D2.2
System modelling tools: Full Prototype Implementation
15/01/2013

Dissemination Level: PU

Explanation of the dissemination level in relation to other classification systems that may be in use, e.g. explaining that a report may be "CO" as defined in the DoW, but "Unclassified" in terms used in corporate or military classification systems familiar to security officers.

Lead Author:
A. Chakravarthy, IT Innovation Centre

Contributors:
M. Surridge, IT Innovation Centre
M. Hall-May, IT Innovation Centre
B. Nasser, IT Innovation Centre
W. Chen, IT Innovation Centre
T. Leonard, IT Innovation Centre

Internal Reviewer: H. Zeiner, JRS

Version History

Version	Date	Changes	Author
0.1	14/05/2012	Created initial outline	A.Chakravarthy
0.2	23/05/2012	Updated description on core ontology modelling	A.Chakravarthy, M.Surridge
0.3	10/06/2012	Updated network and space asset threats	M.Hall-May
0.4	15/06/2012	Updated threat tables	B.Nasser, M.Surridge, A.Chakravarthy
0.5	23/06/2012	Updated SAM policy model section	T.Leonard
0.6	02/07/2012	Updated concrete model generation section	M.Hall-May, A.Chakravarthy
0.7	03/07/2012	Finalised deliverable for review	A.Chakravarthy
0.8	06/07/2012	Internal QA done (first revision)	H.Zeiner
0.9	08/07/2012	Final release	A.Chakravarthy
1.0	09/07/2012	Fixed minor formatting issues and WP leader attribution.	M.Surridge
1.1	25/12/2012	Various updates describing work since summer 2012.	M.Surridge
1.2	03/01/2013	Updates to Section 5 and minor updates throughout the document.	A.Chakravarthy
1.3	07/01/2013	Internal QA done (second revision)	H.Zeiner
1.4	08/01/2013	Final updates after inter nal review	A.Chakravarthy
1.5	15/01/2013	Fixed minor formatting issues prior to deliver y to the EC.	M.Surridge

Executive Summary

This report represents deliverable D2.2 from the SERSCIS project, as specified in the revised project Description of Work (SERSCIS: Annex I Description of Work, 2011). It describes the system modelling technology for the SERSCIS framework. This provides a generic dependability model ontology from which an abstract system model of a yet to be composed system can be reliably created. It also provides a means to model access control mechanisms using either object capability security, or legacy access control.

This report describes the updated 'final' core SERSCIS modelling ontology. This is an update from the original 'proof of concept' version, taking account of the lessons learned from the proof of concept test bed and validation scenarios.

The main issues with the proof of concept ontology that needed to be addressed were as follows:

1. All threat instances are considered to be potentially active, regardless of whether it is possible, given the system topology, for the threat to be carried out;
2. The effect of threats is not always confined to the assets directly involved; sometimes the effect on one asset has knock-on consequences for other assets;
3. Threats are considered to be independent when testing threat activity hypotheses via Bayesian inference, but we know threats often have common causes;
4. Some types of network-related threats could not be modelled due to the lack of 'connectivity' assets (networks, spaces, etc) in the proof-of-concept ontology.

To tackle these limitations we have added the new asset types in the dependability model, and introduced the concept of opportunities and secondary behaviours in the core ontology.

It is possible to optimise the Bayesian estimator and reduce its run-time, but that will only cut the time needed to process each threat activity hypothesis. It will not be enough to compensate for a factorial increase in the number of hypotheses that must be sampled if we are to include multi-threat situations needed to address points (2) and (3) above. What we need is a way to recognise which threats are secondary effects, so we can include them without sampling other combinations of multiple threats. This has been done by extending the ontology to model secondary effects as separate threats that are caused by an observable behaviour in an observable asset.

Threats should normally be assumed to be possible. However, threats that can only be carried out if the attacker is sufficiently well connected can be modelled by adding an opportunity rule defining the required connectivity. If the rule evaluated to 'False' then the threat is in principle impossible. However, this extension proved less useful because it depends on having complete knowledge of any potential attacker (malicious or otherwise) in the system. Since attackers often remain undetected prior to an attack, one cannot completely exclude threats on this basis. This is equivalent to cases where a threat that seems to be blocked by security controls may still occur if the controls are not correctly implemented.

In order to capture information pertaining to the execution of attacks on communications between assets, it is necessary to extend the generic dependability model to include those assets used for communication. We have added additional assets in the form of Networks, Spaces and Servers to the dependability model and also encoded how threats involve these new assets alongside other existing assets in the system. Further, we have separated the assets into two distinct types, namely, Logical Assets and Physical Assets.

During the process of system modelling, we have also reviewed and compared out modelling approach with existing standards such as ISO 270001, MEHARI, CRAMM, Risk IT Framework etc. The state of the art section of the report describes the outcomes of this investigation in detail.

Table of Contents

List of Figures

List of Abbreviations

Abbreviation	Meaning
ACDM	Airport Collaborative Decision Making
AGHT	Actual Ground Handling Time
AIBT	Actual In-Block Time
ALDT	Actual Landing Time
ANSP	Air Navigation Service Provider
AOBT	Actual Off-Block Time
API	Application Programming Interface
ARDT	Actual Ready Time (for Movement)
ASAT	Actual Startup Approval Time
ASRT	Actual Startup Request Time
ATM	Air Traffic Management
ATOT	Actual Take-Off Time
CDM	Collaborative Decision Making
CEP	Complex Event Processing
CFMU	Central Flow Management Unit
CI	Critical Infrastructure
COBIT	Control Objectives for Information and related Technology
CORAS	A Platform for Risk Analysis of Security Critical Systems
CRUD	Create Read Update Delete
DBSy	Domain Based Security
DoW	Annex I of the EC Contract, which provides the Description of Work for the project.
DST	Decision Support Tool
EIBT	Estimated In-Block Time
EMOF	Essential Meta Object Facility
EOBT	Estimated Off-Block Time
GH	Ground Handler
GIT	Free and open source, distributed version control system
GRIA	Service Oriented Collaborations for Industry and Commerce (orig. Grid Resources for Industrial Applications)
HAZOP	Hazard and Operability Study
ICT	Information Communication Technology
ITIL	Information Technology Infrastructure Library
KB	Knowledgebase
KPI	Key Performance Indicator
NextGRID	Architecture for Next Generation Grid Projects
OGC	Office of Government Commerce
OWL	Web Ontology Language
OWL-DL	Web Ontology Language (Description Logics)
OWL-S	Web Ontology Language for Services
OWL-WS	OWL for Workflow and Services
PAP	Policy Administration Point
PBAC	Policy Based Access Control
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
PoC	Proof of Concept
QoE	Quality of Experience
QoS	Quality of Service
RDF	Resource Description Framework

Abbreviation	Meaning
RDF-S	Semantic Resource Description Framework
RFP	Resource Functional Profile
SAM	SERSCIS Access Modelling
SDA	Service Dependability Agreement (is a SLA that also describes dependability properties)
SDM	SERSCIS Dependability Model (a system-independent ontology to model assets, threats and controls in dynamic, multi-stakeholder systems)
SERSCIS	Semantically Enhanced Resilient and Secure Critical Infrastructure Services
SES	Single European Sky
SESAR	Single European Sky ATM Research
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOBT	Scheduled Off-Block Time
SOI	Service Oriented Infrastructure
SoS	System of Systems
SPARQL	SPARQL Protocol and RDF Query Language
SUPER	Semantics Utilised for Process Management within and between Enterprises
SWIM	System Wide Information Management
TOBT	Target Off-Block Time
TOC	Table of Contents
TSAT	Target Start Approval Time
UDDI	Universal Description Discovery and Integration
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WP	Work Package
WP1	Project Management
WP2	System Modelling
WP3	System Governance
WP4	System Composition
WP5	Decision Support
WP6	Testbed Integration
WP7	Application Case Studies
WP8	Dissemination and Impact
WSDL	Web Service Description Language
WSLA	Web Service Level Agreements
WSMO	Web Service Modelling Ontology
XML	eXtensible Markup Language

1 Introduction

This deliverable describes the updated 'final' SERSCIS modelling ontology. This is an update from the original 'proof of concept' version described in [1], taking account of the lessons learned from the proof of concept test bed and validation scenarios.

The main issues with the proof of concept ontology that need to be addressed are as follows:

1. All threat instances are considered to be potentially active, regardless of whether it is possible, given the system topology, for the threat to be carried out;
2. The effect of threats is not always confined to the assets directly involved; sometimes the effect on one asset has knock-on consequences for other assets;
3. Threats are considered to be independent when testing threat activity hypotheses via Bayesian inference, but we know threats often have common causes;
4. Some types of threats could not be modelled due to the lack of 'connectivity' assets (networks, spaces, etc), which should determine whether and by what path the threat could be attempted.

It should also be noted that the proof of concept ontology does not support all aspects of the SERSCIS system modelling approach. The effect of threats (i.e. the probability that an active threat will induce detectable compromised behaviour) was not represented in the semantic model.

In the next section, we start by providing a brief review of the state of the art for the approaches used for semantic system modelling and reasoning (threats, controls and assets), summarising rather than repeat the details already covered in Deliverable D2.1b [2]. The new material covers a review of existing security standards and tooling to support related risk management methodologies, which were not extensively reported on last time.

Section 3 describes updates to the SERSCIS core model. This includes two updates that were removed in the final version to avoid adding too much computational complexity (opportunity rules and detection of unsuccessful as well as successful threat activity). Updates that were retained in the final model addressed the factoring of threats to separately model primary and secondary effects, added modelling of beliefs about future asset behaviour to the existing model for observations of current asset behaviour, and reduced the chances of misinterpretation by a change in one relationship label.

The SERSCIS dependability model is a specialisation of the core model representing the types of assets, threats, behaviours and controls needed to describe dynamically composed, multi-stakeholder service oriented systems. Section 4 describes in detail the updates made this year, including late changes made during the validation testing phase.

Abstract system models represent a design-time view of specific applications prior to composition from dynamically composed, multi-stakeholder service oriented systems. There is no detailed description of an abstract system model because that was developed in WP7 and will be covered in deliverable D7.3 [3]. WP2 provided some semantic modelling tools to help non-experts create abstract system models, and these are described in Section 5. A brief discussion on the updates made to the concrete model generation using stream reasoning is also described here in Section 0. Full implementation details will be described in Deliverable D5.2 [4].

Updates to the Semantic Access Modelling (SAM) tool are described in Section 7.

The report finishes with a brief summary of conclusions, followed by a list of references.

2 State of the art

In D2.1b [2] we have described the state of the art from a system dependability modelling point of view. The Secure Business Austria Security Ontology (SBA) [5] has a class structure similar to the dependability models in SERSCIS. The core structure is populated directly with instances derived from the IT Grundchutz manual and ISO 27001. The SBA ontology uses OWL axioms to capture security knowledge in the form of constraints on these instances. Controls to mitigate the threats are also introduced in SBA. However, the complete list of threats in SBA does not seem well structure and comprehensive, and most are not obviously derived from the IT security guidelines referenced. Moreover, there are inconsistencies in the use of OWL classes and instances. Instances are used to describe generic concepts instead of OWL classes. The other ontologies reviewed in the previous deliverable included the Ontology of Information Security [6]which has a more extensive treatment of the kinds of assets, threats and countermeasures that exist than the SBA Security Ontology. Furthermore, these are modelled as subclasses, not as instances. We also reviewed the NRL security ontology [7] which focusses on describing the security properties of Web services. In this deliverable we will investigate the state of the art with respect to existing risk management standards and methodologies.

2.1 Risk Management Standards

State of the art risk management methodologies such as COBIT [8], and ISO 27005 [9] are based on analysis of information security risks for a given system design and configuration. The analysis allows system vulnerabilities to be detected, and associated risks quantified. If the risk is too great to be accepted, strategies are defined to reduce the risk (using security controls), avoid the risk (by not using the vulnerable system feature) or transfer the risk to someone else (by outsourcing system functions, or insuring against potential losses).

For example, ISO 27005 standard provides guidelines for information security risk management in an organization. The risk management process includes context establishment, risk identification, risk analysis, risk evaluation and risk treatment. Monitoring and communication are also included across all these phases. A risk is a combination of the consequences that would follow from the occurrence of an unwanted event and the likelihood of the occurrence of the event.

The risk identification phase is to determine what event could happen and cause potential loss to the organization objectives. This phase starts with asset identification. An asset is anything that has value to the organisation and which therefore requires protection. An asset owner should be identified to provide responsibility and accountability for the asset. The asset owner may not have the property rights, but has responsibility for its production, development, maintenance, use and security as appropriate.

As a first step, therefore, an organisation needs to identify its assets. According to the standard, there are two kinds of assets: primary assets (business processes and information), and supporting assets on which the primary assets rely (e.g. hardware, software, networks, personnel, sites, etc). Asset valuation and impact assessment steps follow in order to understand the importance of the assets to the organisation and organisation objectives in terms of confidentiality, integrity, authenticity, non-repudiation, availability and replacement cost.

Once assets are identified, one proceeds to threat identification. Threats should be identified generically and by type (e.g. unauthorized actions, physical damage, technical failures) and then where appropriate individual threats within the generic class identified. Some threats may affect more than one asset and may cause different impacts depending on which assets are affected. Asset vulnerability identification (including technical vulnerabilities) and risk assessment phases are then carried out. Risk identification consists of matching the threat to the vulnerability. The risk treatment phase includes employing controls so that residual risk is judged acceptable.

ISO27005 thus depends on the identification of all significant risks to one's assets, and awareness of what controls are needed to address vulnerabilities to each type of risk. This is not easy for anyone who is not an expert in security. The ISO27005 standard therefore refers to other standards (ISO 27001 [10] and ISO 27002 [11] which describe a general approach identifying risks based on checklists of typical security control objectives (i.e. vulnerabilities) and possible controls. Other standards such as RFC 2828 [12] take a more direct approach by describing threats directly, in terms of possible threat actions (actions or circumstances that create a threat) and consequences (the effect of the threat action on the system). In the final phase of the SERSCIS project, the RFC 2828 classification of threats was used as a starting point for overhauling the SERSCIS threat models. This makes it far less likely that any threats will be overlooked, although SERSCIS did not attempt a complete coverage. The final SERSCIS dependability model focuses on on threats related to service performance, information and communication security, and some aspects of physical security, focusing on threats that could be simulated in the simplified Vienna Airport emulator test bed.

Finally, ISO 27005 defines a monitoring, review and improvement phase, after the system has gone into service, during which risks as well as the risk management system should be continuously monitored and reviewed to keep any change under control. In practice, ISO 27005 (and various tools and methodologies that use a similar approach) assume this is a manual process, which may at some stage involve a fresh reanalysis of risks by security experts and modification of the system. This is where SERSCIS goes beyond the state of the art, by explicitly supporting run-time changes in agile, multi-stakeholder, service oriented systems. The SERSCIS approach is based on semantic modelling at design time, and uses machine reasoning at run-time to construct and analyse a model of the running system, allowing dynamic changes to be taken into account, including dynamic changes in the likelihood of specific threats being active.

2.2 Risk Management Tools

Standards like ISO27005 describe the steps that should be used to analyse and manage risks, but exactly how these steps are executed is left to the reader. However, there many tools, languages and methodologies have been devised to help users carry out these steps.

2.2.1 MEHARI

MEHARI [13] proposes a knowledge base of risk scenarios that can be used by organisations. Like ISO 27005, MEHARI classifies assets into primary assets and secondary (or supporting) assets. Primary assets correspond to functional needs while secondary assets correspond on a physical and concrete level to the means required to meet functional needs. However, unlike ISO 27005, the primary assets are classified into three categories services (IT Services and general), data necessary for the services to function and management processes.

Supporting/Secondary assets are the different forms the primary asset may take or the different contingencies on which that asset may depend. They are described by types of means required to meet the functional needs. Examples of secondary asset types under the categories services, data and management processes:

- Category Services: personnel and service providers necessary for the service (internal and external), service support hardware equipment, software support media, accounts and means necessary to access the service, premises
- Category Data: logical entities: files or databases, transiting data packets or messages, physical entities: media and devices
- Category Management processes: internal guidelines and procedures, personnel and service providers necessary for management processes

MEHARI contains a detailed list of attack scenarios and controls. MEHARI is scenario based which means that the threats are defined within a scenario of circumstances (activity, time, place, e.g. ERASURE during maintenance activity). The threats provided in the MEHARI knowledge base are linked to the primary and secondary assets types. Not all of the MEHARI secondary assets and threats are considered in SERSCIS (e.g. configuration files). This is because the generic dependability model in SERSCIS uses a higher level of abstraction, focusing on the agile SOA aspects rather than the detailed implementation of each service. However when SERSCIS is deployed in real system, these secondary assets become relevant and relevant risks should be considered. It is possible to assign these risks to the relevant abstract entity in SERSCIS.

2.2.2 CCTA Risk Analysis and Management Method (CRAMM)

CRAMM (CCTA Risk Analysis and Management Method) [14] is a risk management tool developed by UK government's Central Computer and Telecommunications Agency (became OGC in April 2001). CRAMM has been completely redeveloped by Siemens Enterprise Communications Ltd. to become an information security toolkit covering the following stages: asset identification and valuation, threat and vulnerability assessment and risk calculation, countermeasure selection and recommendation. CRAMM considers assets dependencies and provides a technique for incorporating the dependency knowledge into the asset valuation and risk assessment process. Moreover the dependency knowledge is used to identify the correct level that the countermeasure should be implemented at (e.g. on the physical asset where IT equipment are stored, on the host where the services are deployed).

In SERSCIS, the dependency model is the starting point to deal with asset dependencies. However the dependency knowledge is incorporated into specialised risks (e.g. resource malfunction, too few resources). A technique similar to the CRAMM threat and countermeasure selection could be used in SERSCIS. Indeed, the SERSCIS approach could be used to encode the CRAMM knowledge base, allowing semantic reasoning to be used. However, this would mean that SERSCIS must reconstruct the CRAMM knowledge base from scratch as it is not available for the public.

2.2.3 The Risk IT framework

The Risk IT framework [15] considers that IT-related risk is not only a component of operational risk but underlies all categories of business risk (strategic, environmental, credit, operational, compliance). It considers IT risk to be a business risk associated with the use, ownership, operation, involvement, influence and adoption of IT within an enterprise. A process model is proposed divided into three domains containing three processes: risk governance, risk evaluation and risk response.

The risk governance section focuses on the responsibilities of the stakeholders in the IT risk management process and defining a unified vision of the risk within the organisation.

The risk evaluation focuses on description business impact and risk scenarios. The Risk IT framework requires IT risks to be translated/expressed in business-relevant terms but doesn't prescribe any single method. Multiple examples are provided including COBIT information criteria (effectiveness, efficiency, confidentiality, integrity, availability, compliance, reliability), COSO ERM (strategic, operations, reporting, compliance), FAIR (Productivity, Response (cost of), Replacement (cost of), Competitive advantage, Legal, reputation).

Asset/resource: an asset is any object of value to the enterprise that can be affected by the event and lead to business impact. A resource is anything that helps to achieve IT goals. Assets and resources can be identical e.g. IT Hardware is an important resource because all IT applications use it and is an asset because it has a certain value to the enterprise.

Assets/resources include:

- People and organisation
- IT processes or business process (including COBIT processes)
- Physical infrastructure
- IT infrastructure including computing hardware, network infrastructure, middleware
- Other enterprise architecture components including information and applications

All risk scenarios can be classified under one or more of these categories:

1. IT benefit/value enablement risk: associated with missed opportunities to use technology to improve efficiency or effectiveness
2. IT programmes and project delivery risk: associated with the contribution of IT to new or improved business solutions.
3. IT operation and service delivery risk: associated with the operational stability, availability, protection and recoverability of IT series.

Risk IT provides as starting point a set of 36 high level risk scenarios. An IT risk scenario is a description of IT–related event that affects resource/asset and leads to a business impact.

The Risk IT COBIT examples provide risks scenarios similar to those identified in SERSCIS. However, the threats in the SERSCIS dependability model are more detailed and specialized (e.g. OneOffResourceInaccuracy). Considering the level at which Risk IT operates, it is sensible to consider negative as well as positive impact risks (opportunities). This is not applicable in SERSCIS where we focus on risks with negative impact only.

2.3 Risk modelling languages

2.3.1 Secure Tropos

Work by [16] uses an extension of the Secure Tropos language to support the modeling of security risks. The domain model is mainly structured around three groups of concepts: asset-related concepts, risk-related concepts and risk-treatment related concepts. Further security criteria for each of these assets are identified in terms of confidentiality, integrity and availability. This work is an extension of the work on Secure Tropos, and includes the development of syntactic, semantic and methodological extensions that would support security risks and their counter measures. This representation is in a diagrammatic in nature and is used to present abstract syntax elements for risk modeling and the rules on how these can be combined together.

The domain model in [16] is similar to the core ontology model we use in SERSCIS (consisting of Threat, Asset and Control). The risk modelling approach we used during the initial brain storming session includes features found in [16] and [17], however, in SERSCIS we use OWL ontologies to model Core, Dependability and Abstract system models in addition to the diagrammatic representations. This allows our models to be much more expressive (due to expressive nature of OWL syntax) and encoded in a way such that existing rule based languages (e.g. SWRL) and DL reasoners (e.g. Hermit) can be used with these OWL ontologies for the automatic identification and mitigation of threats.

2.3.2 Graphical risk modelling

Matulevicius et al [17] present work on a graphical approach to identify, explain and document security threats and risk scenarios. A graphical notation was developed to perform the five phases needed for security analysis 1. Context establishment, 2. Risk identification, 3. Risk estimation, 4. Risk evaluation and 5. Treatment identification. Diagrams are created during each of these steps (similar to UML models) under the guidance of a domain expert. We have followed a similar approach for the identification of threats and their mitigation strategies. However, the work described in [17] does not go beyond the modelling phase (diagrammatic modelling). The novelty of our SERSCIS system modelling approach is that we use an abstract modelling approach (OWL ontology based) to address the challenge faced in adaptive systems (where the composition of the system is not known in advance). Further, we go beyond the modelling stage and integrate our ontologies into the runtime dynamic stakeholder system. The concrete model (instance information) is automatically generated at runtime depending upon the current composition of the system.

2.3.3 Semantic modelling and machine reasoning

In most methodologies, the effectiveness of control strategies must be monitored, and if necessary the analysis is revisited, e.g. if new vulnerabilities are discovered, or threats prove more likely than expected. However, the approach is essentially a design-time approach, in which changes are effected by amending the system design (e.g. adding new controls), having security experts analyse the new design to check the changes will have the desired effect, and then adding controls to the system. In a dynamically composed service-oriented architecture this approach is not sufficient for two reasons. There is no conventional system design including controls: the system composition changes during run time as services are added removed or replaced, each service having its own controls that may differ from others of its type. And there is no time for a conventional security expert analysis of system changes that occur dynamically.

One approach that could address these problems (unknown composition of dynamic multi-stakeholder systems) is to use machine reasoning to analyze risks, so this can be done rapidly whenever the system composition changes. There is an existing body of research into how one might create semantic models of a system to support such an automated analysis, though with the motive of capturing security standards and expertise so tools can be developed to support non-experts. A useful overview is provided by [18]. For example, the NRL Security Ontology [7] provides a way to describe the security properties of Web Services, which was later used as a starting point for a Web Service vulnerabilities ontology [19]. The Ontology of Information Security [6] by Herzog et al describes a system in terms of assets, vulnerabilities and threats, so making the link to system risks (via threat models). However, this ontology uses N-ary relationships making it hard to deduce security properties via machine reasoning.

The Security Ontology [5] from Secure Business Austria (SBA) uses only conventional RDF relationships, and captures security threats and controls from the German IT Grundschutz Manual [20] , so providing a way to model systems with common threats and control strategies. The SBA approach goes a long way towards the goal of capturing security expertise in a form that can be reused (with supporting tools) by non-experts. However, this ontology describes only deployed systems and security controls, and cannot be used to create an abstract system model for a dynamically composed system whose concrete composition is not known at design time. The SBA approach also makes extensive use of Web Ontology Language (OWL) instances, which makes it hard to cater for multi-stakeholder systems where it is often necessary to attach different properties to the same threat depending on which stakeholder or sub-domain is targeted.

The approach used in SERSCIS is based on a reformulation of the Security Ontology approach, designed specifically to handle agile, multi-stakeholder, service oriented systems in which the model has to be created before the system composition is known. For this reason, all the OWL modelling ontologies in SERSCIS are based on OWL classes only. The concrete system model is not created until run-time, when instances of these classes are automatically created based on system monitoring data. The knowledge captured in the class axioms and other rules based on the class structure can then be applied at run-time using machine reasoning techniques. See Section 3.1 for details of the final SERSCIS modelling ontologies.

2.4 Risk management in Air Traffic Control

SERSCIS is aligned with the ISO 27005 approach based on an asset definition and classification process (into primary and secondary assets), but SERSCIS does not explicitly support asset valuation as proposed by ISO 27005 as a way to determine the impact of risks to the assets. Instead, SERSCIS assigns a threat severity level to each system-specific threat class indicating its overall impact on the system as a whole. This is a design-time (i.e. static) procedure, so it was possible to simply adopt the SESAR approach for threat impact severity [21] as published in 2011 (i.e. while the proof of concept modelling approach was being developed). The SESAR procedure involves identifying the secondary assets affected by each threat, and deriving a rating from the importance of the primary assets supported.

SERSCIS extends the SESAR approach in two important ways:

- threats are described by a generic dependability model ontology created by security experts, which can be automatically extended by a system designer to capture threats in a specific system;
- the run-time phase is explicitly addressed by creating a system model automatically from monitoring data, and using it to detect active threats and system vulnerabilities.

The SESAR methodology doesn't use a pre-defined checklist of threats, although it would be possible to start from checklists derived from standards such as ISO 27001 [9]. SERSCIS goes beyond this by automating the application of such a checklist to the specific system of interest. SERSCIS then provides run-time assessments of system vulnerabilities (e.g. if new assets join the system, or controls are deployed or deactivated), and replaces a priori (and hence static) threat likelihood estimates from security experts with threat activity levels computed at run-time from observed system behaviour.

3 SERCIS core model refinement

3.1 Starting point: the SERSCIS Proof of Concept model

The proof of concept core ontology was represented graphically in SERSCIS Deliverable D2.1b [2] as shown in Figure 1.

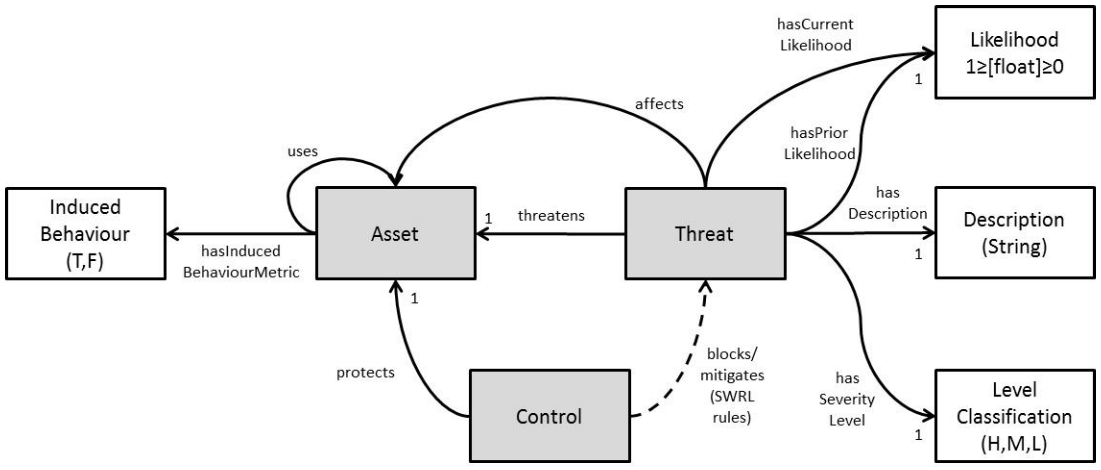


Figure 1. Proof of concept: initial core ontology

This shows the main ideas embodied in the ontology:

- we describe assets, threats and controls as OWL classes;
- assets may have associated metrics specifying the presence or absence of threat-induced behaviours;
- threats have a human readable description, an impact severity (based on the value of assets that would be compromised), and prior and current likelihood ratings (probability levels).

There are two aspects of the proof of concept core model that are not shown properly in Figure 1. The first is the relationship between controls and threats. The dashed arrow does not represent a conventional OWL relationship. It indicates one or more SWRL rules to classify threat instances as mitigated or blocked, based on the presence of adequate controls. In fact, these rules do not simply link controls with threats – they refer to controls protecting assets affected by a type of threat.

The second aspect which is not shown at all in Figure 1 is the relationship between threats and threat-induced behaviours, which is used to determine the current likelihood. The prior likelihood represents the expected probability that the threat is active in the absence of any evidence for or against. It is called the 'prior' because it is the probability we would assign in advance (i.e. at design time) so its value can be specified in each (system-specific) threat class definition. The current likelihood is the probability that the threat is active given the available evidence. Its value can only be deduced at run-time for each threat instance, when we have evidence (the presence or absence of behaviours) from which to determine whether the threat is indeed active.

This determination is made using Bayesian inference. If the system is operating normally, no threat-induced behaviours will be detected, and the current likelihood will be lower than the prior likelihood. If the system has threat-induced behaviours, threats that cause those behaviours may have a higher current likelihood, and threats that cause only those behaviours will have the highest current likelihood, indicating that they are probably active. The Bayesian inference approach requires information about how likely it is that if a threat is active, it will cause a given induced behaviour to be detectable at a given asset, and how likely it is that a given behaviour will be detectable at a given asset in the absence of any active threat. These are known as the behaviour causation rates and the background occurrence rates, and in the proof of concept implementation they were supplied to the Bayesian threat activity estimator via a separate input file.

Adding control rules and behaviour causation and background rates to Figure 1 gives the complete model shown in Figure 2 (the missing elements and relationships are highlighted):

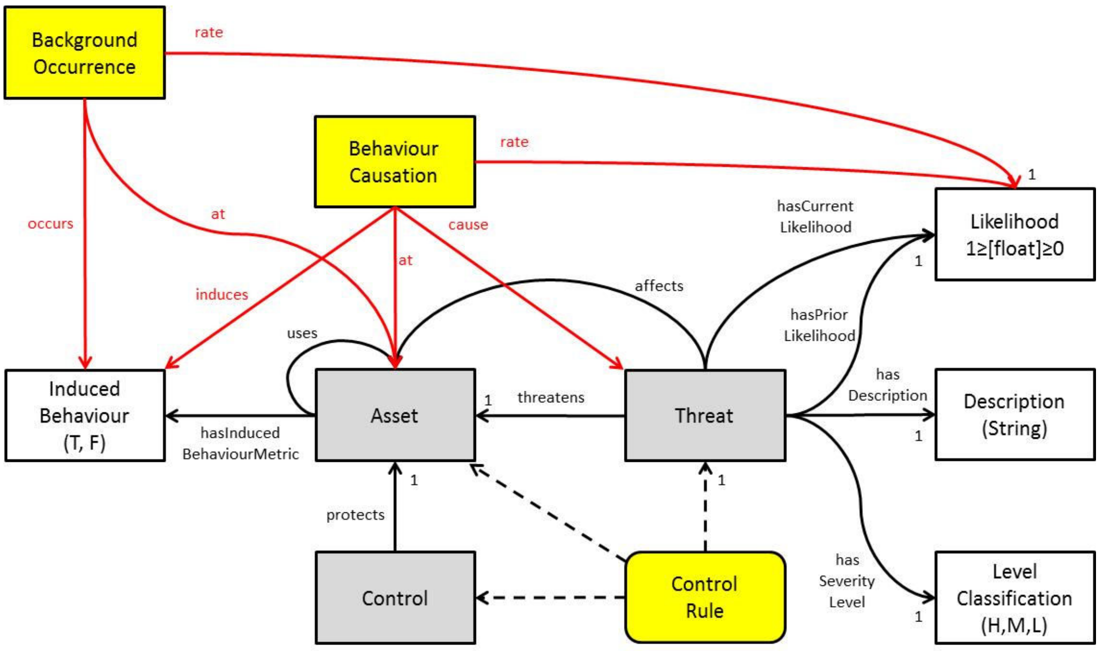


Figure 2. Proof of concept: complete core ontology

3.2 Core model enhancements

3.2.1 Behaviour monitoring and model editing

The model from Figure 2 could have been encoded in OWL and SWRL without using auxiliary input files. The reason we did not do this is because we expected the behaviour causation and background rates would need tuning during the evaluation experiments.

The behaviour causation and background rates should represent the likelihood that the behaviour will be detectable in the system (in our case an emulated airport). Unfortunately, we had no idea how likely it would be that if we injected emulated threats, the corresponding emulated behaviour would be detectable, given the level of emulated monitoring. The initial causation and background rates for each threat class were based on educated guesswork. Once the first experiments were carried out, it became clear that some of our monitoring policies were insensitive or unpredictable. ServiceGroup and ResourceGroup assets caused particular difficulties, as the behaviour of the group was defined in terms of the proportion of its members with the behaviour, and the threshold was often difficult to set when the number of members could be a run-time variable.

To update these values during the validation experiments using Protégé as an editor was too difficult. Instead, the values were stored in an Excel spread sheet which is far more easily updated. A suitable text input file for the Bayesian estimator was generated from this each time the values had to be changed. This was far easier than changing the ontology itself, even though it made it harder to keep everything consistent. Initially it was planned to transfer all the behaviour background and causation rates into the OWL model, so keeping everything together and reducing the chances of version mismatches occurring during validation. However, this was not done mainly because adding the rates would make the ontology mode complex, and because no better tools than Protégé could be developed for maintaining the resulting ontology during this period. Note that tools were created to help system designers create abstract system models, but these focused on the generation of system-specific asset and threat classes only.

In the final SERSCIS model, the concept of behaviour causation was refined slightly, and the corresponding rates were renamed behaviour correlation rates. This is because the event caused by the threat is actually the detection of behaviour by an asset monitor – the behaviour itself may be caused by the threat, but it may also be part of the threat action. We also had to consider whether the correlation should be taken into account by the threat activity estimator. In some cases, the threat may simply model the knock-on consequences of asset compromise which itself has another cause. In those cases, detection of this compromise should be correlated with its original cause and not its secondary effects (see Section 3.2.2).

This analysis also led to a change in the core model relationships. In Figure 2, a threat 'threatens' one asset and 'affects' potentially many assets. Any asset whose behaviour is correlated with the threat should be affected, so that the machine reasoning algorithms used in SERSCIS can determine that its behaviour should be taken into account when estimating the likelihood that the threat is active. Similarly, any asset whose controls contribute towards blocking or mitigating the threat must also be considered affected, so the machine reasoning algorithms know to take its controls into account when determining whether the system is vulnerable to the threat. The use of the label 'affects' for this relationship began to cause misunderstandings among the model developers, given that a correlated behaviour may not actually be caused by the threat, and the presence of controls certainly isn't. To avoid such misunderstandings, the relationship label was changed from 'affects' to 'involves', which conveys better to human readers that the asset plays some role in the threat or its control strategy.

3.2.2 Secondary effects

The most important change in the core model during the final year of SERSCIS was the introduction of the concept of a secondary effect. This was done initially to simplify the process of capturing the asset behaviours that could be caused by a threat, which didn't work as well as initially hoped.

For example, suppose we define "ResourceUnderperformance" to be a threat whose cause is some internal problem at a Supplier of an upstream Resource, which leads the Resource to underperform. This underperformance is behaviour caused directly by the threat, so it is easy to understand that the Resource is an affected asset and assign a causation rate for the behaviour. The only question is how sensitive the monitoring probe compared to the strength of the effect, i.e. how likely it is that the probe would detect underperformance at the level caused by this particular threat. We're not very interested in unknown internal supplier problems that don't cause a significant detectable effect, so we would probably set the causation rate close to 100% in this case.

However, if a Resource underperforms then it may be enough to trigger underperformance in the associated ResourceGroup. In the original model, if the ResourceUnderperformance threat does not also have a causation rate for this behaviour, then the Bayesian activity estimator will compute a much lower likelihood that the threat is active. This happens because the estimator must assume that the ResourceGroup underperformance is coincidental, so the likelihood that it occurs is the background or spontaneous rate for that behaviour, which is typically close to 0%. To avoid this situation, such knock-on consequences of a threat had to be explicitly included, i.e. the ResourceGroup has to be considered as affected by the ResourceUnderperformance threat, and a non-zero rate inserted for causation of ResourceGroup underperformance by this threat. The problem is that if the ResourceGroup is underperforming then this might lead to ServiceGroup underperformance, which in turn may affect Customers, etc.

Thus a simple threat to a Resource becomes a very complex threat in the dependability model, affecting a whole chain of assets, with behaviour causation rates linked to all of them. What is worse, the further one gets from the original cause, the harder it is to determine what the behaviour causation rates should be. This is because at each step in the chain additional factors have to be considered (e.g. how many other Resources are there in the ResourceGroup, how critical the performance of this type of Resource is to the ServiceGroup as a whole, how likely it is that each individual Customer will be affected, etc).

Ideally, we would treat these knock-on consequences as separate threats, with their own behaviour causation rates. So if the Resource and ResourceGroup were both underperforming, the Bayesian activity estimator should conclude two threats were active: the original ResourceUnderperformance threat which does not itself explain ResourceGroup underperformance) plus a second threat to the ResourceGroup that does.

We couldn't use this approach with our proof of concept implementation because the Bayesian estimator ran too slowly if we included samples in which multiple threats are active. To keep the run-time within bounds, the estimator sampled only one active threat at a time. Since neither threat alone could explain the observed behaviour of both assets, neither would be considered very likely.

It is possible to optimise the Bayesian estimator and reduce its run-time, but that only cuts the time needed to process each threat activity hypothesis. It quickly became clear that a chain of secondary effects might involve many threats, especially once physical assets were added to the dependability models. No amount of algorithm optimisation could compensate for a power-law increase in the number of hypotheses that must be sampled if we are to include these multi-threat explanations. What we need is a way to recognise which threats are secondary effects before Bayesian activity estimation, so they can be considered active in every combination of threats sampled.

One way to do this is to note that secondary effects should always be caused by an observable behaviour in an observable asset. If we include threat causes in our models, we could add a SWRL rule for each class of threat that might be secondary. The rule would then check if the causing asset has the behaviour that represents the threat, as shown in Figure 3.

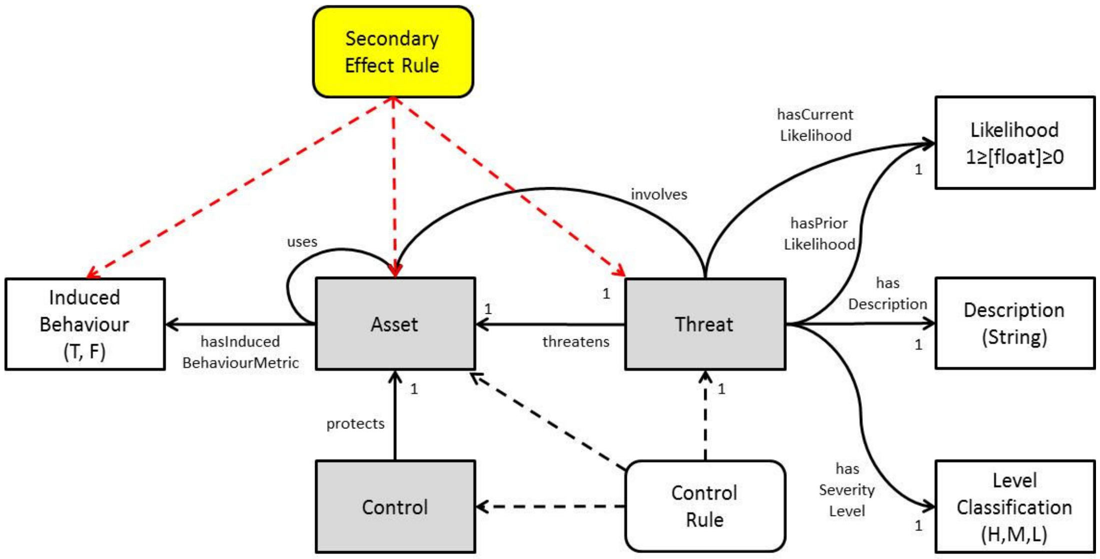


Figure 3. Threat causes and secondary effects

Note that the control and secondary effect rules should only refer to one Threat class (since they are supposed to classify one Threat instance at a time). They should also refer only to assets that are involved in the threat, and to controls that protect those assets or behaviour observations about those assets. This restriction makes it possible to perform incremental updates in the concrete model if an asset changes its status, which was important to reduce computation in run-time machine reasoning.

An example of such a rule would look something like:

```
ResourceGroupUnderperformance(?t) ^ Resource(?r) ^ ResourceGroup(?g)^
Underperformance(?b) ^ involves(?t,?g) ^ involves(?t,?r) ^
hasBehaviour(?r,?b)^ hasValue(?b,true)→ SecondaryThreat(?t)
```

Note that this approach doesn't directly chain threats together – there is nothing here that says ResourceUnderperformance could cause ResourceGroupUnderperformance. The presence of secondary effect depends on asset behaviours only. Thus if ResourceGroupUnderperformance doesn't actually cause the ResourceGroup to underperform (i.e. any underperformance is not severe enough to be classified as compromised behaviour), then there will be no further knock-on consequences for the ServiceGroup or Customers, etc. Moreover, ResourceGroupUnderperformance will be triggered as a secondary effect if a Resource underperforms for some other reason.

If a threat instance is classified as a secondary threat by such a rule, then that threat can automatically be considered active, leaving the Bayesian activity estimator to sample only the possible primary threats. In the above example, the activity estimator would assume ResourceGroupUnderperformance was active, and accounts for some of the observed behaviour (specifically any underperformance observed in the ResourceGroup). It would sample other threats to find how likely it is that they could explain the remaining behaviour (in this case, underperformance of the Resource), and so determine which threat(s) are likely to be the primary cause of the overall observed behaviour.

This idea of secondary threats can also allow us to define threats with a common cause. For example, in our proof of concept system there is one ACISP inbound resource per customer. We might be concerned about the threat of a DoS attack against each of these resources. However, if all the ACISP inbound resources were provided by a single ACISP stakeholder, we can't reasonably claim that DoS attacks on the individual ACISP inbound resources are independent. Instead, we can model a DoS attack on the ACISP stakeholder, which makes the ACISP unavailable. Then we create a secondary effect rule for DoS threats against individual ACISP inbound resources, making them all into secondary effects caused by the ACISP providing them being unavailable.

The inclusion of secondary effect rules is a major improvement in the final period, as it allows SERSCIS to perform root cause analysis at run-time. Previously it was not possible to distinguish root causes from secondary effects, and root causes could only be reliably detected if all of their consequences had been identified in advance and modelled accurately enough by a threat modeller. The new ontology allows secondary effects to be modelled without needing to know their root causes, making it far easier to create a model from which the root causes can be automatically inferred.

3.2.3 Security attributes and threat opportunities

In the original model, we did not include any constraints on network connectivity or physical access. In the new dependability model, we wanted to include network assets and spaces, and take account of whether a threat is possible given the network connectivity or physical proximity between the asset causing the threat and the target. It was expected that this could provide another way to reduce the number of threat combinations that has to be sampled by the Bayesian estimator, by excluding threats for which there is no opportunity to carry them out.

The way to do this is to use another threat classification rule, similar to the secondary effect rule shown in Figure 3. Whereas the secondary effect rule checks whether the cause is exhibiting the assumed behaviour, threat opportunity checking may also depend on security attributes (states). The concept of a security attribute can be defined as follows:

- it represents some persistent (security) property of an asset, which is initially 'True' or 'False' depending on the type of asset and what the attribute refers to;
- if a threat is carried out successfully against the asset, it may change the attribute's value;

- if a recovery action is performed by the asset operator, it may be reset to its original value.

Broadly speaking, asset behaviour represents an observation that it is misbehaving, whereas an attribute represents a belief as to whether it will misbehave in future.

For example, an attacker might want to deny service to an ACISP service by flooding it with requests. This is only possible if the attacker has access to the network the ACISP uses – if the attacker doesn't use this network then they have no opportunity to launch the attack. But if the attacker can corrupt another service that uses the ACISP's network, then they could launch an attack from there. The important point here is that the corrupt service might have been corrupted some time ago, so the fact that it is untrustworthy is a state, not necessarily a current active behaviour.

With this definition, we can then define an 'opportunity rule' as another SWRL classification rule as shown in Figure 4:

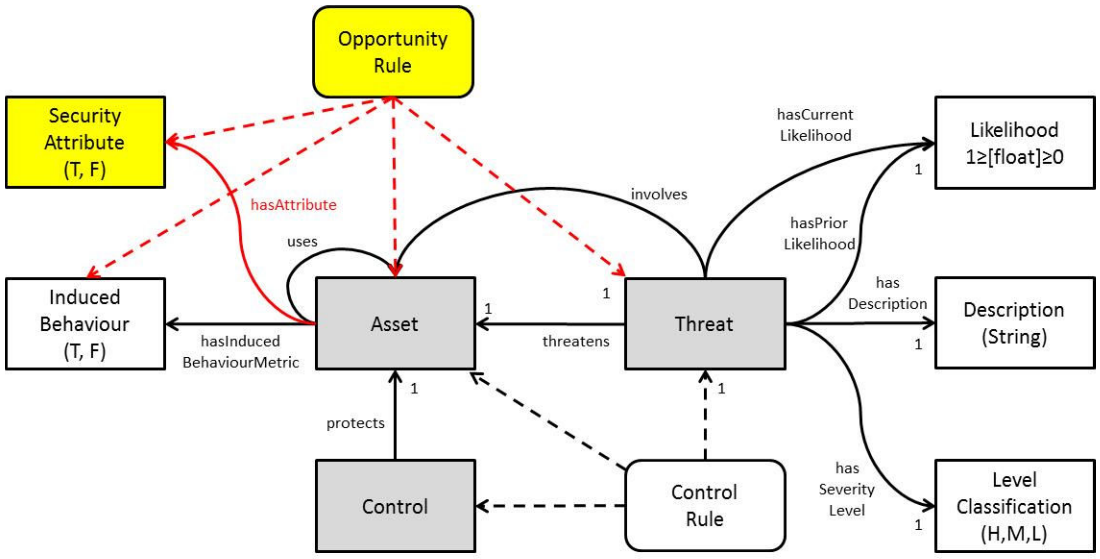


Figure 4. Opportunity rules (not used) and security attributes

An opportunity rule would look something like this:

```
DenialOfService(?t) ^ Service(?s) ^ Network(?n) ^ Asset(?a) ^
threatens(?t,?s) ^ involves(?t,?n) ^ involves(?t,?a) ^ Trustworthy(?u)
^ has Attribute(?a,?u) ^ hasValue(?u,F) ^ uses(?s,?n) ^ uses(?a,?n) →
PossibleThreat(?t)
```

Note that asset '?a' in this rule can be any asset that uses the same network as the target service. If the network were public, then the system model should include an 'Outsider' asset class that is by definition not Trustworthy and has access to the network.

If a threat is not classified as 'possible' by such a rule, then in principle no opportunity exists to carry out the threat. Of course for many types of threats, connectivity restrictions are not relevant, and for these threats a trivial rule must be used to classify them as possible.

There is a twist in the tail, however. With control rules, we know it is possible for a control to be ineffective, so a threat that is classified as 'blocked' might still be able to happen. The Bayesian inference approach therefore ignores whether a threat is blocked or mitigated, so if one of these threats is active, it will be detected. One can then infer that one or more of the controls believed to be blocking the threat must be defective. This type of inference is of great value to a system operator seeking to understand how to manage a threat, as it highlights a possible misunderstanding they may have about their system. One of the proof-of-concept test scenarios demonstrated this, in which a slightly flawed access control policy allowed an unauthorized update of information at the ACISP service. Identifying this apparently blocked threat as active showed that the access control policy must be incorrect, triggering an analysis using the SERSCIS Access Modeller which found the problem.

The same principle applies to threats for which there seems to be no opportunity. We shouldn't exclude them from the Bayesian inference procedure because if found to be active, that would imply the opportunity does exist, and that something is wrong with our assumptions, e.g. about the security attributes of some asset. However, some initial experimentation showed that the cost of opportunity rules is very high, and the benefits far lower than they are for control rules. The problem is due to the fact that if an attack that shouldn't be possible turns out to be active, we can't tell which of (possibly many) assets that have the required connectivity are no longer trustworthy.

Moreover, all these assets should strictly be considered 'involved' in the threat, as a change in any of those assets may signify a change in the status of the threat. After initial tests in July 2012, it was recognised that semantic reasoning is the most computationally expensive run-time analysis step. To reduce the run-time for this step, localised updating was introduced, so that when an asset changes its controls, behaviour or attributes, only threats that involve the changed asset are reclassified. But if large numbers of assets are involved in each threat, the savings from this approach become very limited. In the end, it was concluded that the cost of opportunity rules was too high compared with the benefits of having them, and they were removed from the final model.

3.2.4 Threat activity and success

Another idea that was removed in the final model was a refinement of behaviour correlation rates to distinguish between unsuccessful and successful threat activity. The idea here was to introduce two sets of correlation probabilities:

- the probability of a threat causing detecting of a behaviour if the threat is active but unsuccessful; and
- the probability of a threat causing detecting of a behaviour if the threat is active and successful.

The idea behind this approach was to determine automatically whether a threat may have changed the security attributes of a threatened asset, and thus potentially created new attack opportunities. This is shown in Figure 5:

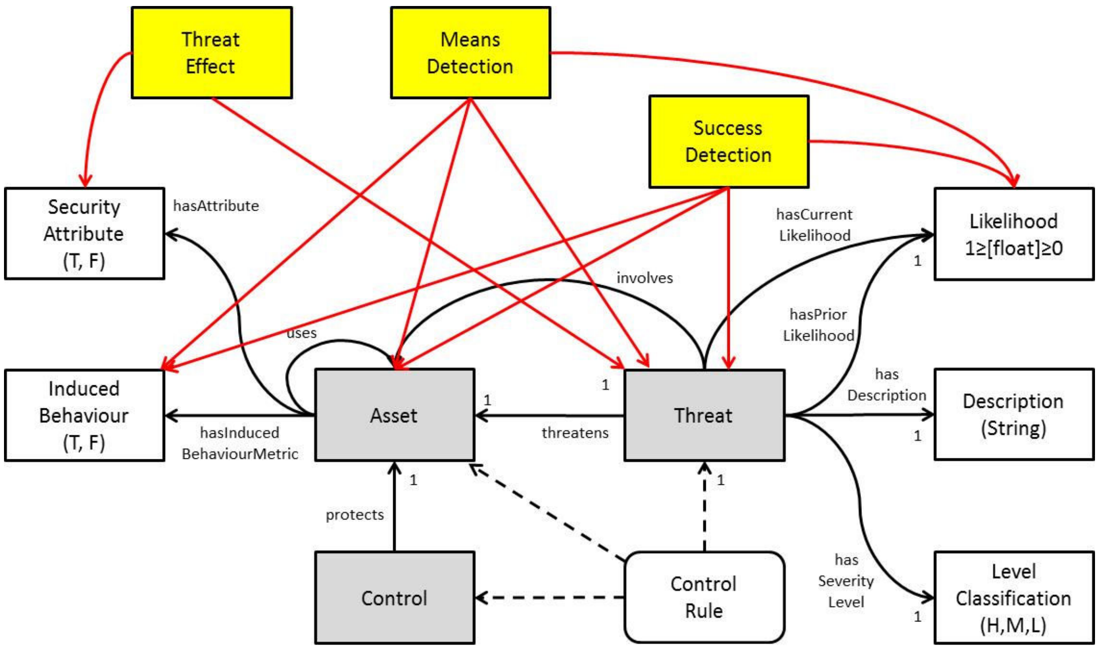


Figure 5. Threat activity and success detection and attribute updating (not used)

The problem with this approach is that it greatly increases the search space for the Bayesian threat activity estimator. Instead of considering two possible states for each threat (active or inactive), the estimator now had to consider three states (active and successful, active but unsuccessful, and inactive). In early summer 2012, it seemed that we would have to search for combinations of up to N threats (where N was at least 4), so scaling as 3^N instead of 2^N appeared to be a heavy price to pay. At the same time, it was becoming clear that opportunity rules were themselves of limited value, so the need to automatically update security attributes in response to a successful threat was diminishing.

In light of these concerns, the separate rates and threat effects were removed from the final model, leading to the simpler behaviour correlation model shown in Figure 6. Security attributes were retained in the core model, but the only attribute used is 'Trustworthiness', which has to be modified manually by the operator should they decide an asset that was assumed to be trustworthy can no longer be relied upon, or vice versa. In principle, the dependability model can refer to attributes as well as controls or behaviours in its SWRL control or secondary effect rules. This has not proven necessary yet (see Section 4), but the notion of trustworthiness is directly related to assumptions used in the SERSCIS Access Modeller (see Section 7).

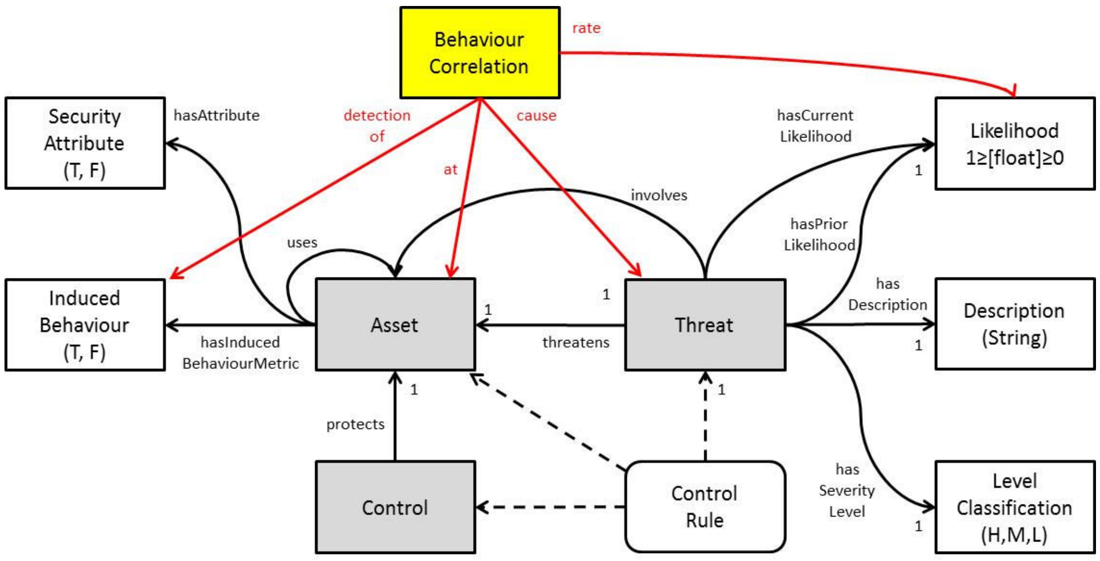


Figure 6. Final behaviour correlation rates model

3.3 Core model refinement summary

In summary, the main refinements of the SERSCIS core modelling ontology were:

- the label for a relationship between Threats and (potentially many) Assets was changed from 'affects' to 'involved', to better reflect human understanding of its meaning;
- secondary effect rules were introduced allowing threats to be classified as active where this is caused by observed behaviour of system assets;
- security attributes were introduced representing persistent beliefs about future asset behaviour, in contrast to behaviours which represent current observations of the asset.

Refinements intended to determine whether an opportunity exists to carry out a threat (using further SWRL rules), or to distinguish successful from unsuccessful threat activity were dropped during the summer of 2012. This decision was based on an assessment of the extra computational costs compared to the expected value of inferences produced.

The most significant change is the introduction of secondary effect rules. This allows threats to be modelled with only their primary threats, leaving (chains of) knock on consequences to be modelled separately. As such, they provide a basis for root cause analysis, which was previously only possible in SERSCIS if the modeller identified and correctly modelled all the primary and secondary effects of each threat.

4 SERSCIS Dependability model refinement

4.1 Starting point: the SERSCIS Proof of Concept model

The proof of concept dependability model developed in 2011 specialised the generic core model described above so it could be more easily used to model multi-stakeholder, agile service oriented systems. The main features of the dependability model are:

- subclasses of Asset representing services offered by the primary stakeholder, customers who use those services, and resources composed in order to deliver those services;
- subclasses of Threat representing different ways the value of these assets could potentially be compromised;
- subclasses of Control representing security or system management mechanisms that could be used to block these threats or mitigate their effects;
- subclasses of Behaviour representing specific types of adverse behaviour that could be induced by active threats.

The proof of concept model was very successful. It provided a basis for constructing models of specific systems, allowed different types of threats to be modelled and distinguished at run-time, and supported automation of several steps (threat vulnerability classification and activity estimation) and semi-automation of others (notably system threat class generation).

However, there were several shortcomings with the original dependability model:

- asset classes did not support modelling of physical or electronic connectivity, making it impossible to fully model network-based attacks such as denial of service attacks;
- threat classes were based on ad-hoc 'brainstorming' by security experts to identify possible mechanisms, with no systematic relationship to any existing methodology or standard.
- all the effects of each threat had to be captured, including knock-on consequences as well as the primary effects, making threat models complex and hard to parameterise;

The first of these shortcomings was noticeable at the 3rd Review: a denial of service threat that had been modelled (unsuccessfully) in Year 2 could not be handled by the new approach. The other two emerged from the validation work in late 2011, in light of the challenges posed by new scenarios that included network-based attacks.

To address these shortcomings in 2012, the dependability model asset classes were extended to model connectivity as well as logical services, the threat model was reorganised and aligned with the RFC 2828 security standard, and threats were factorised into primary and secondary effects using the extensions of the core model as described in Section 3.

4.2 Asset classes

4.2.1 Asset overview

In order to capture information pertaining to the execution of attacks on communications between assets, it is necessary to extend the generic dependability model to include those assets used for communication. Figure 7 shows the assets and asset relationships added to the dependability model.

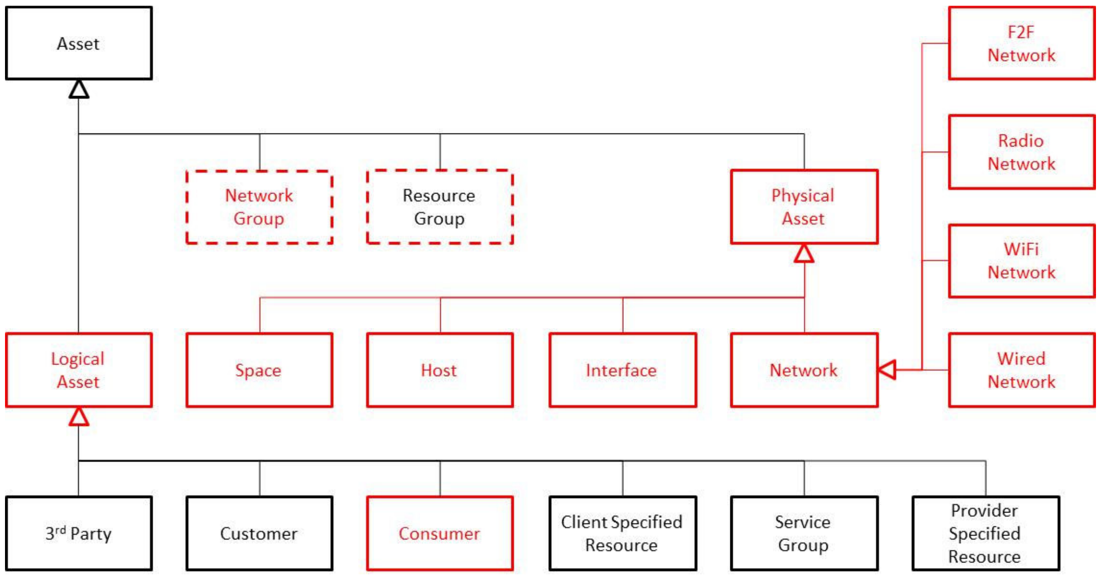


Figure 7: Extended dependability asset model

The service provider is the primary stakeholder for the model, and is sometimes referred to as the Provider for orientation purposes. However, the Provider is not represented explicitly in the model as their motives and behaviour are considered entirely reliable. This is because the model as a whole is supposed to reflect the Provider's motives and envisaged behaviour, so there is no need to associate these with a specific asset in the model.

4.2.2 Logical asset classes

This is not the case for other entities within the Provider's organisation and supply chain, as they may turn out to be unreliable and affect the delivery of highly dependable services. These are modelled as logical assets in the supply chain:

- the ServiceGroup representing processes carried out in-house by the Provider to deliver a service, including the use of software;
- the Customer representing external entities who buy (i.e. are accountable for the use of) the Provider's services;
- the Consumer represents anyone authorized by a Customer to use services on their behalf;
- the ClientSpecifiedResource representing external services composed by the ServiceGroup on instruction from a Customer or their authorized Consumer;
- the ProviderSpecifiedResource representing external services composed by the ServiceGroup at the discretion of the Provider, having agreed terms with their Supplier;
- any ThirdParty representing other users of either type of Resource, but who are neither authorised to use the Provider's services, nor to access Resources on the Provider's behalf.

The LogicalAsset was introduced as a superclass to capture the fact that these assets model processes or interactions that have value to the Provider, but without referring to where the processes operate or what mediates their interactions with the Provider. The only new LogicalAsset subclass is Consumer, which was added to provide a more explicit model of delegation among service users.

The relationships between these assets (some of which have changed due to the introduction of a better model of delegation) are shown in Figure 8:

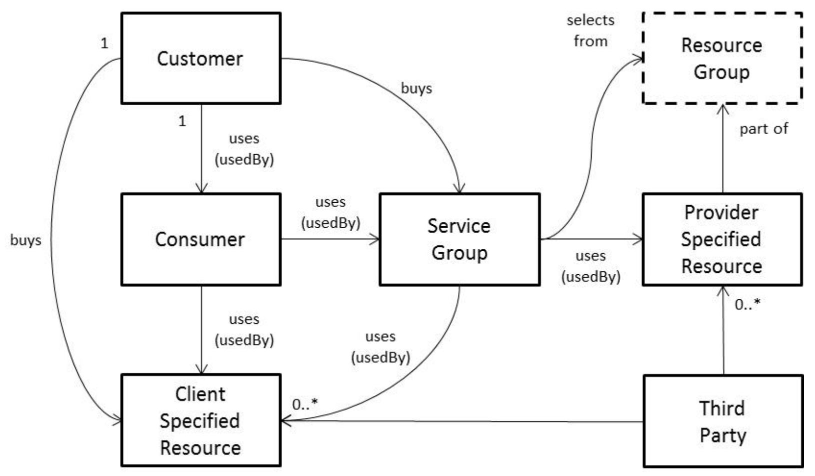


Figure 8. Logical asset relationships including dynamic resource selection

Labels shown in brackets represent the reverse relationship, going back along the directed arc. Note that ThirdParty assets typically use resources, but they may also be related to Customers or Consumers, though never directly to the ServiceGroup.

As in the proof of concept dependability model, logical assets represent the value of an interaction with the Provider, derived from the commitments and expectations agreed with their operators, rather than a process. Thus a Customer asset reflects the terms of a service level agreement (SLA) requiring them to pay for services they or their authorized consumers use and respect any limitations on usage, etc. Similarly, a ProviderSpecifiedResource represents the right to use an upstream service, as agreed with its supplier. A ServiceGroup represents the delivery of services to all Customers and their authorized Consumers (hence we refer to it as a Group as it aggregates all services of the relevant type delivered by the Provider). The LogicalAsset subclasses (except Consumer) thus represent the same things they represented before.

4.2.3 Physical asset classes

The PhysicalAsset class was then added as a superclass encompassing those assets that model where logical processes take place, and how they interact via spatial proximity or electronic communication networks. Subclasses of PhysicalAsset are as follows:

- Host: an asset representing a computational or organisational platform to support execution of processes associated with logical assets, i.e. every logical asset is supported by a Host;
- Network: an asset that can be used to communicate (send and receive information) between Hosts that are connected to it;
- Interface: a link between a Host and a Network through which communication has to pass;
- Space: a physically connected space in which people and equipment (i.e. Hosts or Networks) are located and from which they can be physically accessed.

Relationships between physical assets, and between Hosts and logical assets, are shown in Figure 9:

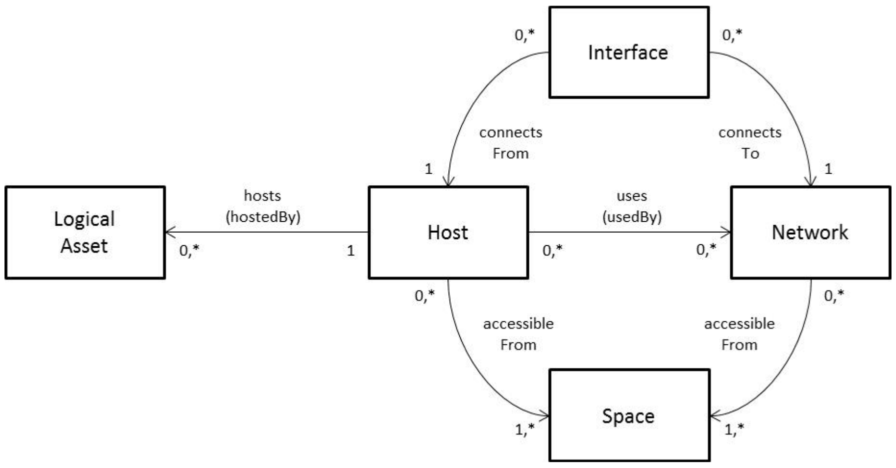


Figure 9. Physical asset relationships

In early 2012, the concept of a Host was intended to model only a computational server on which software could be executed to manage the information exchanges required in a logical asset. This was used in early testing up to a first wave of validation tests in July 2012. However, it became clear that in the context of the SERSCIS validation scenarios, such a limited interpretation was not useful. In the SERSCIS validation scenarios, logical assets represent processes that are carried out by people using physical equipment (trucks, consumable supplies, etc) and communicating face-to-face when co-located, as well as by software exchanging information over electronic communication networks. In the final model, a Host can be an organisation including people and equipment able to carry out physical processes in the real-world, as well as servers capable of running software.

Another refinement introduced after the initial validation scenarios was to add subclasses of Network representing different inter-host communication mechanisms that may be subject to different types of compromising attack. These appear on the right in Figure 7: face-to-face communication between co-located people is easier to protect against interference or corruption, while radio (voice) or WiFI (data) communication over radio frequencies is susceptible to electronic jamming, etc. Consideration of how one might carry out a denial of service attacks by either jamming a network or flooding a service endpoint led to the introduction of Interface assets. The Interface represents a local target for a network denial attack, allowing the threat model to distinguish between network-based attacks against one location or against the network as a whole. Interfaces are completely defined by their relationships to a Host and a Network, and so can be added automatically once the Host uses Network relationships have been specified.

Modelling the relationships between Spaces, Networks and Hosts allows us to represent potential attacks vectors in more detail. For example, to physically connect to or disrupt a network one must have access to it, i.e. one must be able to access a space from which it is accessible. Thus traffic interception or jamming attacks must involve a space (where the attacker is located) as well as the network targeted by their attack. Controls can be deployed at all points of this vector, e.g. physical security to block unauthorised access to restricted spaces, network controls such as firewalls, patching of an organisation's servers, etc. In the space model, neither hierarchical (space A contains space B) nor adjacency (space A adjoins space B) information is considered – i.e. all spaces are treated as interconnected. Adding relationships between spaces would allow modelling of attacks (and defences) that depend on the path taken by an attacker to get from an unrestricted space to a restricted one. This was omitted from the current model for the sake of computational simplicity, but it could easily be added if required.

4.2.4 Selection asset classes

Two asset classes from Figure 7 are neither physical nor logical assets. They don't represent the physical infrastructure or the processes or interactions supported by that infrastructure. They represent the ability of the Provider to choose between other assets.

The ResourceGroup asset class existed in the 2011 SERSCIS dependability model, and represents the ability of Provider (or strictly the process represented by a ServiceGroup) to choose between different ProviderSpecifiedResource assets when it is necessary to use (compose) one to deliver a service. This asset class also appears in Figure 8, including its relationship to the ServiceGroup indicating that the ServiceGroup can select from the ResourceGroup. For each type of ProviderSpecifiedResource distinguished by the ServiceGroup implementation (i.e. each system-specific subclass) there should be a ResourceGroup asset. If a Supplier provides an upstream service that can fulfil more than one function needed by the Provider, then it would be a member of multiple ProviderSpecifiedResource subclasses, and also part of multiple resource groups.

The NetworkGroup asset class is a new addition in 2012. It represents the ability of any logical asset to choose between different networks when exchanging messages with any other logical asset, as shown in Figure 10.

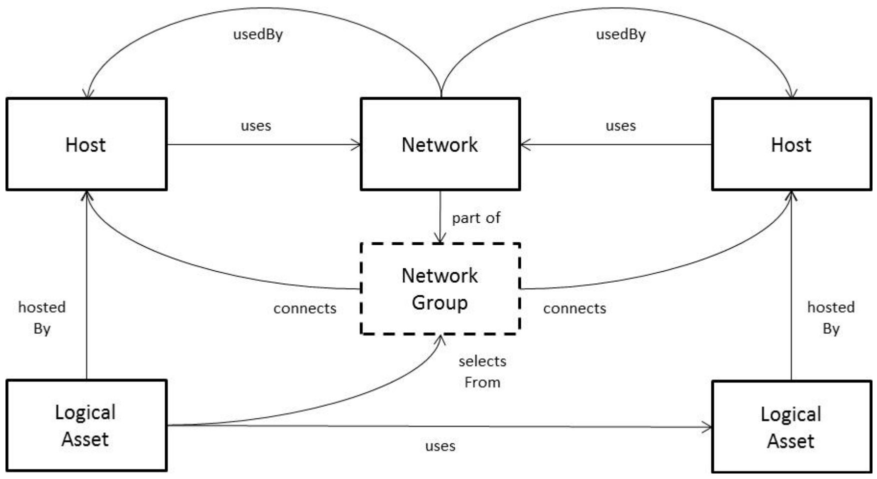


Figure 10. Dynamic network selection relationships

Like Interface assets, NetworkGroup assets are completely defined by their relationship to a pair of Hosts. Any network used by both Hosts is automatically part of the NetworkGroup. Consequently NetworkGroup instances and their relationships can be added automatically once the other relationships have been defined.

There is one special case of Figure 10, when a Host runs internal communications over a Network that is accessible outside the Host, e.g. from a Space. A typical example would be the use of a radio communication network within an airport, which could then be jammed from within the airport. In this situation, the same Host appears on the left and right of Figure 10, and the LogicalAssets on each side may also be the same (in which case there is no ‘uses’ relationship between them).

4.2.5 Inferred asset classes

It turns out that the above asset classes are not sufficient to unambiguously define some types of threats in which more than one asset of the same class may be involved. For example, if an attacker wants to interrupt communications between two logical assets (e.g. a ServiceGroup and a Provider-SpecifiedResource), then the threat must involve a network connecting the two Host assets where the logical assets are hosted. The problem with this is that in control or secondary effect rules, it may matter which host is which, but in OWL we can only say both are involved. Any SWRL rule to check the behaviour or control status of the involved Hosts would match with them both ways round. Thus a SWRL control could incorrectly classify a threat as blocked or mitigated if the right controls are deployed at the wrong host.

To get around this problem, it is necessary to further classify Hosts and also Interfaces based on what types of logical assets they are associated with. For Host assets, this is done by introducing a HostClassifier (a subclass of Thing), which has further subclasses defined by their relationships to logical assets, as shown in Figure 11.

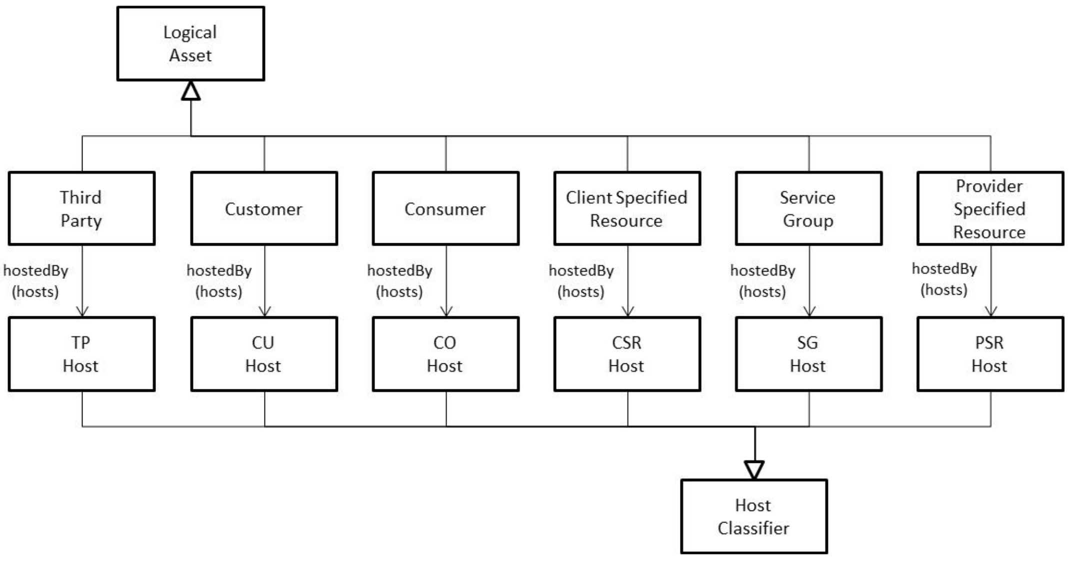


Figure 11. Inferred classification of Host assets

Each type of Host that appears in an abstract system model can be assigned by semantic reasoning to one of the HostClassifier subclasses shown. Thus each system-specific Host type is asserted to be a subclass of the dependability Host class (i.e. it hosts logical assets and uses Networks to support their communications), and inferred to be a subclass of one or more host classifier classes (indicating what types of logical assets it hosts). Note that this inference can be made based on system-specific classes, so it doesn't have to be done at run-time when system-specific host instances are created to model a running system.

Once the Hosts are classified in this way, a similar procedure is used to classify Interfaces, based on the inferred classification of the Host assets they are associated with. This is shown in Figure 12 – the InterfaceClassifier class is (like HostClassifier) a subclass of the OWL class Thing.

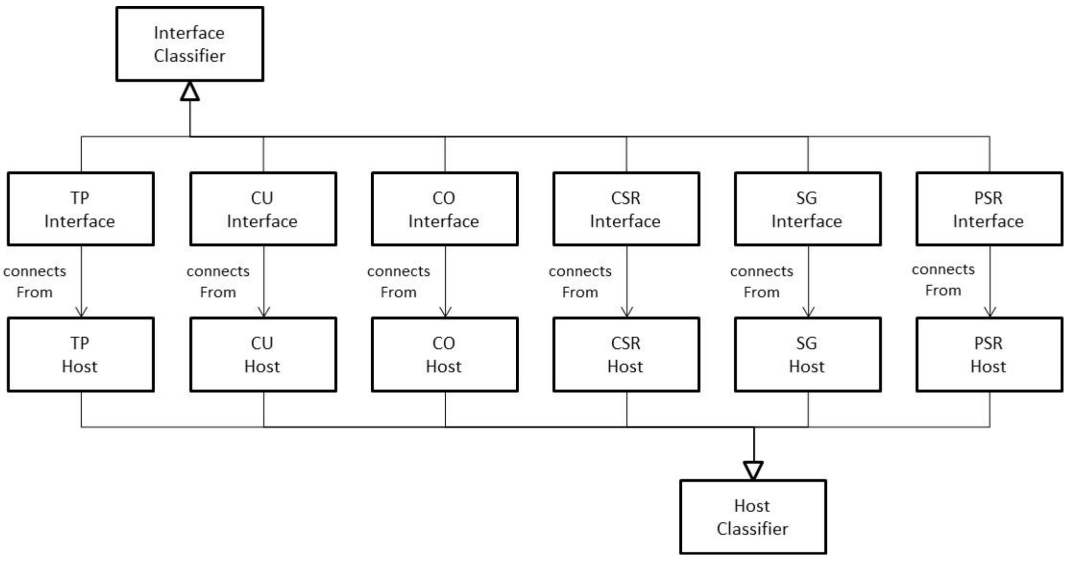


Figure 12. Inferred classification of Interface assets

Interface classification cannot be done in the abstract system model. This is because Interface assets do not have system-specific subclasses – they are too simple to require any customisation. System modellers usually don't bother to define Interfaces, but rely on automated insertion of Interface asset instances wherever they are needed in the concrete system model. These interfaces must classified at instance level before applying any SWRL control or secondary effect rules.

4.3 Induced adverse behaviour classes

The induced behaviours have changed only slightly since D2.1b.

The most important but possibly least visible change is to recognise that behaviours allow observations of whether an asset is compromised to be asserted as a relationship between the asset and the corresponding observation. Behaviours are therefore semantic concepts, and strictly not 'metrics' as stated in D2.1b. An observation whether an adverse behaviour is present or absent is based on metrics produced by system monitoring probes, but strictly speaking the observation is not a metric, but an assertion based on some (possibly trivial) analysis of the system metrics.

Apart from this refinement of what is meant by behaviour in the semantic model, the behaviours from the proof of concept model have been reinterpreted so they can also apply (where appropriate) to the new assets from Figure 7. There has also been one change, replacing the old 'indiscreet' behaviour which meant 'allowing inappropriate access to functionality or data' by two behaviours:

- indiscreet: allowing inappropriate access to data by any means; and
- promiscuous: allowing inappropriate access to functionality.

This slight change was prompted by an analysis of RFC 2828, where a distinction is made between inappropriate access to information held or used in a system, and usurpation of system functionality. The new behaviours are summarised in Table 1.

Behaviour	Meaning	Applies to											
		Space	Host	Network	Interface	ServiceGroup	Customer	Consumer	CSR	PSR	Third-Party	ResourceGroup	NetworkGroup
Dissatisfied	Interactions don't produce the expected mutual benefit.					Y	Y	Y	Y	Y		Y	
Inaccessible	The asset can't be accessed, e.g. due to over-restrictive security.	Y	Y	Y					Y	Y		Y	Y
Inaccurate	Information exchanged with the asset is too often inaccurate.		Y	Y		Y	Y	Y	Y	Y		Y	Y
Indiscreet	The asset allows inappropriate access to data.		Y	Y		Y	Y	Y	Y	Y		Y	Y
Over-committed	The asset is promising more than it can deliver.					Y							
Overloaded	The asset is interacting more than it should, or more than it agreed.		Y	Y	Y	Y	Y	Y		Y		Y	Y
Promiscuous	The asset allows inappropriate use of its functionality.	Y	Y	Y		Y			Y	Y		Y	Y
Unaccountable	The asset denies responsibility for its actions.						Y						
Unauthentic	The asset can't be relied on to be who or what it claims to be.		Y	Y		Y	Y	Y	Y	Y		Y	Y
Unauthorized	The asset doesn't have the rights it claims to have.						Y	Y					
Unavailable	The asset can't be contacted, or can't respond to contacts.		Y	Y	Y	Y	Y	Y	Y	Y		Y	Y
Underper-forming	Interactions with the asset take too long.		Y	Y		Y	Y		Y	Y		Y	Y
Unreliable	Interactions with the asset fail due to internal problems.		Y			Y	Y	Y	Y	Y		Y	

Table 1. Refined behaviour model

4.4 Threat classes

4.4.1 Systematic threat classification based on RFC 2828

Early in 2012, the threat model developed in 2011 was reviewed, and two types of problems identified:

- multiple threats in the model sometimes described the same phenomenon, i.e. there were overlapping and indistinguishable threat classes in the model;
- some phenomena that were relevant and important did not correspond to any threat in the model, i.e. there were gaps in coverage that arose by accident rather than by design.

It is not reasonable to insist that a dependability model cover all possible threats to assets, as in any type of system, some types of threats will be more important than others. However, one should decide consciously which types of threats to include and which to leave out. The presence of overlaps and unintentional gaps showed that a more methodical (and ideally standards based) approach should be used to derive the final SERSCIS dependability model.

In the first half of 2012, an attempt was made to do this using ISO 27001 and ISO 27005 as the starting points. As discussed in Section 2, these are well established standards for risk identification and management, and are similar to the SESAR approach with which we wanted to align due to its relevance to SERSCIS validation case studies. However, this approach failed, because while ISO 27001 provides checklists indicating the types of threats that should be considered, these are expressed in terms of security control objectives at system level. No more systematic derivation of threat classes could be found than the original ad-hoc brainstorming approach, and by late June 2012, little progress had been made.

The initial round of validation tests in July 2012 therefore used a simple extension of the SERSCIS proof of concept dependability model. The new Host (then called 'Server') and Network asset classes were added, along with a few threats against them. The old proof-of-concept threat models were then revised where they should involve Network assets. In a couple of cases, this required the introduction of secondary effect rules indicating that the old threat could be a knock-on consequence of a compromised Network (i.e. one showing specific behaviours). This was enough to test the value (or otherwise) of core model enhancements described in Section 3, but not to convince anyone that the SERSCIS model either was or could be made comprehensive enough for their needs.

The problems with ISO 2700x led to a reappraisal of alternative standards, which triggered a return to the old RFC 2828 standard. This standard is intended as an information security vocabulary, and the portion directly related to threats is quite brief. In 2011 this seemed to be a drawback, but after the experience with ISO 2700x the brevity of the relevant section of RFC 2828 began to look more like a virtue. But the main advantage of RFC 2828 was its decomposition of threats into threat actions (the event or situation that compromises the system) and threat consequences (the nature of the resulting compromise). The examples in RFC 2828 don't always rigorously maintain the distinction between action and consequences, but they show that factorising threats along these lines would simplify the problem of controlling overlaps and gaps in the threat model.

To derive a comprehensive set of threats, the first step was to enumerate possible consequences to any or all of the asset classes described in Section 4.2. This led to the list of outcomes shown in Table 2, which is actually slightly more comprehensive than RFC 2828 at least for the type of system covered by the SERSCIS dependability model.

No	Consequence	Meaning
1	Data disclosure	The threat causes data to be passed to or read by the wrong party. (One or more assets will become indiscreet).
2	Data corruption	The threat causes corruption of data. (One or more assets will be become inaccurate).
3	Underperformance	The threat causes the system to run too slowly. (One or more assets will begin to underperform).

No	Consequence	Meaning
4	Process error	The threat causes the system to fail in its function. (One or more assets will become unreliable).
5	Theft of service	The threat causes the system to function correctly but for the wrong party. (One or more assets will become promiscuous).
6	Control of function	The threat causes control of the system to be lost to an inappropriate party. (One or more system assets will be rendered untrustworthy).
7	Overload	The threat places more load on the system than it can handle. (One or more assets will become overloaded).
8	Denial of service	The threat makes it impossible for one or more users to interact with some or all of the system functions. (One or more assets will become unavailable).
9	Loss of reputation	The threat undermines other parties trust that the system will work correctly on their behalf. (One or more assets will become dissatisfied with their interaction).

Table 2. Threat consequences

Note that these consequences don't correspond exactly to the behaviours identified in Table 1.There are two reasons for this. Firstly, some consequences affect future behaviour rather than current behaviour, so the consequence is to change an asset's security attributes rather than its behaviour. For example, consequence 6 means an inappropriate party is now in control, making assets untrustworthy without necessarily affecting their current behaviour. And secondly, some behaviour is associated with threat actions rather than consequences, e.g. an unauthentic asset may indicate an impersonation attack, but this is a means to several of the possible ends listed in Table 2.

A similar approach was used to generate a set of generic threat actions (i.e. means to the above ends), which are listed in Table 3.

No	Consequence	Meaning
1	Destruction	An asset is destroyed.
2	Unrestricted access	An asset is accessed without proper restrictions.
3	Theft	An asset is removed by an unauthorized party.
4	Corruption	An asset is altered, but not removed or destroyed.
5	Insider attack	An asset is misused or caused to fail by an authorised operator.
6	Traffic interruption	Messages to and from an asset are blocked.
7	Traffic interference	Spurious messages are sent to or from an asset.
8	Traffic corruption	Messages to and from an asset are altered.
9	Traffic snooping	Messages to and from an asset are read by someone who is not the source or the intended destination.
10	Software bug	There is a bug in the software components of an asset. (This is handled separately from other malfunctions, because of its importance to information security).
11	One-off malfunction	A fault occurs in an asset.
12	Persistent malfunction	A fault keeps occurring in an asset. (I.e. the asset becomes predictably faulty).
13	Impersonation	An imposter pretends to be an asset.
14	Deception	An imposter deceives an asset by pretending to be another.
15	Under-provisioning	An asset's capacity is lower than it should be.
16	Over-use	An asset is subjected to more use than expected (or than it committed to).
17	Over-commitment	An asset commits to do more than it can.
18	Remote known exploit	An asset is hacked over a network using a previously known security flaw.
19	Remote zero day exploit	An asset is hacked over a network using a previously unknown security flaw.
20	Other actions	Includes secondary effects that may arise through multiple primary causes.

Table 3. Threat actions

It is reasonably easy to convince oneself that the lists of threat actions and consequences from Table 3 and Table 2 contain no obvious overlaps or gaps. One cannot guarantee nothing has been overlooked, but by considering the two aspects separately, the number of possibilities one must consider is reduced so one can avoid overlaps and have greater confidence that all major security concerns are covered.

It is also apparent that the two tables maintain a clear distinction between threat actions and consequences: most of the actions from Table 3 could be used to achieve several possible ends from Table 2. Thus taking all meaningful combinations of the two aspects should cover a large and varied set of potential threats in a systematic way avoiding overlaps and obvious gaps in the coverage.

We have to do this for each type of asset described in Section 4.2, so the set of potential threats is actually very large indeed. In practice, not all combinations are represented in the final SERSCIS dependability model, for one of four reasons:

- some combinations of action and consequence are not meaningful, e.g. the traffic snooping action cannot be meaningfully combined with consequences other than data disclosure;
- limitations in emulation fidelity in the SERSCIS test bed mean some actions cannot be distinguished, e.g. software bugs are not simulated, so actions 10, 18 and 19 look the same;
- the run-time needed for semantic reasoning over the concrete system model depends on the number of threats, so one should not add threats that have little relevance to the system;
- encoding threat classes in OWL requires expertise with Protégé as well as security, so the work involved multiple people making it labour intensive and error prone.

The first of these reasons is valid in all situations, and even the most comprehensive dependability model should not include all action-consequence combinations. The other reasons are really about the trade-off between coverage and time/cost. Ideally, the dependability model would include all the meaningful threats. These would then be filtered when constructing the abstract system model, to remove threats that can’t be distinguished by the available system monitoring, or that aren’t worth the extra computation time given their relative importance in the desired application. The problem is that a system designer who is not expert in both semantics and security would need sophisticated tools to filter threats correctly. Lacking the resources to encode all meaningful threat models and develop such tools, it was decided to filter the threats before encoding the dependability model. The final dependability model is therefore not completely comprehensive, but thanks to RFC 2828 and the above Tables, a systematic procedure can be used to fill any gaps if they are found to be important.

The following subsections and tables provide information about the new threat classes for each type of asset. The threat class names are based on the systematic construction approach: all names begin with a series of letters indicating the threatened asset type, followed by two numbers indicating the threat action from Table 3 and consequence from Table 2. This prefix provides an easy way to understand the nature of the threat and its consequences in general terms. A descriptive name is then added which is sometimes repeated where the same attack method can be used for different purposes.

4.4.2 Threats to Space assets

The table below lists the threats to Space assets.

Threat class	**Description**	**Involved assets**	**Primary/Secondary**
S3.5-PhysicalIntrusion	Access to a Space by an unauthorised party, enabling them to make use of the Space for unauthorized purposes.	Space	Primary only.

Table 4. Threats to Space assets

Other threats to spaces are possible, but were left out because they are not very relevant for the selected validation scenarios. For example, SG1.8 (denial of use of a space by physical destruction) is unlikely to be used to disrupt A-CDM operations, as opposed to (say) terrorist attack on the airport.

4.4.3 Threats to Network assets

The table below lists the threats to Network assets.

Threat class	Description	Involved assets	Primary/Secondary
N6.8-RadioSignal Interference	Jamming attack launched from a space in which a RadioNetwork is accessible, thus making it unavailable. Does not affect other types of Network.	RadioNetwork Space	Primary only.
N6.8-WiFiSignal Interference	Jamming attack launched from a space in which a WiFiNetwork is accessible, thus making it unavailable. Does not affect other types of Network.	WiFiNetwork Space	Primary only.

Table 5. Threats to Network assets

These threats were included in the model because they are relevant for some of the network-based attack scenarios used in validation testing. Other threats such as N13.1 (impersonating a network) were left out because the network emulators used in the test bed don't simulate network authentication procedures so this type of threat couldn't be emulated including its detection and countermeasures.

4.4.4 Threats to Host assets

The table below lists the threats to Host assets, including attacks on Interface assets associated with the Host.

Threat class	Description	Involved assets	Primary/Secondary
I7.8-PacketFlood	Remote attack, sending large numbers of messages over a network to the hosts interface with the network making the host unavailable..	Host Network Interface	Primary only.
H10.2-ServerSoftware Malfunction	Software malfunction making hosted processes inaccurate.	Host	Primary only.
H18.8-RemoteKnown Exploit	Remote attack, exploiting a known bug to disrupt the host making it unavailable.	Host Network Interface	Primary only.

Table 6. Threats to Host assets

Many other threats are possible, e.g. other H18.x threats with different consequences such as data access or corruption, or usurpation of control, along with equivalent H19.x threats (using unknown or zero-day exploits). Many were left out because they were not considered relevant for the validation test scenarios. Others were omitted because of limitations in the test bed emulators, e.g. H18.2 or H19.2 are relevant, but would be indistinguishable from H10.2 threat because the emulators don't explicitly simulate different types of software bugs.

4.4.5 Threats to ServiceGroup assets

The table below lists the threats to ServiceGroup assets. To save space here and in subsequent tables, ClientSpecifiedResource and ProviderSpecifiedResource assets are abbreviated to CSR and PSR.

Threat class	Description	Involved assets	Primary/Secondary
SG2.1-Unaccountable ServiceDataAccess	Access to service data with no customer context (i.e. nobody is accountable).	ServiceGroup	Primary only.
SG2.2-Unaccountable ServiceDataUpdate	Update of service data with no customer context (i.e. nobody is accountable).	ServiceGroup	Primary only.
SG2.5-Unaccountable ServiceAccess	Theft of service, charged to nobody (i.e. there is no accountable customer).	ServiceGroup	Primary only.

Threat class	Description	Involved assets	Primary/Secondary
SG6.8-InternalTraffic Interruption	Interruption of communications needed by the service, caused by a loss of availability of an involved network or interface.	ServiceGroup SGHost SGInterface Network NetworkGroup	Secondary only.
SG10.2-Service SoftwareMalfunction	Software malfunction making the service inaccurate. May be a secondary effect if there is a malfunction in the host.	ServiceGroup SGHost	Can be either.
SG11.2-OneOff ResourceInaccuracy	One-off inaccuracy in a resource makes the service intermittently inaccurate. Note that the resource itself is involved but not threatened as a one-off problem does not undermine its future value to the service Provider.	ServiceGroup PSR ResourceGroup	Primary only.
SG11.2-OneOff Resource Underperformance	One-off underperformance in a resource makes the service underperform intermit-tently.	ServiceGroup PSR ResourceGroup	Primary only
SG11.2-OneOff ResourceMalfunction	One-off malfunction in a resource makes the service intermittently malfunction.	ServiceGroup PSR ResourceGroup	Primary only.
SG15.3-HostUnder Performance	An underperforming host causes a service to underperform.	ServiceGroup SGHost	Secondary only.
SG15.3-Service Overload	A service underperforms as a knock-on consequence of being overloaded.	ServiceGroup	Secondary only.
SG15.3-Resource Underperformance	Underperforming resources cause a service to underperform.	ServiceGroup ResourceGroup	Secondary only.
SG15.3-Resource Overload	Overloaded resources cause a service to underperform through having to wait for resources of a given type.	ServiceGroup ResourceGroup	Secondary only.
SG15.8-Host Unavailable	An unavailable host causes a service to become unavailable.	ServiceGroup SGHost	Secondary only.
SG15.8-Resource Unavailable	Unavailable resources cause a service to become unavailable.	ServiceGroup ResourceGroup	Secondary only.
SG17.7-TooMany Customers	A Provider takes on too many customer commitments, leading to the service being overloaded.	ServiceGroup	Primary only.
SG20.8-Internal ServiceTraffic Interruption	A lack of available networks for internal communication causes a service to become unavailable.	ServiceGroup SGHost NetworkGroup	Secondary only.

Table 7. Threats to ServiceGroup assets

Several types of threats implied by RFC 2828 have been omitted, e.g. SG18.x or SG19.x threats involving remote exploits to achieve various ends (data access, data corruption or service misdirection, performance degradation, denial of service, etc). These were left out either because they are not as relevant to the application scenarios, or because they can't be distinguished from other threats due to limitations of the airport emulation used for validation testing.

4.4.6 Threats to Consumer assets

The table below lists the threats to Consumer assets.

Threat class	Description	Involved assets	Primary/Secondary
CO6.8- Consumer TrafficInterruption	Interruption of communications between a consumer and service, caused by a loss of availability in an involved network or interface.	Consumer COHost COInterface ServiceGroup SGHost SGInterface Network NetworkGroup	Secondary only.
CO8.2-Consumer TrafficCorruption	Corruption of communications between a consumer and service. May be a secondary effect of a malfunctioning network, or a primary effect of malicious tampering of messages in transit.	Consumer COHost ServiceGroup SGHost Network	Can be either.
CO9.1-Consumer TrafficSnooping	Unauthorised reading of communications between a consumer and service.	Consumer COHost ServiceGroup SGHost Network	Primary only.
CO10.2-Client SoftwareMalfunction	Software malfunction making the consumer inaccurate. May be a secondary effect if there is a malfunction in the host.	Consumer COHost	Can be either.
CO13.1-Consumer Impersonation	Unauthorised access to data though access to a service using false/forged credentials.	Consumer COHost ServiceGroup	Primary only.
CO14.1-Service Impersonation	Unauthorised access to data by convincing a consumer that an imposter is the service. May be a secondary effect caused by the service being unauthentic (e.g. lacking any identification), but a consumer may also be fooled even if this is not the case.	Consumer SGHost ServiceGroup	Can be either.

Table 8. Threats to Consumer assets

Several threats based on ConsumerImpersonation or ServiceImpersonation could be added with different threat consequences (e.g. data corruption, theft of service, denial of service, etc). These are relevant to the validation sector, but were not used in simulated validation scenarios and so were left out to save time.

4.4.7 Threats to Customer assets

The table below lists the threats to Customer assets.

Threat class	Description	Involved assets	Primary/Secondary
CU2.1-Unauthorized CustomerDataAccess	Access to the customer's data via unauth-orized access to a service.	Customer ServiceGroup	Primary only.
CU2.2-Unauthorized CustomerDataAccess	Corruption of the customer's data via unauthorized access to a service.	Customer ServiceGroup	Primary only.
CU2.5-MissAccounted ServiceAccess	Theft of service paid for by the customer via unauthorized access to the service.	Customer ServiceGroup	Primary only.

Threat class	Description	Involved assets	Primary/Secondary
CU6.6- Customer TrafficInterruption	Interruption of communications between a customer and service, caused by a loss of availability in an involved network or interface. The customer manages rather than uses the service, so the effect is a loss of control by the customer, rather than loss of access to the service (as in CO6.8).	Customer CUHost CUInterface ServiceGroup SGHost SGInterface Network NetworkGroup	Secondary only.
CU9.1-Customer TrafficSnooping	Unauthorised reading of communications between a customer and service.	Customer CUHost ServiceGroup SGHost Network	Primary only.
CU10.6-Customer SoftwareMalfunction	Software malfunction making the customer inputs inaccurate. May be a secondary effect if there is a malfunction in the host. The customer manages rather than uses the service, so the effect is a loss of control by the customer, rather than a loss of accuracy in the service (as in CO10.2).	Customer CUHost	Can be either.
CU13.1- Customer Impersonation	Unauthorised access to data though access to a service using false/forged credentials.	Customer CUHost ServiceGroup	Primary only.
CU14.1-Service Impersonation	Unauthorised access to data by convincing a customer that an imposter is the service. May be a secondary effect caused by the service being unauthentic (e.g. lacking any identification), but a customer may also be fooled even if this is not the case.	Customer SGHost ServiceGroup	Can be either.
CU16.3-Oversized CustomerRequests	Users of the service authorized by the customer send oversized requests which take too long to process, making the service underperform relative to its commitments to this customer, and in general.	Customer ServiceGroup	Primary only.
CU16.7-Excessive CustomerRequests	Users of the service authorized by the customer send too many requests, thus overloading the service relative to the customer's SLA, and in general.	Customer ServiceGroup	Primary only.
CU20.1-Indiscreet Consumer	The secondary effect on a customer of having an indiscreet consumer, i.e. data leakage and customer dissatisfaction.	Customer Consumer ServiceGroup	Secondary only.
CU20.2-Inaccurate Consumer	The secondary effect on a customer of having an inaccurate consumer, i.e. data corruption or inaccuracy.	Customer Consumer ServiceGroup	Secondary only.
CU20.1-Indiscreet CSR	The secondary effect on a customer of having an indiscreet client-specified resource: data leakage and customer dissatisfaction.	Customer CSR ServiceGroup	Secondary only.
CU20.2-Inaccurate CSR	The secondary effect on a customer of having an inaccurate client-specified resource: data corruption or inaccuracy.	Customer CSR ServiceGroup	Secondary only.
CU20.8-Unavailable CSR	The secondary effect on a customer of having an unavailable client-specified resource: the service would also become unavailable to the customer's authorised consumers.	Customer CSR ServiceGroup	Secondary only.

Table 9. Threats to Customer assets

Some CU20.9 secondary effects were omitted which are relevant to the validation scenarios, e.g. loss of trust due to service indiscretion, inaccuracy or underperformance. This happened because threat consequence 9 (loss of reputation) was initially overlooked in Table 2. The oversight was detected during the final validation tests, but it was too late to update the threat model before the end of the project.

4.4.8 Threats to ClientSpecifiedResource assets

The table below lists the threats to ClientSpecifiedResource assets.

Threat class	Description	Involved assets	Primary/Secondary
CSR2.1-Unauthorized ClientSpecified ResourceDataAccess	Access to the customer's data via unauthorized access to a CSR.	CSR	Primary only.
CSR2.1-Unauthorized ClientSpecified ResourceDataUpdate	Corruption of the customer's data via unauthorized access to a CSR.	CSR	Primary only.
CSR6.8-Client SpecifiedResource TrafficInterruption	Interruption of communications between a service and a CSR, caused by a loss of availability in an involved network or interface.	ServiceGroup SGHost SGInterface CSR CSRHost CSRInterface Network NetworkGroup	Secondary only.
CSR8.2-Client SpecifiedResource TrafficCorruption	Corruption of communications between a service and a CSR. May be a secondary effect of a malfunctioning network, or a primary effect of malicious tampering of messages in transit.	CSR CSRHost ServiceGroup SGHost Network	Can be either.
CSR9.1-Client SpecifiedResource TrafficSnooping	Unauthorised reading of communications between a CSR and service.	CSR CSRHost ServiceGroup SGHost Network	Primary only.
CSR10.2-Client SpecifiedResource SoftwareMalfunction	Software malfunction making the CSR inaccurate. May be a secondary effect if there is a malfunction in the host.	CSR CSRHost	Can be either.
CSR13.1-Client SpecifiedResource Impersonation	Unauthorised access to data by pretending to be a CSR and fooling a service to send the data.	CSR CSRHost ServiceGroup	Primary only.
CSR14.1-Authorized CSRUserImperson-ation	Unauthorised access to data at a CSR by pretending to be an authorized user (but not the service). May be a secondary effect of having an unauthentic consumer, or a primary attack on a faulty delegated access control system at the CSR.	CSR Customer Consumer	Can be either.
CSR14.1-Service Impersonation	Unauthorised access to data at a CSR by pretending to be a service that has been instructed and authorized to use it. May be a secondary effect if the service is already unauthentic (e.g. lacks identification), or a primary attack on a faulty delegated access control system at the CSR.	CSR ServiceGroup SGHost	Can be either.
CSR14.2-Authorized CSRUserImperson-ation	Corruption of data at a CSR by pretending to be an authorized user (but not the service). May be a secondary or a primary effect (see the corresponding CSR14.1 threat above).	CSR Customer Consumer	Can be either.

Threat class	Description	Involved assets	Primary/Secondary
CSR14.2-Service Impersonation	Corruption of data at a CSR by pretending to be a service that has been instructed and authorized to use it. May be a secondary or a primary effect (see the corresponding CSR14.1 threat above).	CSR ServiceGroup SGHost	Can be either.
CSR18.8-Remote KnownExploit	Renders a CSR unavailable by exploiting a known vulnerability in its software.	CSR CSRHost CSRInterface Network	Primary only.
CSR20.8-Client SpecifiedResource TrafficInterruption	A lack of available networks connecting a CSR and a service causes the CSR to be unavailable to the service.	CSR CSRHost NetworkGroup ServiceGroup SGHost	Secondary only.
CSR20.8-Host Unavailable	An unavailable host causes a CSR to become unavailable.	CSR CSRHost	Secondary only.

Table 10. Threats to ClientSpecifiedResource assets

Some threats implied by RFC 2828 were omitted because they have low relevance in the validation scenarios, but some were left out just to save time towards the end of the project. For example, CSR9.1 network traffic snooping, or more CSR18.x remote exploit threats to achieve other consequences (data access or corruption).

4.4.9 Threats to ProviderSpecifiedResource assets

The table below lists the threats to ProviderSpecifiedResource assets.

Threat class	Description	Involved assets	Primary/Secondary
PSR2.1-Unauthorized ResourceDataAccess	Access to service data via unauthorized access to a PSR.	PSR	Primary only.
PSR2.2-Unauthorized ResourceDataUpdate	Corruption of service data via unauthoriz-ed access to a PSR.	PSR	Primary only.
PSR2.5-MissAccount-edResourceAccess	Theft of resource usage from a provider, by pretending to be the service.	PSR ServiceGroup SGHost	Primary only.
PSR6.8-Provider SpecifiedResource TrafficInterruption	Interruption of communications between a service and a PSR, caused by a loss of availability in an involved network or interface.	ServiceGroup SGHost SGInterface PSR PSRHost PSRInterface Network NetworkGroup	Secondary only.
PSR8.2-Provider SpecifiedResource TrafficCorruption	Corruption of communications between a service and a PSR. May be a secondary effect of a malfunctioning network, or a primary effect of malicious tampering of messages in transit.	PSR PSRHost ServiceGroup SGHost Network	Can be either.
PSR9.1-Provider SpecifiedResource TrafficSnooping	Access to service data by snooping traffic between a service and a PSR.	PSR PSRHost ServiceGroup SGHost Network	Can be either.
PSR10.2-Provider SpecifiedResource SoftwareMalfunction	Software malfunction making the PSR inaccurate. May be a secondary effect if there is a malfunction in the host.	PSR PSRHost	Can be either.
PSR13.1-Provider SpecifiedResource Impersonation	Unauthorised access to data by pretending to be a PSR and fooling a service to send the data.	PSR PSRHost ServiceGroup	Primary only.

Threat class	Description	Involved assets	Primary/Secondary
PSR13.2-Provider SpecifiedResource Impersonation	Corruption of data by pretending to be a PSR and fooling a service to send updates, so they are not applied correctly to the real data held by the real PSR.	PSR PSRHost ServiceGroup	Primary only.
PSR14.1-Service Impersonation	Unauthorised access to data at a PSR by pretending to be the service. May be a secondary effect if the service is already unauthentic (e.g. lacks identification), or a primary attack on a faulty delegated access control system at the PSR.	PSR ServiceGroup SGHost	Can be either.
PSR14.2-Service Impersonation	Corruption of data at a PSR by pretending to be the service. May be a secondary effect if the service is already unauthentic (e.g. lacks identification), or a primary attack on a faulty delegated access control system at the PSR.	PSR ServiceGroup SGHost	Can be either.
PSR18.8-Remote KnownExploit	Renders a CSR unavailable by exploiting a known vulnerability in its software.	PSR PSRHost PSRInterface Network	Primary only.
PSR20.3-Overloaded ProviderSpecified Resource	The PSR underperforms as a knock-on consequence of being overloaded (but not necessarily by the provider).	PSR	Secondary only.
PSR20.3-Overloaded RG	The PSR becomes overloaded because the ResourceGroup it is in is overloaded (e.g. it has too few resources of that same type).	PSR ResourceGroup	Secondary only.
PSR20.8-Provider SpecifiedResource TrafficInterruption	A lack of available networks connecting a PSR and a service causes the PSR to be unavailable to the service.	PSR PSRHost NetworkGroup ServiceGroup SGHost	Secondary only.
PSR20.8-Host Unavailable	An unavailable host causes a PSR to become unavailable.	PSR PSRHost	Secondary only.

Table 11. Threats to ProviderSpecifiedResource assets

Some threats implied by RFC 2828 were omitted because they have low relevance in the validation scenarios, but some were left out just to save time towards the end of the project, as with CSR threats.

4.4.10 Threats to ResourceGroup assets

The table below lists the threats to ResourceGroup assets.

Threat class	Description	Involved assets	Primary/Secondary
RG12.2-Persistent ResourceInaccuracy	A resource group's value is undermined by containing a PSR that is often inaccurate.	PSR ResourceGroup	Secondary only.
RG12.3-Persistent ResourceUnderperf-ormance	A resource group's value is undermined by a PSR that often underperforms.	PSR ResourceGroup	Secondary only.
RG12.3-Persistent ResourceUnavaila-bility	A resource group's value is undermined by a PSR that is often unavailable.	PSR ResourceGroup	Secondary only.
RG15.7-TooFew Resources	A resource group has too few resources, leading it to become overloaded.	ResourceGroup ServiceGroup	Primary only.
RG15.8-TooFew Resources	A resource group has too few resources, leading it to become unavailable (i.e. have no resources available when the service needs one).	ResourceGroup ServiceGroup	Primary only.

Threat class	Description	Involved assets	Primary/Secondary
RG16.7-Overloaded Service	A resource group becomes overloaded due to a service using it being overloaded, i.e. not because it has too few resources.	ResourceGroup ServiceGroup	Secondary only.

Table 12. Threats to ResourceGroup assets

4.4.11 Secondary effect chains and root cause analysis

One feature of the above threat model is the large number of threats that are included only to capture secondary effects. This makes the model far more powerful as a root cause analysis tool, which is very useful when analysing system behaviour at run time.

For example, suppose an attacker carries out a packet flood attack against a host across a network, i.e. an I7.8-PacketFlood, targeting the supplier of a provider specified resource. This would cause the supplier's interface to the network to become unavailable, and interrupt communication traffic between their PSR and the provider's service over that network, i.e. a PSR6.8-ProviderSpecified-ResourceTrafficInterruption threat. If there are no other networks, or if the attacker can carry out the attack across all the available networks, then we get a PSR20.8-ProviderSpecifiedResourceTraffic-Interruption and the PSR becomes unavailable. This in turn compromises the ResourceGroup for this type of PSR via RG12.3-PersistentResourceUnavailability, and possibly thence to a loss of availability in services that use this type of resource via SG15.8-ResourceUnavailable.

In this situation, the system monitoring would detect a lot of unavailable assets: a service, a resource group, a PSR within that group, some networks, etc. Unfortunately, most of these effects are not unique to a packet flood attack – other possible root causes that could explain many of the symptoms include a remote exploit disabling the PSR host, jamming of the network, a lack of capacity at the PSR, a shortage of PSRs of the required type, etc. Since most of the knock-on effects are common to all these causes, it won't be easy to distinguish between them. Moreover, if the probability of each threat causing detection of adverse behaviours be a little off, the Bayesian inference system may conclude that some are coincidental and misread secondary effects as due to other, unrelated primary causes. Essentially, the Bayesian inference method has the same problems that a human might, when faced with a system that is misbehaving in multiple ways.

The new model avoids this by using SWRL rules to determine whether each threat could be caused as a secondary effect of the observed set of behaviours, as discussed in Section 3.2.2. Any threats that are determined to be secondary are automatically considered active by the Bayesian estimator. In the above scenario the PSR6.8, PSR20.8, RG12.3 and SG15.8 threats would be considered secondary, so accounting for most of the observed misbehaviour. The primary threat identification is then based on the remaining observations – in this case unavailability of the interface used by one PSR, but not other interfaces or other assets using the same networks. It is therefore far easier to distinguish between the possible root causes, allowing the packet flood attack to be identified and not alternatives such as network jamming or software malfunction.

4.5 Controls

The last element of the SERSCIS core model alongside assets, adverse behaviours and threats are the controls. The dependability model provides an updated set of control subclasses describing various mechanisms that can be used to counteract dependability model threats. These are similar to the control classes used in 2011, but with some merging where there is no need to distinguish similar concepts applied to different assets, and some new mechanisms related to network security.

The control classes from the final dependability model are listed in Table 13.

Control	Meaning	Applies to											
		Space	**Host**	**Network**	**Interface**	**ServiceGroup**	**Customer**	**Consumer**	**CSR**	**PSR**	**ThirdParty**	**ResourceGroup**	**NetworkGroup**
Access Control	The asset refuses attempts to make use of it by unauthorized parties.	Y	Y	Y		Y			Y	Y			
Access Logging	The asset makes records of all usage.	Y	Y	Y	Y	Y			Y	Y			
Blacklisted	The asset is not used, thus blocking any threats involving it.			Y	Y			Y		Y			
Blacklisting	It is OK for the asset to blacklist other assets related to it.					Y	Y					Y	Y
Client Authentication	The asset authenticates any other asset seeking to use it.	Y	Y	Y		Y			Y	Y			
Delegation	The asset has a means to delegate rights it possesses to another asset.					Y	Y				Y		
Encryption	The asset can encrypt and decrypt data passed to it, and refuses any unencrypted communication.			Y		Y	Y	Y	Y	Y	Y		
Failover	The asset can dynamically switch between different candidate logical assets should any fail.					Y	Y						
Firewall	The asset blocks unwanted communication traffic.				Y								
Frequency Hopping	The asset evades jamming by constantly switching frequency[1].			Y									
Identification	The asset has a means to identify itself, which can be checked by ClientAuthN or ServiceAuthN.		Y										
Message Verification	The asset can verify messages and refuses to accept messages where this is not possible.					Y	Y	Y	Y	Y	Y		
Network Switching	The asset can dynamically switch between Networks should any become unavailable.					Y	Y	Y					
Redundancy	The asset has multiple related assets it can use to deliver its functions.						Y					Y	Y
Resource Capacity Management	The asset has a mechanism for acquiring more options if needed.											Y	Y
Service Authentication	The asset authenticates any other asset before using it.					Y	Y	Y			Y		
Signal Boosting	The asset evades jamming by increasing the power used[2].			Y									
SLA	The asset makes well-defined agreements as on what it can do, and refuses access without an SLA.					Y	Y			Y			
SLABreach	The asset uses SLAs, and is able to selectively breach some to protect the service delivered under the rest.					Y							

[1] WiFi and Radio networks only.
[2] Radio networks only.

Control	Meaning	Applies to											
		Space	Host	Network	Interface	ServiceGroup	Customer	Consumer	CSR	PSR	ThirdParty	ResourceGroup	NetworkGroup
SLA Commitment Management	The asset uses SLAs, and has a way to ensure it never takes on an SLA without the capacity to deliver.					Y							
SLA Enforcement	The asset uses SLAs, and has a way to enforce any limits on usage.					Y							
Software Patching	Software updates are applied as soon as they are available, thus protecting against known bugs.		Y			Y	Y	Y	Y	Y	Y		
Staff Vetting	Staff are checked to ensure they won't act maliciously against the wishes of their employer.		Y										

Table 13. Refined model of controls

4.6 SWRL rule editor

The dependability model in SERSCIS contains the SWRL rules for the classification of concrete threat instances into BlockedThreats, MititagedThreats, Vulnerabilities and SecondaryEffects. There is limited tooling support for the creation and management of these SWRL rules in Protégé. There is no way to search for a particular rule once it has been created and this makes the task of updating and editing the rules very time consuming (100 plus rules in the dependability model). To solve this problem, we created a SWRL rule editor which simplifies the process of managing SWRL rules in OWL ontologies. The main functionalities provided by the SWRL rule editor are as follows:

1. Users are able to create rules using an intuitive user interface (with auto suggest function).
2. The editor automatically groups the rules at start-up (e.g. BlockedThreats, MitigatedThreats etc).
3. Users are able to search over the rules and the system automatically filters and displays them on the fly. This makes the updating and management of rules straightforward.
4. Syntax checking for SWRL is inbuilt and enforced upon the user during creation. This eliminated syntax errors and misspelling during rule creation.

5 Abstract system modelling tools

The abstract system model is produced by further sub-classing the assets and threats from the dependability model, representing system-specific assets and their relationships, and variations on the threats involving different combinations of system-specific assets. The main challenge is that abstract system model development should be carried out by an expert in the system, who may not be an expert in security and almost certainly won't be expert in semantic modelling and reasoning technology. A typical system modeller therefore cannot be expected to develop models using Protégé. Two tools were created in SERSCIS to provide this help: a graphical system model editor for system-specific asset class definition, and an automatic threat class generator.

5.1.1 SERSCIS system model editor

The system model editor was a completely new development in 2012, responding to feedback from Austrocontrol that users will need graphical tools that present their system in a familiar way. The user can create asset subclasses and define their relationships by simply manipulating diagrammatic views representing logical or physical assets in the system.

The SERSCIS system model editor allows no semantic experts (who can also use tools like Protégé) to create abstract system models using intuitive point and click interfaces which generate diagrammatic representation of the system specific assets along with the relationship between these assets. The models created conform to the axiomatic and subclass descriptions provided in the dependability model (See Figure 13).

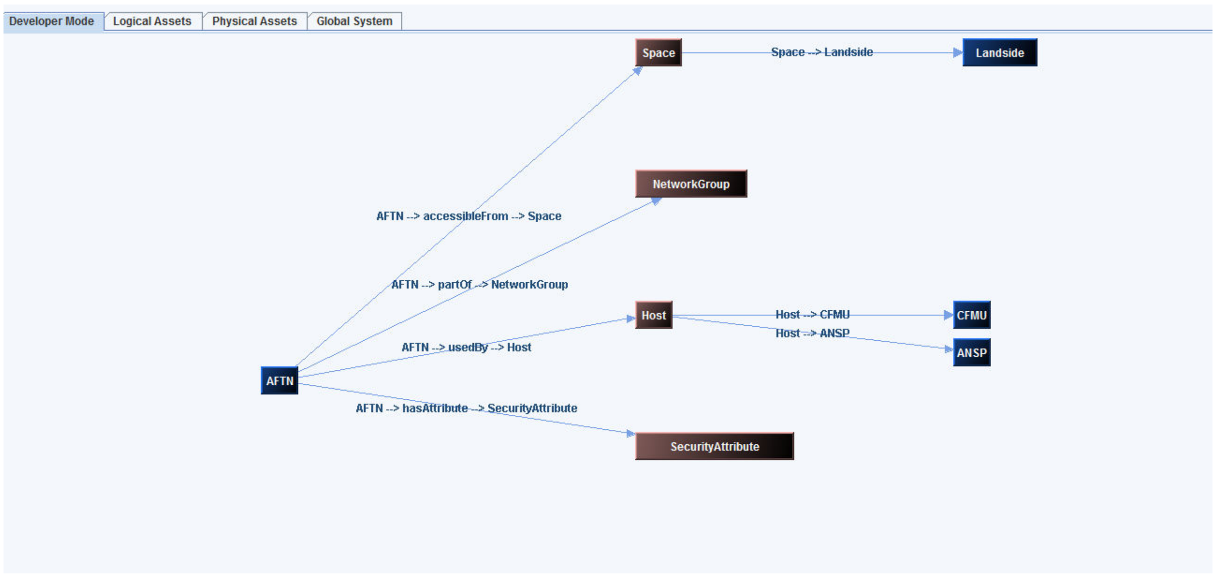


Figure 13: SERSCIS system model editor.

The editor supports four main modes of operation:

1. The developer mode: In this mode, the user can create a new abstract system model and add system specific subclasses along with the relationship information to other system specific assets. The user can also switch to any specific asset of the system to further refine it.
2. The logical view: In the logical view, the user is able to visualise the logical assets of the system and their relationship to other logical assets. The user can either focus on a particular logical asset or switch to the global logical view which displays the entire logical asset system overview. Editing the system model is not possible in this view. This view also lists the connection between the logical assets to their respective physical assets using list dropdowns. (e.g. hosts and hostedBy relations)
3. The physical view: In the physical view, the user is able to visualise the physical assets of the system and their relationship to other physical assets. The user can either focus on a particular physical asset or switch to the global physical view which displays the entire physical asset system overview. Editing the system model is not possible in this view. This view also lists the connection between the physical assets to their contained logical assets using list dropdowns. (e.g. hosts and hostedBy relations)
4. The global view: The global view visualises the entire abstract system model (both logical and physical assets) and the relationships between them. The global physical view is less useful for very complex models with a large number of assets as the view can be cluttered. A "save positions" button is provided within all the views using which the user can position the assets in a form which they find convenient and the coordinates are stored as annotations within the OWL ontology and the same view is restored upon reloading the model.

The OWL system model editor has also been used as a starting point for the SERSCIS DST for the visualisation of assets at run-time.

5.1.2 System threat generator

The system-specific threat class generator is based on an earlier (and imperfect) version created in 2011. The new version has been updated for consistency with the changes in the SERSCIS core model as described in Section 3.2, and the algorithms have been improved so they no longer produce slightly incorrect results for some types of threats.

The sequence in which the system threat generator works is as follows:

1. Load the ontologies along with their dependencies (e.g. system.owl dependent on dependability.owl dependent on core.owl etc)
2. Retrieve all the threat subclasses. (i.e. leaves of the Threat superclass)
3. For even given threat subclass in the dependability model:
    a. Get the involved assets (check for involves relationship)
    b. Get the threat class description. (check for hasDescription property)
    c. Get the threatened asset (check for threatens relationship)
4. For every involved class:
    a. Find the leaf asset classes from the system model
    b. Store the involved class and its system specific leaf classes in a map
5. Use the ListCombinationGenerator to find all the possible relationships between the involved system specific asset subclasses
6. For each combination produced by the ListCombinationGenerator:
    a. Find the unique combinations after applying a filter
    b. Determine a final involved class list after taking the highest combination tuple.
7. Create system specific threat classes for the final involved asset classes.

6 Concrete System Model Generation

The decision support components for creating the concrete system model have been extended and updated for the final testbed. Figure 14 shows the architecture of the decision support components with reference to the rest of the testbed.

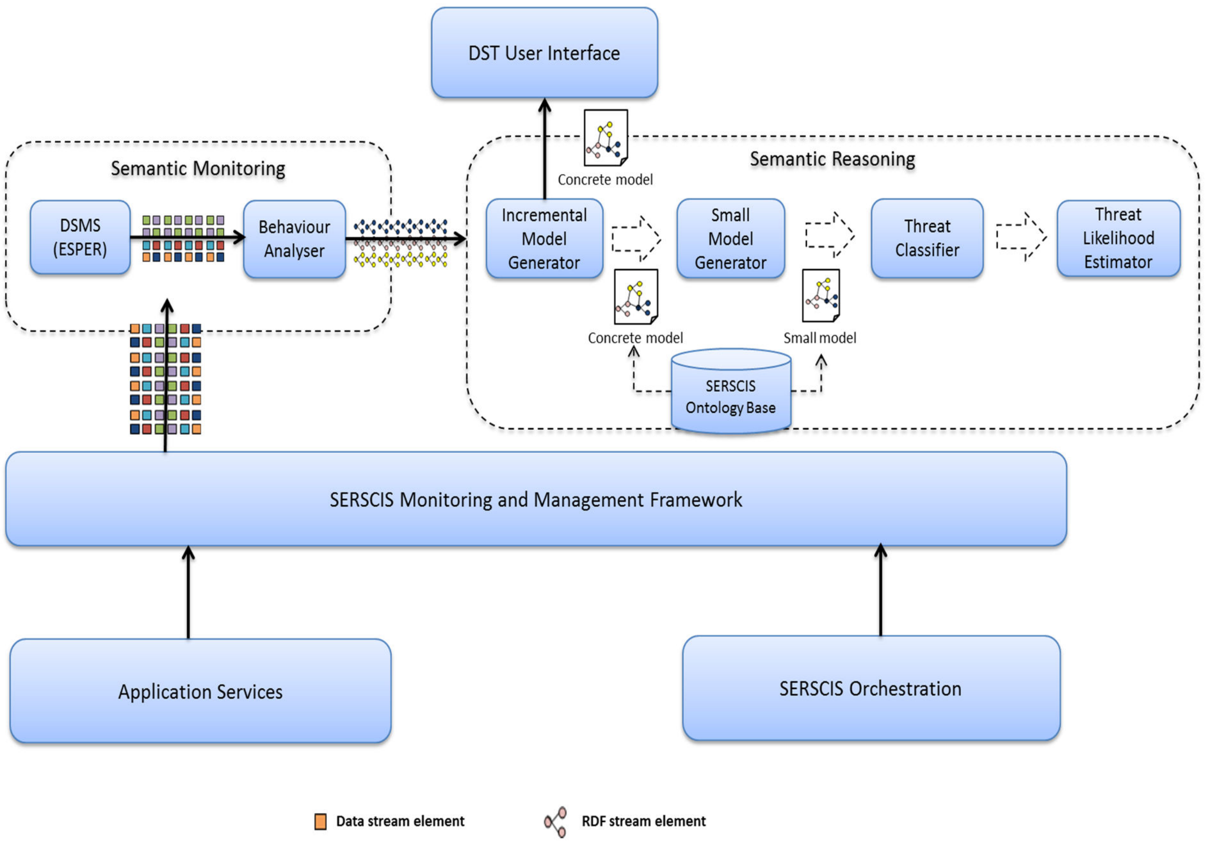


Figure 14: Stream reasoning components incorporated in the testbed architecture

The following is a brief summary of each component, indicating whether it is new or has been updated since the PoC testbed:

- DSMS (Data Stream Management Service): samples and filters the streams of monitoring data generated by the service monitoring and management components. This is a new component. We have used ESPER [22] for this purpose.
- Behaviour analyser: analyses specific system behaviours by processing multiple streams of data values (relating to assets' metrics) from the DSMS and indicates the start or end of the observation of a particular assets, asset behaviour or control. Then, it triggers the components defined in the semantic reasoning framework step by step to update, classify and reason the concrete model. This is a new component that replaces and extends the functionality that was implemented using WP3 management policies in the PoC testbed.
- Incremental model generator: creates and incrementally updates a concrete model of assets, behaviours and controls, based on the event information produced by the behaviour analyser combined with the WP2 system model. This component has been updated since the PoC to allow the concrete model to be incrementally updated, in contrast to the previous process of taking regular 'snapshots' of the running system.
- Small model generator: is a newly introduced component that extracts a sub-graph form the concrete model and generates a small-sized model consisting of a particular updated asset (i.e. the newly created asset, the removed asset, the asset with control updates, the asset with induced behaviour updates, etc.). Generated small models contain recently system updates on

per affected asset basis, therefore small sized but with sufficient semantic relations for the following semantic classifications.

- Threat classifier: is fed with a small model and classifies threats as mitigated or blocked, based on the presence of adequate controls. The classification results are then populated back to the singleton concrete model. This component has been updated since the PoC to take account of the new system model.
- Threat likelihood estimator: takes full concrete model and updates the current threat likelihood for all threats. The output is the updated concrete model. This component has been updated since the PoC to improve its efficiency in dealing with large numbers of threats and to take into account rules encoded in the new models indicating which threats cannot occur (e.g. those whose opportunity rule evaluates to false).
- DST UI: the graphical user interface that allows the user to understand the level of risk to which they are exposed and the options available for recovery. This component has been updated since the PoC to rationalise the display of risks.

Details of the component design and implementation is provided in D5.2 [4].

There are two key integration points from WP5 components to other components in the testbed:

- Monitoring and management framework (WP3): the DST interfaces to the monitoring components through the DSMS, which uses long polling to fetch time-stamped metric values from the monitoring component as they become available.
- Ontologies (WP2): the system model (provided as a set of ontologies) is used by the incremental concrete model generator, which creates instances from the classes provided by it. The ontologies are also imported by the threat classifier and threat likelihood estimator when they update the concrete model.

7 Access Policy Modelling

The SERSCIS Access Modeller (SAM) takes a model of a system (e.g. a set of objects within a computer program or a set of machines on a network) and attempts to verify certain security properties about the system, by exploring all the ways access can propagate through the system. It is designed to handle dynamic systems (e.g. systems containing factories which may create new objects at runtime) and systems where behaviour of some of the objects is unknown or not trusted.

A typical approach is to model a system initially with most objects having defined (trusted) behaviour, and then to explore the effects of making them undefined or of adding additional actors with undefined behaviour into the system.

Once you have a model that meets the goals, it should tell you:

- what behaviours must be ensured for components you own
- what behaviours you require of other parties you rely on

Without a model (formal or not), we would never know whether it was safe to grant access to anything to anyone. Having a formal model (rather than simply relying on the programmers' and administrators' intuitions) is useful because:

- it reduces the chance of mistakes; and
- it makes assumptions explicit.

For example, if a security property could be enforced by adding a restriction in either one of two components being developed, each component developer might assume it would be added at the other point. Modelling the whole system forces us to make that choice and document it.

All the safety properties that are checked when building the initial system can be automatically rechecked when the system changes. When safety properties are checked manually when writing code (or deploying systems), changes to the system later can make the assumptions behind those checks invalid.

The SAM tool has been extended to make a wide range of modelling tasks simpler:

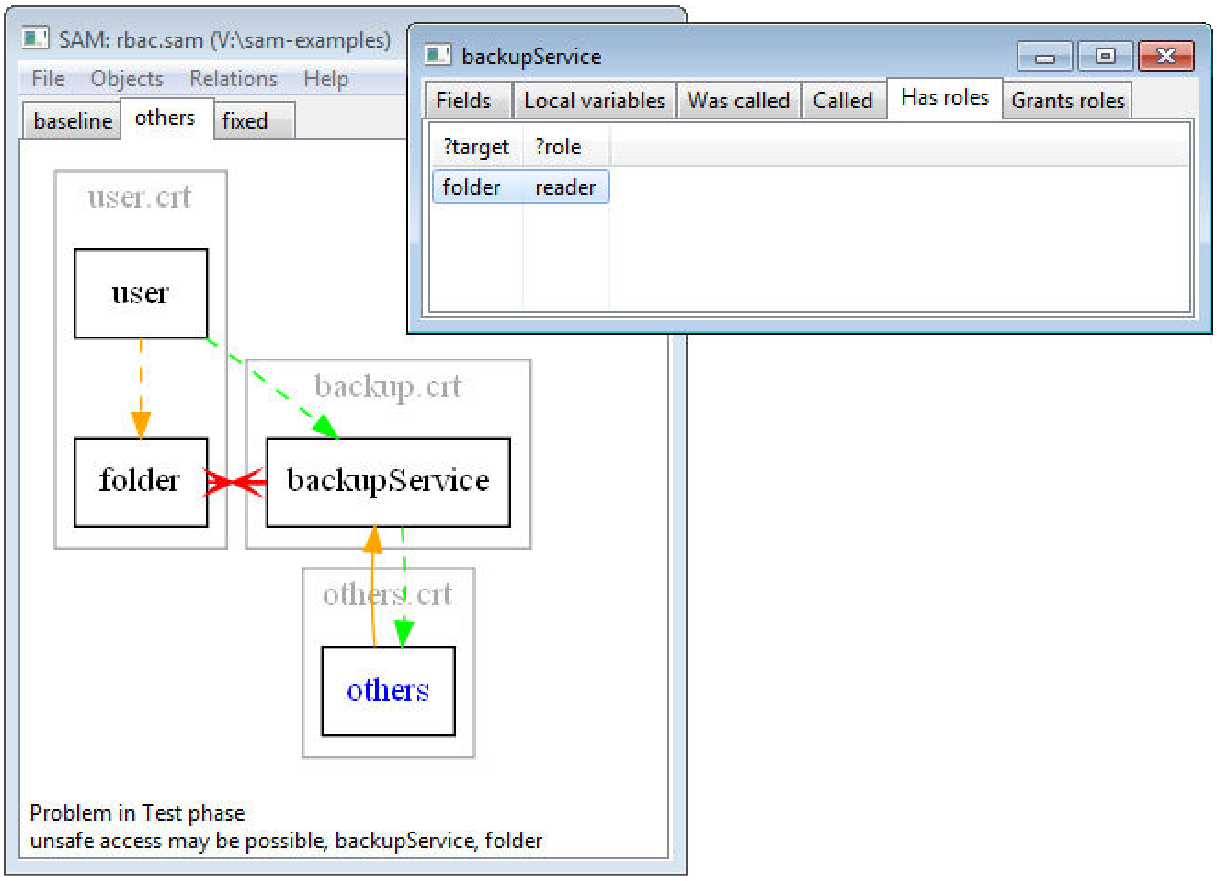
- Objects used to be represented as Datalog String objects. This could cause false positives due to confusion between object references and string literals, and caused Datalog stratification problems if a rule needed to check whether something was a String or a Reference using the type relation. A new separate reference type has therefore been added to avoid these problems.
- Predicate declarations now include types and a simple type-checker has been implemented to ensure that rules use the correct types.
- Support for literals has been extended to include Booleans.
- Wildcard "any" terms have been added to indicate any value of a particular type. For example, Unknown objects now automatically get access to "any(Value)", which avoids the need to model them having access to every constant in the system (which tends to produce confusing counter-examples containing irrelevant constants).
- Checking for new events caused by attackers previously required writing the baseline model, saving the results, then creating the attack scenario and running again. SAM now allows multiple scenarios to be defined in a single file. It evaluates all of them against the baseline scenario to generate the list of possible attacks in a single step.
- All invocations by Unknown objects are now grouped in a single context (the one in which the object was created). This avoids some false positives (for example, if a user in context "User" invokes a factory and an unknown object, and the unknown object invokes the factory, we no longer aggregate the two sets of objects created by the factory).
- To model validation checks, SAM now allows "if" statements in behaviour definitions. Previously, this required embedding some non-trivial Datalog in the models.
- The Unknown class now extends a new BaseUnknown class that implements the actual behaviour. Unknown objects then extend this with various useful defaults (they are assumed to be active by default, even if not invoked, they have a fully permissive access control policy, and they have references to all "public" objects). BaseUnknown can be used directly if these defaults are not desired (e.g. to model an object with unknown behaviour but a known access policy).
- Modelling of exceptions has been improved to allow modelling of objects that catch all exceptions (before, all exceptions were assumed to propagate) and modelling of systems where exceptions carry no authority (e.g. (Joe-E [23])).
- There is new syntax for modelling forwarding proxies, allowing the proxy to get the name of the method used to invoke it and to call a method with the same name on its target. This is useful for modelling access control features such as caretakers and loggers.
- Annotations can now be attached to fields (not just methods). The standard library uses this to define @FieldGrantsRole, for example, which allows the access control policy to be modelled as part of the object's state (and therefore updated through methods, etc).
- The baseline record now includes the context of each recorded call, allowing us to detect a call that was made due to a request from an attacker as opposed to the same call made by the same object on behalf of a trusted caller).

The standard library has been extended to include rules for role-based access control (previously, these rules had be defined by each model individually), overriding of the default context aggregation rules (allowing invocations to be grouped by argument value, for example) and overriding of the default rules for identity inheritance (which allows modelling systems such as service objects using GSI proxy certificates rather than the server's certificate).

A number of enhancements have also been made to the user interface (some of which can be seen in Figure 15):

- The debugger now has rules to provide an English description of many common predicates, which makes the proof-trees easier to read.
- Datalog predicates can be used to define additional tabs in the object information dialog, allowing the user to show custom information about each object. For example, a rule can be added to show the physical networks to which the object is connected, or the space it inhabits.
- It is now possible to double-click on a negative result in the debugger to see a list of rules which could have provided the missing fact. Previously, only the causes of positive facts could be followed.
- A scenario can now define rules for clustering objects in the output graph (e.g. grouping objects by the server they are running on).
- Unknown objects are now shown in blue rather than in red, so that red always indicates an actual problem.

The full list of changes can be found in the SAM version control repository, which is now available publicly [24]. We have also submitted a paper describing SAM to the ACM Transactions of Information and System Security (TISSEC) and has been accepted with some minor changes.



?target	?role
folder	reader

Problem in Test phase
unsafe access may be possible, backupService, folder

Figure 15: Updated SAM showing clusters and custom object information

8 Conclusions and Future Work

In this deliverable we have presented the updates on the system models after the initial PoC scenario, after taking into consideration the lessons learnt from the initial validation experiment. Further, we have extended the state of the art review after the previous deliverable D2.1b and compared our modelling approach to the existing risk management standards and methodologies.

The introduction of opportunities and secondary effects means that the number of SWRL rules in the dependability model has increased significantly. This is a problem from a model authoring point of view as the existing OWL ontology authoring tools such as Protégé have limited support for SWRL rules and from previous experience, it was noted that managing these rules was too laborious (due to the lack of edit/sort/enable/disable features). We have solved this problem with the development of a SWRL rule editor. Using the SWRL editor's easy to use interface, users can easily create new rules and manage existing rules. We have also developed a system model editor which allows system designers to create system specific assets and encode the relationships between these assets using diagrammatic views in an intuitive point and click interface. The editor was designed after observing the difficulties for no semantic experts in using ontology editors like Protégé. The output of the system model editor is an OWL ontology.

Future work will involve improvements to the system specific threat class generation algorithm (e.g. maintaining a separate threat ontology which is independent of the dependability model). Further, we will investigate the process of optimising the OWL models for faster reasoning and classification. A few initial experiments showed that the removal of inverse relationships between OWL classes improved the performance of the reasoner significantly. The classification time for an ontology with a large number of SWRL rules and hundreds of instances is still quite large (up to 40 minutes). We have developed a small model generator within the Incremental Model Generation component to reduce the classification time to less than 1 minute in most cases. Further research in this direction is necessary and will be continued in forthcoming projects.

9 References

[1] V Tsoulkas et al., "SERSCIS Deliverable D5.1b: Decision Support Tools: Revised Prototype Implementation," 2011.

[2] M Surridge, A J Chakravarthy, W Chen, M Hall-May, and T A Leonard, "SERSCIS Deliverable D2.1b: System Modelling Ontology and Tools: Revised Prototype Implementation v1.0," 2011.

[3] R Nossal and others, "SERSCIS D7.3: placeholder, details to be fixed later," 2012.

[4] V Tsoulkas and others, "SERSCIS D5.2 v2.0.Decision Support Tools: Full Prototype Implementation," 2013.

[5] S Fenz and A Ekelhart, "Formalizing information security knowledge," in *International Symposium on Information, Computer, and Communications Security*, Sydney, Australia, 2009.

[6] I Herzog, N Shahmehri, and C Duma, "An ontology of information security," *Internation Journal of Information Security and Privacy*, pp. 1-23, 2007.

[7] A Kim, J Luo, and M Kang, "Security ontology to facilitate web services description and discovery," *Journal on Data Semantics*, pp. 167-195, 2007.

[8] "COBIT 4.1," 2007.

[9] ISO/IEC 27005:2011. Information technology -- Security techniques -- Information security risk management, 2011.

[10] ISO/IEC 27001:2005. Information technology – Security Techniques – Information security management systems – Requirements, 2005.

[11] ISO/IEC 27002:2005 Information technology -- Security techniques -- Code of practice for information security management, 2005.

[12] R Shirey. (2000, May) RFC 2828: Internet Security Glossary. [Online].

http://www.ietf.org/rfc/rfc2828.txt

[13] (2010, August) MEHARI 2010 Risk analysis and treatment Guide.

[14] (2012) CRAMM V Information Security Toolkit.

[15] (2009, January) The Risk IT Framework Excerpt - ISACA.

[16] I Hogganvik and K Stolen, "A graphical approach to risk identification, motivated by empirical investigations," in *International Conference on Model Driven Engineering Languages and Systems*, 2006.

[17] R Matulevicius, H Mouratidis, N Mayer, E Dubois, and P Heymans, "Syntactic and Semantic Extensions to Secure Tropos to Support Security Risk Management," *Journal of Universal Computer Science*, pp. 816-844, 2012.

[18] C Blanco, J Lasheras, E Fernandez-Medina, R Valencia-Gracia, and A Toval, "Basis for an integrated security ontology according to a systematic reviww of existing proposals," *Comput. Standards & Interfaces*, pp. 372-388, 2011.

[19] A Vorobiev and A Bekmamdova, "An ontology-driven approach applied to information security," *Journal of Research and Practice in Information Technology*, pp. 61-76, 2010.

[20] "IT Grundschutz Manual," 2005.

[21] J Touzeau et al., "SESAR DEL16.02.01-D03: SESAR ATM Preliminary Security Risk Assessment Method.," 2011.

[22] (2013, January) ESPER. [Online]. http://esper.codehaus.org/

[23] Capability-secure subset of Java. [Online]. http://code.google.com/p/joe-e/

[24] T.A Leonard, M Surridge, and M Hall-May, "Modelling Access Propagation in Dynamic Systems," *Submitted to ACM Transactions on Information and System Security*, 2012.

[25] SERSCIS: Semantically Enhanced Resilient and Secure Critical Infrastructure Services, Annex I Description of Work, v1.6, Sep. 14, 2011.