. & Mouratidis, H. (2015). Empirical evaluation of a cloud computing information security governance framework. *Information and Software Technology*, *58*, 44–57.

Rimal, B. P. & Choi, E. (2012). A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing. *International Journal of Communication Systems*, *25*(6), 796–819.

Ristenpart, T., Tromer, E., Shacham, H. & Savage, S. (2009). Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the 16th acm conference on computer and communications security* (pp. 199–212). ACM.

Rong, C., Nguyen, S. T. & Jaatun, M. G. (2013). Beyond lightning: A survey on security challenges in cloud computing. *Computers & Electrical Engineering*, *39*(1), 47–54.

Ruo-xin, Z., Cui, X.-j., Gong, S.-j., Ren, H.-k. & Chen, K. (2014). Model for cloud computing security assessment based on ahp and fce. In *Computer science & education (iccse), 2014 9th international conference on* (pp. 197–204). IEEE.

Saxena, S. (2013). Ensuring cloud security using cloud control matrix. *International Journal of Information and Computation Technology*, 933–938.

Sengupta, S., Kaulgud, V. & Sharma, V. S. (2011). Cloud computing security–trends and research directions. In *Services (services), 2011 ieee world congress on* (pp. 524–531). IEEE.

Shei, S., Alcaniz, L. M., Mouratidis, H., Delaney, A., Rosado, D. G. & Fernandez-Medina, E. (2015). Modelling secure cloud systems based on system requirements. In *Evolving security and privacy requirements engineering (espre), 2015 ieee 2nd workshop on* (pp. 19–24). IEEE.

Shei, S., Delaney, A., Kapetanakis, S. & Mouratidis, H. (2015). Visually mapping requirements models to cloud services. In *Dms* (pp. 108–114).

Shei, S., Kalloniatis, C., Mouratidis, H. & Delaney, A. (2016). Modelling secure cloud computing systems from a security requirements perspective. In *International conference on trust and privacy in digital business* (pp. 48–62). Springer.

Shei, S., Mouratidis, H. & Delaney, A. (2017). A security requirements modelling language to secure cloud computing environments. In *Enterprise, business-process and information systems modeling* (pp. 337–345). Springer.

Subashini, S. & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of network and computer applications*, *34*(1), 1–11.

Susi, A., Perini, A., Mylopoulos, J. & Gi, P. (2005). The tropos metamodel and its use. *Informatica*, *29*(4).

Takabi, H., Joshi, J. B. & Ahn, G.-J. (2010). Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, *8*(6), 24–31.

Thaweejinda, J. & Senivongse, T. (2014). Semantic search for cloud providers with security conformance to cloud controls matrix. In *Computer science and software engineering (jcsse), 2014 11th international joint conference on* (pp. 286–291). IEEE.

Ullah, K. W., Ahmed, A. S. & Ylitalo, J. (2013). Towards building an automated security compliance tool for the cloud. In *Trust, security and privacy in computing and communications (trustcom), 2013 12th ieee international conference on* (pp. 1587–1593). IEEE.

Van Lamsweerde, A. (2001). Goal-oriented requirements engineering: A guided tour. In *Requirements engineering, 2001. proceedings. fifth ieee international symposium on* (pp. 249–262). IEEE.

Van Lamsweerde, A. (2004). Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th international conference on software engineering* (pp. 148–157). IEEE Computer Society.

Van Lamsweerde, A. et al. (2007). Engineering requirements for system reliability and security. *NATO Security Through Science Series D-Information and Communication Security*, *9*, 196.

Van Lamsweerde, A., Darimont, R. & Letier, E. (1998). Managing conflicts in goal-driven requirements engineering. *IEEE transactions on Software engineering*, *24*(11), 908–926.

Vaquero, L. M., Rodero-Merino, L., Caceres, J. & Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, *39*(1), 50–55.

Wichers, D. (2013). Top 10-2013: The ten most critical web application security risks. *The Open Web Application Security*.

Wooldridge, M. (1997). Agent-based software engineering. *IEE Proceedings-software*, *144*(1), 26–37.

Yahya, F., Walters, R. J. & Wills, G. B. (2015). Modelling threats with security requirements in cloud storage. *Int. J. Inf. Secur. Res.(IJISR)*, *5*(2), 551–558.

Yang, H. & Tate, M. (2012). A descriptive literature review and classification of cloud computing research. *CAIS*, *31*, 2.

Youseff, L., Butrico, M. & Da Silva, D. (2008). Toward a unified ontology of cloud computing. In *Grid computing environments workshop, 2008. gce'08* (pp. 1–10). IEEE.

Yu, E. (2011). Modelling strategic relationships for process reengineering. *Social Modeling for Requirements Engineering*, *11*, 2011.

Yu, E. S. (1997). Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements engineering, 1997., proceedings of the third ieee international symposium on* (pp. 226–235). IEEE.

Yu, E. & Mylopoulos, J. (1998). Why goal-oriented requirements engineering. In *Proceedings of the 4th international workshop on requirements engineering: Foundations of software quality* (Vol. 15, pp. 15–22).

Zardari, S. & Bahsoon, R. (2011). Cloud adoption: A goal-oriented requirements engineering approach. In *Proceedings of the 2nd international workshop on software engineering for cloud computing* (pp. 29–35). ACM.

Zhang, Q., Cheng, L. & Boutaba, R. (2010). Cloud computing: State-of-the-art and research challenges. *Journal of internet services and applications*, *1*(1), 7–18.