STRUCTURED SYSTEMS ECONOMICS FOR SECURITY MANAGEMENT

ADAM BEAUTEMENT AND DAVID PYM

ABSTRACT. We develop an ontological account of information security architectures that is inspired by economic models of trade-offs between confidentiality, integrity, and availability. Our approach clarifies the nature of the trade-offs by making a clear distinction between declarative and operational concepts in security. We integrate this approach with a semantically justified mathematical systems modelling technology, thus providing a basis for a systematic methodology to support operational decision-making in information security investments and trade-offs.

1. INTRODUCTION

Organizations deploy systems technologies in order to achieve their business objectives. Typically, it will be necessary for an organization to invest in deploying information security policies, processes, and technologies in order to protect the confidentiality, C, integrity, I, and availability, A, of its business processes. Faced with a finite security budget, managers must choose how to deploy their resources to protect each of these aspects of their operations; that is, they must accept that there are trade-offs to be made. We have argued elsewhere that utility theory — particularly, but not exclusively, as employed in macroeconomics and financial economics — provides a suitable framework for characterizing and exploring such trade-offs [24], building on pioneering work such as Anderson [1] and Gordon and Loeb [17, 18], among others.

Whilst CIA provides an elegant framework for organizing security objectives, many practitioners and researchers have found it convenient — when faced with the need to implement — to consider a range of other concepts, such as access control, authentication, and non-repudiation, among others, alongside CIA [38]. Unfortunately, these additions do not lead to a systematic analysis of security requirements and their implementation. Rather, they tend to confuse the objectives of information security operations with the mechanisms that may be employed in order to achieve those objectives [38], and the goal of a systematic approach to security management remains unachieved.

In this paper, we propose an ontological framework for integrating the economic analysis of systems managers' preferences between investments to protect against confidentiality, integrity, and availability and the structure and dynamics of the system itself. Specifically, we develop a hierarchical ontological model of security needs and objectives within an organization. The purpose of the ontological model is to represent security functionality in such a way that the model can be used as a tool to aid in both the design and subsequent audit of the security behaviour of an organization. It represents and expresses the security design and implementation choices of the organization in terms of a hierarchy of needs. It also aims to reveal the tensions between the business and security processes, and also the trade-offs that are taking place between the various security objectives of the organization.

The ontological framework provides a conceptual account of security architecture. Its structure, which can be expressed in terms of processes, resources, locations that form the system itself, together with the events that are incident upon the system from its environment, can be captured within a process-algebraic and logical mathematical modelling framework [13, 10, 11], for which tool support is available [12, 11]. These mathematical models allow us to explore the consequences of design decisions in the security architecture using both the (Monte Carlo) methods of discrete even simulation and methods of (formal) logical reasoning.

Neither the ontological framework nor its mathematical realization, however, provides a convenient tool for security managers. We seek to fill this gap by provide an account of our framework in a familiar ontology language. We take OWL [37] as a generic choice. Our OWL representation captures the concepts of our conceptual ontological framework and structures them in a way that corresponds to their mathematical

realization. Thus the OWL interface allows managers, without the need for them to learn complex modelling technologies, to place their knowledge and experience in context that supports rigorous reasoning about their investment requirements. The diagram in Figure 1 below summarizes this situation.

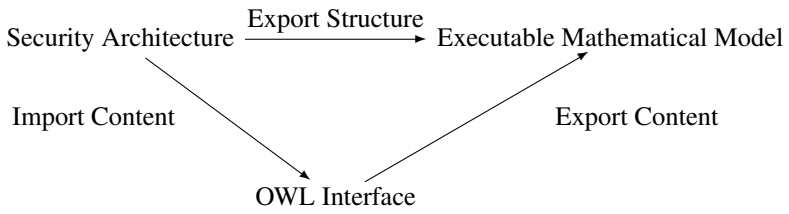


FIGURE 1. The Organization of the Methodology

The remainder of this paper is organized as follows: in Section 2, we discuss the fundamental concepts of information security, identifying the need for a clear distinction between declarative and operational concepts as a basis for understanding how economically characterized trade-offs may be implemented; in Section 3, we introduce our ontological framework for describing security archtitectures; in Section 4, we explain our mathematical and computational realization of the framework, including a brief discussion of our approach to logical reasoning; in Section 5, we provide a sketch of how OWL might provides a syntactic interface to our semantic modelling technologies; finally, in Section 6, we discuss our ongoing and further work in this area. Throughout the paper, we illustrate our ideas with an example that we hope will be familiar to most readers: airport security. This rich example has many features of both information an physical security, and brings many concepts into clear focus.

2. THE CONCEPTS OF INFORMATION SECURITY

The fundamental concepts of information security are confidentiality, integrity, and availability (CIA). Alongside these notions sits the concept of privacy. We are not concerned with privacy in this paper.

In this section, we discuss the status of the basic concepts of information security and how they are related to the similarly basic operational mechanisms that are used to implement them. We consider how methods from economics can be used to inform design choices, and introduce an example that will run throughout this paper.

2.1. **The Basic Concepts.**

It is, however, commonplace in the literature to find observations that CIA does not provide an adequate basis for practical, operational, information assurance. Typically, it is suggested to extend CIA with various additional concepts, such as 'authentication', 'non-repudia(tion)(bility)', 'control', or even 'utility'. Perhaps the leading, most developed example is the 'Parkerian Hexad' (as developed in Parker's elegant account of security concepts [38]), in which to

- confidentiality,
- integrity, and
- availability

are added

- possession,
- authenticity, and
- utility.

These concepts, which it has been argued are 'atomic' and 'non-overlapping' are, indeed, all conceptually valuable and pragmatically useful in the understanding and practice of information security.

A similar collection of concepts may be found, for example, in the 'ISO/IEC 7498-2: Information Technology—Open Systems Interconnection—Basic Reference Model—Part 2: Security Architecture', which identifies

- identification and authentication,
- access control,
- data integrity,

- data confidentiality,
- data availability,
- auditability, and
- non-repudiation.

In the Parkerian Hexad and the ISO/IEC Reference Model, however, as in similar taxonomies, the proposed extensions to CIA constitute, in the Aristotelian sense, category errors. They confuse the (declarative) objectives of information security operations with the (operational) mechanisms deployed in order to achieve those objectives. For one example, access control is an operational notion used, for example, to restrict the availability of a service to a given group of users. For another, 'auditability' , 'authenticity', and 'non-repudiation' are properties — of the kind that might be expressed logically, as discussed below (in 4.3) — of the underlying systems security architecture. For yet more, possession seems to be a notion that is derivable (at least according to Parker's definition) from confidentiality, integrity, and availability, whilst utility is, evidently, not a security concept, but rather an economic concept that is useful in the security context

This situation is problematic not only from the conceptual point of view — because declarative and operational concepts must be treated differently in order to understand how objectives are delivered (or not) by making (in)appropriate implementation choices — but also from the economic and management points of view — because we are concerned with how the objectives of information security measures trade off against one another.

There are also evident redundancies — as discussed in, for example, [38, 35] — though this issue is not our primary concern here.

2.2. **An Economic View.** The economic point of view is our driving concern in this paper. We seek to provide a model of information security that captures the (undoubted) value of ideas such as the Parkerian Hexad in framework that supports a direct and simple integration with the basic components of executable models of the underlying system, which can be explored by simulation, Monte Carlo experimentation, and model checking. A key component of this integration concerns utility.

Utility is an economic concept. Utility theory (see, for example, [28, 47]), particularly as developed in the contexts of macroeconomics and financial economics, provides a highly expressive framework for representing the preferences of the managers of a system.

For example (e.g., [44]), in the macroeconomic management of market economies, central banks play a key rôle. The managers of a central bank are given, by their national governments, targets for certain key economic indicators, such as unemployment (u_t) and inflation (π_t) at time t (time can be either discrete or continuous here). Their task is to set a (e.g., monthly) sequence of controls, such as their base (interest) rates (i_t) so that the key indicators are sufficiently close to their targets, $\overline{u_t}$ and $\overline{\pi_t}$, respectively. Typically, using this example, the managers' policy is expressed as a utility function

$$U_t \quad = \quad w_1 f_1(u_t - \overline{u_t}) + w_2 f_2(\pi_t - \overline{\pi_t}) \tag{1}$$

together with system equations, $u_t = s_1(i_t)$ and $\pi_t = s_2(i_t)$, expressing the dependency (among other things) of u and π on interest rates in terms of functions s_1 and s_2 that describe the (macro) dynamics of the economy. Two key components of this set-up are the following:

- The weights w_1 and w_2 (typically, values between 0 and 1) that express the managers' preference between the components of the utility function — that is, which they care about more; and
- The functions f_1 and f_2 that express how utility depends on deviation from target. A simple version of this set-up would take the f_is to be quadratic. Quadratics conveniently express diminishing marginal returns as the indicators approach target, but make utility symmetric around target. More realistically, Linex functions [51, 53, 44], usually expressed in the form $g(z) = (exp(\alpha z) - \alpha z - 1)/\alpha^2$ are used to capture a degree of asymmetry that is parametrized by α.

The managers' task, then, is to set a sequence of interest rates i_t such that the *expected* utility, $E[U_t]$, remains with an acceptable range, as u_t and π_t vary, and trade-off against each other, as the sequence of rates i_t evolves. In general, there can of course be as many components as required in a utility function.

This economic framework can be deployed in the context of information security (see, for example, [3, 24, 23, 5]), where concepts — such as confidentiality, integrity, and availability — that lie within competing declarative categories can be seen to trade-off against one another as the relevant controls —

such as system configurations or investments in people, process, and technology system configurations — vary.[1]

To formulate an analysis of this kind, we follow, *mutatis mutandis*, the prescription outlined above.

- The organization that deploys information security measures exists in an economic and/or regulatory environment. This environment places constraints upon the systems and security architectures available to the organization's managers.
- The managers formulate a utility function that expresses their policy preferences, which will depend upon the nature of their organization. For example, state intelligence agencies and online retailers will have quite different priorities among confidentiality, integrity, and availability; see, for example, [24].
- In a highly complex situation, such as a security architecture, it will typically not be possible to formulate system equations (in terms of functions s_1 and s_2) in the way that is usually possible in, for example, macroeconomic modelling. Typically, though, the key control variables, such as system interconnectivity or investment in various aspects (people, process, and technology) of security operations, will be identifiable.
- Instead, however, an executable system model [12], using the key control variables, can used in order to simulate the the dynamics of the utility function.

The last step raises a big question, however: How should a system model, that captures appropriately not only the inherent structural and dynamics for the system, but also its security architecture, be formulated? In the next three sections, we address this problem. We start, in Section 3, with a (conceptual) ontological account of security architectures, which makes a clear distinction between (declarative) objectives and (operational) implementation choices. We then explain, in Section 4, how the system model required to simulate the utility function can be constructed so as to represent accurately the security architecture. Then, in Section 5, we explain how an ontology language (we use OWL [37] here) can be used to capture the conceptual ontology for representing security architectures in a way that facilitates the construction of system models.

Thus, in practice, the modeller would express the security architecture of the system in the relevant ontology language template, so providing a basis for discussion and refinement with the system's managers, and prototypical structure for implementation as an executable model.

2.3. **Our Running Example.** To illustrate our approach, we will develop an example that is, in fact, not purely an information security example. Rather, it is an example that involves both physical and information security concerns, about access control and information flow, and which illustrates the much broader applicability of our approach to the concepts of information security. It is also rather familiar.

Consider the management of an airport's security process [2, 46, 49, 52, 34]. A key aspect of this is checking passengers and their bags for acceptability to fly. A passenger, with luggage, must navigate from the concourse of an airport's terminal building to a seat on an aircraft; that is, it is an access control process that is predicated on maintaining an *integrity* property of aircraft, and this is achieved by maintaining that property for passengers and their baggage. This integrity property trades-off against other concerns, principally costs, incurred in providing security staff and equipment, and service availability.

The passenger is subject to a range of security controls that are intended to ensure a certain integrity property — roughly, that certain dangerous or substances and objects are not present — of the aircraft. The manager of the security process has decided that, in order to access the aircraft, passengers must submit to the security process and must therefore sacrifice their confidentiality — the bags will be searched, there will be body searches. Thus the manager has given a preference weighting of 0 to passengers' confidentiality. There are, however, non-trivial trade-offs with cost and availability:

- *Cost*. The effectiveness of the integrity might be improved by, for example, introducing more expensive scanning devices that are able to detect more things, more reliably. The efficiency of the integrity check might be improved by introducing more scanning devices and more security staff, thereby facilitating greater parallelism in the security process;

[1] We remark, without any development, that it seems that *(Cumulative) Prospect Theory* [50] might be deployed as as useful alternative to the use here of expected utility.

- *Availability*. An important measure of availability[2] of access to the aircraft is the length of time that must be allocated for passengers to navigate the airport's security procedures.

We can, of course, consider very different points of view. From the point of view of a smuggler, the utility function might look rather different. She might submit to the process without deciding to sacrifice her confidentiality — her preference weighting for confidentiality is not 0, but rather is something close to 1 — hoping to conceal her contraband by some means. We will not develop this part of our example here because it is not concerned with the integrity property of the aircraft. Rather, it is concerned with an integrity property of an international boundary (be it outgoing or incoming), and, in practice, may or may not be considered within the security process.

3. Modelling the Security Architecture

In this section, we introduce our conceptual-ontological account of security architectures. The purpose of the ontology is to give a structured, conceptual description of the components of a security architecture that can naturally be integrated with the natural structure of executable system models (see Section 4, [12]) and which can be conveniently represented in an ontology language, such as OWL [37].

There are two key layers in our representation of a security system, the *Framework* layer and the *Instantiation* layer. There is a commonality of organization between these layers although they represent conceptually different parts of the model. Both layers are organized into a hierarchy of rôles with each rôle sub-divided into dependencies, priorities, and preferences.

The hierarchy contains all the relevant rôles that make up the organization being modelled. Rôles are ordered by their ability to influence the security architecture of the system. In other words, they are classified by the toolbox that is available to them for modifying *security objects* (that characterize security tasks, defined below). The system accepts multiple and partial orderings. For example, the top level of the model might represent the strategic decision-makers of the organization, such as an airport's security managers or their regulators, while the bottom level might represent an individual employee or user of the organization, such as an airport's check-in staff or a passenger navigating airport security. The rôles represent the possible positions individuals can adopt in the hierarchy. They do not represent any entity themselves. They are instead populated by *actors*, which are another component in system and are described below.

Each hierarchy level contains three sections representing the dependencies, priorities and preferences of that level. For our purposes we define the terms as follows:

- *Dependencies* (strong requirement): Externally enforced requirements that actors populating the rôle must meet all of in order to function within the model. Actors occupying this rôle have no choice in whether or not (and possibly even how) to meet these requirements regardless of how resource inefficient they are. Dependencies will often be informed by the environment within which the hierarchy exists;
- *Priorities* (weak requirement): Externally supplied tasks, as many as possible of which should be met by Actors in the associated rôle. Actors have some choice in which priorities to meet and how they are approached. In a limited resource environment, Actors can select the most resource efficient priorities and methods first. Priorities will often be informed by the rôle that the level represents;
- *Preferences*: Actor-generated tasks that the Actor has decided are worth achieving from its own perspective. These can be generated by the Actor's inclusion in other hierarchies.

Dependencies, priorities, and preferences (DPP) and the hierarchy of rôles structure are found in both the framework layer and the instantiation layer.

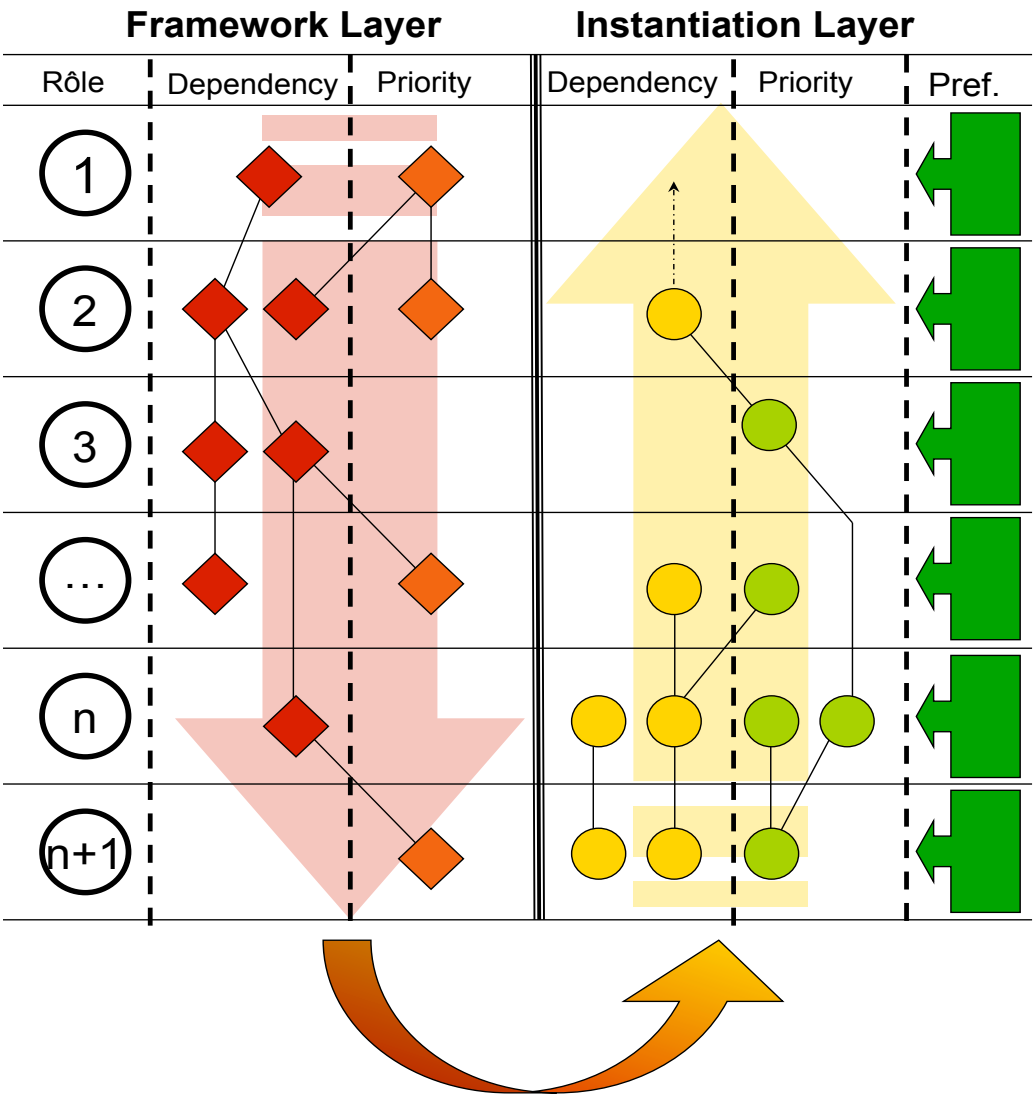
The form and construction of the security architecture is illustrated in Figure 2.

The key components of this diagram are the following:

- *The hierarchy of rôles* (far left). Rôles capture the relevant security management structure of the organization being modelled. They are ordered by their ability to influence the security architecture of the system;
- *The Framework Layer* (centre left). The Framework Layer is constructed top-down. Dependencies and priorities at a given level in the hierarchy induce dependencies and priorities at lower levels;

[2]Recall that one reasonable definition of availability is along the lines of 'accessibility of service when required' [38].

FIGURE 2. Security Architecture: Framework and Instantiation



- *Security Objects* (trees within Framework Layer). Security Objects represent the security tasks which, if completed, will satisfy the dependencies and priorities with which they are associated;
- *The Instantiation Layer* (centre right). The Instantiation Layer is constructed bottom-up, starting where the Framework Layer finishes (see below). The Instantiation Layer is a populated image of the Framework Layer;
- *Security Components* (nodes of trees within Instantiation Layer). Security Components perform the operational checks required in order to deliver Security Objects. They do so by return boolean values up the tree, towards the root. They enter the architecture when the Framework is instantiated;
- *Actors* (far right). Actors occupy rôles. They insert preferences into the hierarchy of rôles at the Instantiation Layer.

A key point here concerns the way in which the dependencies, priorities, and preferences in model are intimately related to the declarative security concepts. For example, given confidentiality, integrity, and availability as the concerns, the dependencies, priorities — in the Framework Layer — express the policies required to implement the managers' preferences as represent in their chosen utility function for the organization. Specifically, given a utility function with definiens of the form

$$w_1 f_1(C - \overline{C}) + w_2 f_2(I - \overline{I}) + w_3 f_3(A - \overline{A}) + w_4 f_4(K - \overline{K})$$

where C, I, and A denote suitable instances and/or measures (see, for example, [24]) of confidentiality. integrity, and availability, respectively, and where K denotes cost or investment, dependencies will have very high weightings, with very little tolerance for deviation from target, and priorities slightly lower weightings. Preferences (see below) have weightings that lower still, and may have high tolerance for deviation from target.

The form and function of the Framework and Instantiation Layers, and the interaction between them, will now be discussed in more detail.

3.1. **The Framework Layer.** The Framework Layer represents the underlying structure of the system. It is static (in the sense that the model does not run on this layer) and declarative but informs the construction of the more operational Instantiation Layer. A completed Framework Layer consists in a hierarchy of rôles (see, for example, [2]) with dependencies and priorities assigned to them. As preferences are derived from actors (see below), they do not appear in this layer because actors appear in the Instantiation Layer. The dependencies and priorities will each have a *Security Object* (SO) assigned to them. SOs are a unique component of the Framework Layer and represent the security tasks which, if completed, will satisfy the dependencies and priorities with which they are associated.

For example, in the setting of the running example of airport security that we have begun to introduce, examples of Security Objects include the examining of checked luggage, the checking of hand luggage and passengers — to identify and so remove any prohibited contents — and the tracking of the relationship between passengers and checked luggage. These examples are developed below.

SOs are unconstrained with respect to their location within the hierarchy. SOs can only exist in one hierarchy and never populate multiple hierarchies. This is a key difference between Actors and SOs. One of the aims of this formulation is to improve the communication between different stakeholders and eliminate the duplication caused by the failure to understand the connectedness of security concepts between levels. In practice, that means a typical SO will exist at multiple levels and multiple sections (dependency, priority) in the Framework. It will commonly be the case that an SO created at a higher level will transition through and connect (or create) priorities and dependencies lower in the framework.

For the more mathematically minded reader, there are many choices of formalization of SO. Our working choice for the purposes of this paper is, roughly speaking, the following:

- SOs are characterized by (directed) and/or forests[3] [4] (illustrated in Figure 2 by the red/orange trees in the Framework Layer) associated with dependencies and priorities;
- Internal nodes of the trees are labelled with boolean variables, each associated with a dependency or priority, and truth conditions are inherited upwards (towards the root);

[3]A (directed) forest is a disjoint union of (directed) and/or trees.

[4]A forest is required because a given SO may, in general, derive from more than one dependency or priority.

- Leaves are nodes for which a boolean instantiation (all components for conjunctions, one component for disjunctions) can be determined at the next level down in the hierarchy of rôles.

The Framework is populated with dependencies, priorities, and security objects through an iterative process that requires design input from an expert source. The criteria under which security objects terminate and by which the Framework can be said to be complete is the same for all frameworks created in this way.

As indicated above, dependencies and priorities are externally generated. In practice, a hierarchy of rôles will not encompass all possible contributors to the framework and will have been bounded at some sensible level. In our example, we have not represented any rôle higher than the airport security manager in our hierarchy. The creation and bounding of the hierarchy of rôles is the first step in creating a framework layer. To populate a framework, it is necessary to determine the dependencies and/or priorities that the top rôle in the hierarchy will inherit from sources external to the hierarchy. At this stage, we will have a complete hierarchy of rôles that is empty of dependencies and priorities except for the top layer. The next step is to assign security objects to these dependencies and priorities that will allow them to be fulfilled; see Table 1.

TABLE 1. SOs Stage 1

Rôle	**Dependencies**	**Security Objects**
Airport Security Manager	Ensure no prohibited materials transit the airport	Scan checked luggage Scan hand luggage and passengers Track relationship between passengers and checked luggage

At this point the iterative process begins. The construction of a framework always proceeds from top to bottom. At each iteration the following are checked:

- Are there any dependencies or priorities without an assigned Security Object?
- Are there any unterminated Security Objects?

The construction of a Security Object terminates once it is possible to return a boolean value from its lowest point. If this is not possible, then the Security Object must be extended to the rôle below in the hierarchy, creating any necessary dependencies and priorities as it does so. The dependencies and priorities created will be informed by the rôle creating them; thus, as the SO descends through the hierarchy, it will become more detailed as the scope of the lowers is necessarily more limited. In our example above none of the security objects can return a value and thus need to be extended. Let us extend the 'scan hand luggage' SO. The following (Table 2) would be the result of two iterations, one would generate the dependency from the SO above, the second would find a dependency without an assigned SO and create one:

TABLE 2. SOs Stage 2

Rôle	**Dependencies**	**Security Objects**
Airport Security Manager	Ensure no prohibited materials transit the airport	Scan checked luggage Scan hand luggage and passengers Track relationship between passengers and checked luggage
Airport Security Staff	Examine all passengers and luggage passing through security checkpoint	Identify contents of hand luggage and verify permitted

Note, for example, that the SO 'scan and luggage and passengers' corresponds to a tree (red/orange in Figure 2) in the Framework Layer.

At this point the SO can return a boolean (true/false that the contents of the bag are permitted) and will terminate. The framework is not yet complete, however, as there are still unterminated SOs at the manager layer. Iterating in this fashion would also close those at a suitable point. The final step in a SO is always a compliance step which indicates that at this level and below the rôles in the hierarchy simply comply with the SO and are not involved in its execution. This would add the following line (Table 3) to the framework:

TABLE 3. SOs Stage 3

Rôle	**Dependencies**	**Security Objects**
Passenger	Comply with SO	

Once the framework is complete under the criteria outlined above the SOs will form a Bolean forest with the leaves connecting each dependency and priority in the framework. At this point we can begin to construct the instantiation layer.

3.2. **The Instantiation Layer.** Where the framework layer is static and declarative the instantiation layer is dynamic and operational. Two new parts of the architecture are added during instantiation, Security Components (SC) and Actors. Actors will be discussed in more detail below; for now it is sufficient to know that they occupy rôles and insert preferences into the hierarchy of rôles at the Instantiation Layer. Security components combine together to form the operational counterparts of security objects.

The instantiation layer needs building in the same way that the framework layer did. Again an iterative process is adopted with certain termination criteria. The key difference here is that this layer is built bottom up. SCs lay out the processes and resources needed to perform the boolean checks specified in corresponding SOs. SCs start at the final 'compliance layer of the SO. Once the processes and resources required at this level are put in place we check to see if they are sufficient to complete the SO. If yes, then the SC terminates. If not then we move up to the rôle above and add additional processes and resources as needed. Again, this process repeats until all SCs are closed. At this point, the Instantiation layer is complete.

A little more formally, corresponding to the slightly more formal view of SOs sketched above, we can describe how SCs are combined to instantiate SOs as follows:

- SCs are combined according to the and/or forest determined by the SO that they instantiate;
- Each SC implements a checking process that applies to Actors at the level below;
- SCs return boolean values that instantiate internal nodes of the corresponding SO.

Working through our example again we start at the passenger level and work upward until we have sufficient processes and resources in place to return a boolean for the statement 'the passengers possessions and luggage are permitted. The finished security component in this case would be as follows (Table 4):

In Figure 2, the SCs correspond to the green/yellow nodes in the Instantiation Layer

Note that whereas the SO terminated in a 'compliance' level the SC terminates at a 'provision' level when it reaches a rôle that can sufficiently provide the resources required to execute the SC without recourse to a higher rôle.

The final component of the instantiation layer (and the model) are Actors. Actors exist independently from any single security hierarchy. They represent entities that transition between hierarchies, this being the key difference between Actors and SOs. They can interact with any and all hierarchies present, simultaneously if necessary. Actors exist solely as a collection of tags corresponding to their attributes. When an Actor interacts with a hierarchy and seeks to populate one of its rôles, the hierarchy examines the Actors tags (some or all of which may be unreadable to the hierarchy). It assigns the Actor to a rôle based on its tags and clones a copy of that rôle for the Actor to inhabit for the duration of its lifecycle in that system. This clone inherits the relevant dependencies and priorities of the rôle. Its preferences are obtained by interrogating the Actors tag-cloud. Some Actors will have no preferences. Such Actors represent items such as inanimate objects (e.g., a passenger's bags in the airport example) or data stores (e.g., a baggage handling label in the airport example) that can be passed between hierarchies (the departure and destination airports will have separate security systems and this will be considered separate hierarchies) but have no

TABLE 4. SCs

Rôle	**Dependencies**	**Security Components**
Airport Security Manager	Ensure no prohibited materials transit the airport	Provide resources (X-ray machine, metal detector, wands) Provide data on prohibited materials for X-ray comparison
Airport Security Staff	Examine all passengers and luggage passing through security checkpoint	Monitor X-ray machine and inspect results for prohibited items Hand-search suspect luggage Hand-scan suspicious passengers
Passenger	Comply with SO	Place luggage on scanner Walk through detector

intentions of their own.[5] Once a clone is created the Actor no long directly interacts with the hierarchy for the duration of its lifecycle.

The Actor clone is now a full part of the hierarchy and can interact with its associated Security Components, updating its dependencies, priorities, and preferences as necessary. Its actions will determine the paths and states of the hierarchy it passes through. These in turn decided the exit point of the actor clone from the hierarchy. Each possible access point has a set of tags associated with it that are then passed back to the core Actor to replace the set used to interact with the hierarchy. These may be identical or contain new or changed statuses. For example, a hierarchy designed to check the citizenship of an unknown actor may return tags, which it did not previously possess, identifying it as a national or an illegal alien.

The introduction of Actors adds a third dimension to the framework. Multiple actors can exist on any one layer of the hierarchy (and, typically, the lower levels will have more actors assigned to them). This also means that Security Objects can span multiple actors as well as multiple levels.

3.3. **Multiple and Interacting Hierarchies.** A given organization may have multiple hierarchies associated with it. The airport example discussed so far has been focussed on the hierarchy that is concerned with controlling access to aircraft; specifically, with ensuring that the passengers and baggage that board the aircraft satisfy certain integrity properties. These integrity properties trade-off against costs — deriving from the provision of staff and equipment — and the availability of the airport's core service; that is, to facilitate travel.

Considering availability, however, provides a link to another hierarchy: the airport's commercial business process. Here the priorities are less concerned with service integrity issues and more with service availability issues. Just as an online retailer will give very high priority to website availability, so the airport's managers will give high priority to placing commercial operations in locations that will optimize revenue without compromising the security operations. Overall, the business process hierarchy will be quite different to the security hierarchy. There may also be interacting security hierarchies: here, we have focussed on passengers and their baggage, but their is also airport perimeter control, for example. Moroever, different hierarchies may interact over time. For example, we have described (aspects of) the security process at the aircraft's origin, but there is also a process (with very different priorities) at its destination.

An example of the choices available when hierarchies interact is afforded by a comparison of the strategies of the principal airports in Bristol and Brussels. At Brussels, most of the commercial operations

[5]In a richer formulation of our set-up, it may be necessary to distinguish between Actors with no preferences and Actors that are unable to exercise their preferences.

(stores, restaurants) are located between immigration control and security control. This has several possible effects. For example, once in the restricted area beyond immigration, passengers may choose to spend time (and money) in the commercial operations whilst waiting for security queues to clear. Some of the items purchased may not be permitted through security control, but cannot be checked-in without exiting immigration. On the other hand, the restricted area may become dangerously congested during certain operational difficulties. At Bristol, however, immigration control and security control are in immediate sequence, with commercial operations present only before immigration and after security. This choice has a different set of consequences for the interaction between the security and business hierarchies.

So different, interacting hierarchies exist for the same organization (and, of course, the hierarchies associated with different organizations also interact with one another). We defer an account of interacting hierarchies to another occasion.

4. Modelling the System Architecture

In the previous section, we have described how the security architecture of a system can be captured in conceptual terms that draws appropriate distinctions between declarative concerns, together with their associated trade-offs, and the associated operational concerns that implement them. In this section, we explain how we model the underlying system architecture, how it supports our model of the security architecture, and how it relates to utility.

4.1. The Components of a System Model.

The security architecture explained in Section 3 has, implicitly, been described in terms of *processes*, *resources*, and *locations*. We describe below how a systems modelling methodology based on these concepts works both mathematically and computationally.

- Processes — the activities that occur in order to deliver security operations (for example, in the airport, scanning of checked baggage).

  Mathematically, our starting point is Milner's synchronous calculus of communicating systems (SCCS) [31], perhaps the most basic of process calculi, the resource semantics of bunched logic, and a basic notion of location [13, 10].

  The basic idea is that processes evolve by performing actions, and in so doing modify the current location and the resources available. This co-evolution of locations, resources, and processes is described by a modification function $(L', R') = \mu(a, L, R)$, which is specifies that the basic action a, when applied to resources R at location L, returns resources R' at location L'. This is written as follows:

  $$L, R, E \xrightarrow{a} L', R', E'$$

  Processes are constructed from basic actions using combinators for action prefixing, concurrent composition, non-deteministic choice, local resources, and recursion.

  The most basic step is that for basic actions guarding a process, $a : E$, with the following operational semantics:

  $$\frac{}{L, R, a : E \xrightarrow{a} L', R', E} \; Action$$

  where $(L', R') = \mu(a, L, R)$. For an example of a more complex combinator, consider concurrent product of processes, $E \times F$:

  $$\frac{R, E \xrightarrow{a} R', E' \quad S, F \xrightarrow{b} S', F'}{R \circ S, E \times F \xrightarrow{ab} R' \circ S', E' \times F'} \; Product$$

  Here we make use of a monoidal combination or resources, as discussed below. The theory of this calculus is developed in full detail in [13, 10].

  Computationally, this point of view is captured in the modelling language Gnosis [12]. The code fragment in Figure 3 gives a flavour of how models are built.

  The process for checking hand luggage first claims some resources (discussed below): a security staff member, an X-ray machine, and a piece of luggage. Then a scan is launched. Either the scan returns true, in which case the luggage is passed and the resources claimed are released, or false, in which case a manual search process is launched. Of course, we could choose to model this aspect of the security architecture in much greater detail.

```
process checkHandLuggage =
claim (1) securityStaff @ SecurityCheckpoint
claim (1) xray @ SecurityCheckpoint
claim (1) handluggage @ SecurityCheckpoint

launch(scan)

(*
scan returns true if luggage approved, otherwise false
*)

if scan = true
release (1) securityStaff @ SecurityCheckpoint
release (1) xray @ SecurityCheckpoint
release (1) handluggage @ SecurityCheckpoint

else
launch manualSearch
```

FIGURE 3. The Process to Scan Hand Luggage

Semantically, multiple processes executing together are interpreted using the concurrent composition structure described above (note that the synchronous calculus is able to express asynchrony [31]).

- Resources — the logical and/or physical entities that enable and are manipulated by the processes (for example, in the airport, the scanning machines or the bags they scan).

  Mathematically, our notion of resource — which encompasses natural examples such as space, memory, and money — is based on (ordered, partial, commutative) monoids (e.g., the non-negative integers with addition, zero, and less-than-or-equals), which capture basic conceptual notions of resource composition and comparison:
  - Each type of resource is based on a basic set of resource elements;
  - Resource elements can be combined (and the combination has a unit);
  - Resource elements can be compared.

  Formally, we take pre-ordered, partial commutative monoids of resources, $(\mathbf{R}, \circ, \mathbf{e}, \sqsubseteq)$, where $\mathbf{R}$ is the carrier set of resource elements, $\circ$ is a partial monoid composition, with unit $\mathbf{e}$, and $\sqsubseteq$ is a pre-order on $\mathbf{R}$, which gives rise to the evident equality.

  Computationally, the modelling language Gnosis employs resources, such in two main ways. First, as 'shares', for which processes may compete; second, as 'bins', the basis for systems of queues. Again, see [12] for more detail.
- Locations — the places (for example, in the airport, the security checkpoint) where things happen.

  Mathematically, just as our treatment of resources begins with some basic observations about some natural and basic properties of resources, so we begin our treatment with the following basic requirements of a useful notion of location [10, 11]:
  - A collection of atomic locations — the basic places — which generate a structure of locations;
  - A notion of (directed) connection between locations — describing the topology of the system;
  - A notion of sublocation (which respects connections);
  - A notion of substitution (of a location for a sublocation) that respects connections — substitution provides a basis for abstraction and refinement in our system models;
  - A product (again, monoidal, also written $\circ$ for now) of locations (an inessential but useful technical property), suitably coherent with the other products [10].

The notions of sublocation and substitution are intimately related, the former being a prerequisite for the latter. We will not develop or implement substitution in this paper (except for brief comments in examples) rather deferring it as a next step.

Computationally, the Gnosis modelling tool declares locations and directed links between them. The code fragment in Figure 4 gives an example.

```
location External, CheckinDesk,  CheckedSecurity, CabinSecurity,
         Aircraft

link External <-> CheckinDesk
link CheckinDesk -> CheckedSecurity
link CheckinDesk -> CabinSecurity
link CheckedSecurity -> Aircraft
link CabinSecurity -> Aircraft
```

FIGURE 4. Locations and Links

Thus there is a two-way link between the locations `External` and `CheckinDesk`, but all the other links here are one way. We use `CheckedSecurity` and `CabinSecurity`, respectively, to denote the locations of the security checkpoints for the hold and the cabin baggage/passengers.

As we have seen in the process example above, Gnosis allows for resources to be declared at locations: for example,

```
share staff @ CheckinDesk (Numdeskstaff)
```

declares `Numdeskstaff` members of staff at the check-in desk.

Related work in this area includes that of Milner and his followers [33] and that of Hillston et al. [22, 15, 21].

Such an architecture works well conceptually, but it is *isolated* — that is, it is not connected to the environment within which it exists. For example, the processes that describe the airport's security operations apply to passengers (and their luggage) who arrive at the terminal building from the outside world. Clearly, for our present purposes at least, we are not interested in modelling the outside world in any detail. Nevertheless, we must have someone way of modelling passenger arrivals at the boundary of our system of interest.

In our system modelling point of view, in common with established practice in discrete event simulation [6], the approach take is to employ stochastic methods:

- Environment — the incidence of events upon the system of interest is represented by an appropriate choice of probability distribution. For example, the arrival of passengers at the airport's terminal building might be captured by a negative exponential distribution, negexp(λ), where the rate parameter λ gives the mean time between passenger arrivals. Similarly, we might capture the arrival of passengers carrying prohibited items with a different negexp parameter.

Once the model has been connected to its environment in this way, it can be executed, explored, and validated using discrete-event and Monte Carlo-style simulation methods [12, 16].

It is important to emphasize that our approach to systems modelling lies squarely in the tradition of classical applied mathematics: models are built at levels of abstraction appropriate to the questions of interest; irrelevant detail is avoided. In particular, our approach is *not* that of formal specification in the tradition of software engineering.

Finally, we emphasize that the details of the system model sketched here, and its experimental space, are beyond the scope of this paper.

4.2. **System Models and Utility.** In our framework, associated with a system model of a security architecture will be a utility function with definiens of the form

$$w_1 f_1(C - \overline{C}) + w_2 f_2(I - \overline{I}) + w_3 f_3(A - \overline{A}) + w_4 f_4(K - \overline{K}),$$

as discussed in Section 4. Of course, the security concepts of interests may not be exactly C, I, A, and K, but this will be the essential form and character.

As we have explained, in Section 2, in a highly complex situation, such as a security architecture, it will typically not be possible to formulate system equations (in terms of the functions s_1 and s_2 explained in Section 2) in the way that is usually possible in, for example, macroeconomic modelling. Typically, though, the key control variables, such as system interconnectivity or investment in various aspects (people, process, and technology) of security operations, will be identifiable and, associated with each of C, I, A, and K will be some resource components of the model that are calculated and animated by the execution of the model. The experimental space of interest is then explored by varying the values of the control variables and observing the dynamics of the utility function, with a typical objective being to seek (possibly local) maxima for the dependencies, priorities, and preferences of the Actors of interest. See, for example, [5] for a concrete example of the relationship between a system model and utility.

In the case of our airport security example, as considered from the perspective of the security manager, the primary trade-offs are between aircraft integrity (very high weighting, highly symmetric preference favouring over-caution), availability of access to the aircraft (moderately weighting, delays acceptable), and the cost of provisioning the security process with equipment and staff (fixed periodically, probably with contingency funding available to respond to emergencies).

A detailed account of the experimental methodology is beyond the scope of this introductory paper. We remark, however, that the elicitation of the knowledge and preferences of the managers that is required in order to populate the parameters required for the model is a delicate matter. Indeed, we emphasize that the empirical studies (with all their well-documented attendant difficulties [20, 26, 27, 30] are beyond the scope of the present paper. For now, we require a plausible way to proceed to test the feasibility and value of the overall framing and modelling methods in the decision process, deferring consideration of more rigorous preference-elicitation methodologies to another occasion

4.3. **Reasoning about System Models.** Process calculi such as SCCS, CCS, and the pi-calculus come along with associated modal logics [19, 32, 48]. Similarly, the calculus (L)SCRP has an associated modal logic, MBI [13, 10]. The basic idea — deriving from Hennessy-Milner logic [19, 48] — is to work with a logical judgement of the form

$$L, R, E \models \phi,$$

read as 'relative to the available resources R at location L, the process E has property ϕ'. Building on the ideas of bunched logic [36, 42, 41] and its application to Separation Logic [43, 25], MBI has, in addition to the usual *additive* connectives — $\top$, $\wedge$, $\rightarrow$, $\bot$, $\vee$, $\neg$ — a *multiplicative* conjunction, $*$, and a multiplicative implication.[6] Similarly, in addition to the usual additive quantifiers and modalities, MBI has multiplicative quantifiers and modalities.

The relationship between truth and action is captured by the clauses of the satisfaction relation for the (additive) modalities, given essentially as (recall that $(L', R') = \mu(a, L, R)$)

$$L, R, E \models \langle a \rangle \phi \quad \text{iff} \quad \begin{array}{l} \text{there exists } E' \text{ such that } L, R, E \xrightarrow{a} L', R', E' \text{ and} \\ R', E' \models \phi \end{array}$$

which expresses what is possibly true after an action a has occurred, and

$$L, R, E \models [a] \phi \quad \text{iff} \quad \begin{array}{l} \text{for all } E' \text{ such that } L, R, E \xrightarrow{a} L', R', E', \\ L', R', E' \models \phi. \end{array}$$

which expresses the necessarily true after an action a has occurred.

In this setting, however, the multiplicative conjunction, $*$, that is available in bunched logic provides a characterization of this judgement that is rather finer that which is available in basic Hennessy-Milner logic. Specifically, we obtain the following characterization of the concurrent structure of models ($\sim$ denotes bisimulation of processes):

$$L, R, E \models \phi_1 * \phi_2 \quad \text{iff} \quad \begin{array}{l} \text{there are } R_1 \text{ and } R_2 \text{ such that} \\ L_1 \circ L_2 = L,\ R_1 \circ R_2 = R, \text{ and there are } E_1 \text{ and } E_2 \\ \text{such that } E_1 \times E_2 \sim E, \text{ and} \\ L_1, R_1, E_1 \models \phi_1 \text{ and } L_2, R_2, E_2 \models \phi_2. \end{array}$$

[6] Both classical and intuitionistic versions of the logic are possible.

Here the truth condition for the multiplicative conjunction requires the combination of resources and locations from the truth conditions for its component formulae.

These characterizations provide tools for reasoning about security properties such as joint-access (control), where two agents must both provide some resource for access to be granted, and authorization by delegation, where Hiding is used to establish a private channel; see [14] for the details of these and other examples.

To see how this works, consider once again part of the airport example, as illustrated in Figure 5. As we have seen, the purpose of the airport's security architecture is to ensure that the aircraft has certain integrity properties. One such property is that no prohibited items be on board. Another is that all baggage in the hold be associated with a passenger in the cabin. We can use the logic introduce above to express such properties.

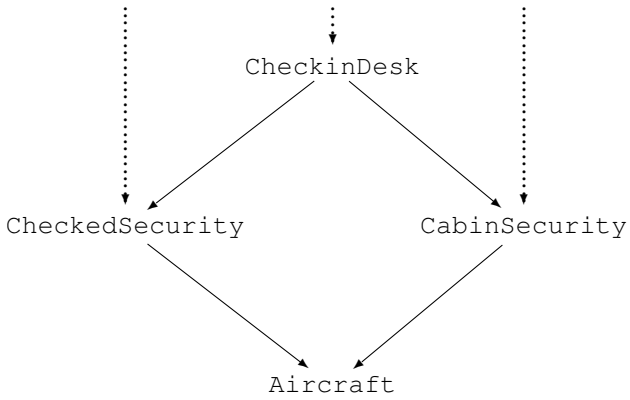


FIGURE 5. Locations of Airport Integrity Checks

Let the overall state of the model of that fragment of the system model that is concerned with checking the integrity of bags and passengers be denoted by $L_{\text{Overall}}, R_{\text{Overall}}, E_{\text{Overall}}$. The following judgement expresses the property that the overall security architecture supports the integrity property of the aircraft that it is not possible for both a checked-bag to be loaded into the hold of the aircraft and for its associated passenger not to board the aircraft; that is, all bags in the hold are associated with passengers in the cabin:

$$L_{\text{Overall}}, R_{\text{Overall}}, E_{\text{Overall}} \models \neg(\langle\text{checked-bag-loaded}\rangle\top * \neg\langle\text{passenger-boarded}\rangle\top)$$

This judgement will hold just in case it is *not* the case that *both*

$$L_{\text{Checked}}, R_{\text{Checked}}, E_{\text{Checked}} \models \langle\text{checked-bag-loaded}\rangle\top$$

and

$$L_{\text{Cabin}}, R_{\text{Cabin}}, E_{\text{Cabin}} \models \neg\langle\text{passenger-boarded}\rangle\top$$

hold, where the overall 'security checking' location divides into that for checked baggage and that for cabin baggage and passengers, where the available security-checking resources — that is, the Security Components and security-staff Actors in the Security Architecture — are divided between these two locations, and where the overall checking process is the concurrent product of the processes for the hold and the cabin.

We could have enriched this expression further by including explicitly in the logical formula an assertion 'Match(Passenger,LuggageLabel,HoldBaggage)', thereby adding another component to the correctness conditions above. This additional component would require that we also take account of the resources necessary — here, the member of the airport's ground staff who checks the aircraft's manifest immediately prior to the closing of the doors — to make the check.

As currently formulated, MBI does not concern itself with stochastic properties of models. In contrast, Probabilistic Computation Tree Logic (see, for example, [45]) is intimately related to Markov chains, but lacks the structural analysis afforded by MBI. Consideration of adding stochastic properties to MBI represents challenging further work. A degree of model checking is available for MBI [11]. In contrast with

PRISM [29], for example, stochastic issues are not considered, the focus being structural decompositions of models via the multiplicative connectives, such as $*$.

5. An OWL Description of the Ontology

So far we have laid out a new system for structuring security systems. The methods outlined have, however, little connection to existing sources of data. One of the aims when formulating this approach was that it should not rely on collecting new information, but rather make use of existing knowledge. To achieve this aim, there needs to be a template into which this knowledge can be inserted. Additionally, we understand that the system as outlined so far is relatively opaque to a pragmatic security manager.

The three key parts of this methodology are the Framework Layer, the Instantiation Layer, and the system model. Each informs the structure and population of the others. Structurally, the Instantiation Layer is organized in such a fashion that it facilitates the exporting of data to the system model. As it is this layer that details the Security Components used to achieve the aims of the Security Objects, it is also the natural place in which to draw in existing system-specific knowledge. To standardize this stage of the process, and provide a syntactic interface to security managers, we advocate using an existing and widely used ontology language to create a template for the structure of the Instantiation Layer. We choose to use OWL here as an example, as it is generic, widely available, and familiar. We describe here how an OWL [37] template, represent using Protégé [40], describes a particular ontology for airport security. It is also just a small abstraction to see that we are illustrating a general form of such ontologies. A similarly, process-oriented (i.e., actors and actions) view of security ontology is adopted in [7].

Such OWL templates retain the structure of the Framework and Instantiation Layers with classes for each rôle and category in the hierarchy. The notions of process, resource, location, and environment are also fully retained. By utilizing this structure, a fully populated OWL template of the Instantiation Layer functions as a map for the implementation stage of the system modelling process. Comparison between the system model and the OWL template will then act as a completeness/redundancy test.

Figure 6 illustrates several key areas of the OWL template. The upper left panel shows the structure of the OWL classes used to represent the hierarchy. This reflects the structure of the Instantiation Layer and holds all of the key concepts of the hierarchy. The lower left frame shows a sample of the relationships used to link entities in the ontology. These allow the creation of constructs within the ontology that are analogous to the Security Objects and Security Components of the Instantiation Layer. To the right of the figure are the members of the 'hierarchyCategory' class. These are an example of the individual entities that can be linked by relationships.

Figure 7 illustrates in more detail one of the individual Security Components of the ontology. This component 'checkHandLuggage2DSC' is a member of the 'checkHandLuggage' Security Object, relates to Rôle 2 in the hierarchy (securityStaff), is a dependency, and operates at the security checkpoint location. This information is contained in the label of the Security Component and also in its associated properties. These can be seen in the right-hand panel of this figure and are made explicit through the use of the previously defined relationships. Additional information that is relevant to the Security Component, such as its objectives, may be listed here.

The creation of a full OWL representation of the target security system (beyond our present scope) is a type of task with which security managers or their staff are likely to be familiar. By structuring it in accordance with this template, the information becomes available for export to the system model.

Related work by Parkin, van Moorsel, and Coles [39] has considered how an information security ontology can be used to understand the implications of security policies for human behaviour in the context of compliance with standards such as BS ISO/IEC 27001/2 [8, 9]. It would be useful to extend our approach in this dimension too.

6. Conclusions and Directions

The primary aim of this paper has been to demonstrate that by separating the declarative and operational components of a security architecture it is possible to reason systematically and effectively about trade-offs between CIA-like security objectives. This separation can be effectively brought about by restructuring system information into two layers, the Framework Layer and the Instantiation layer. This structure also

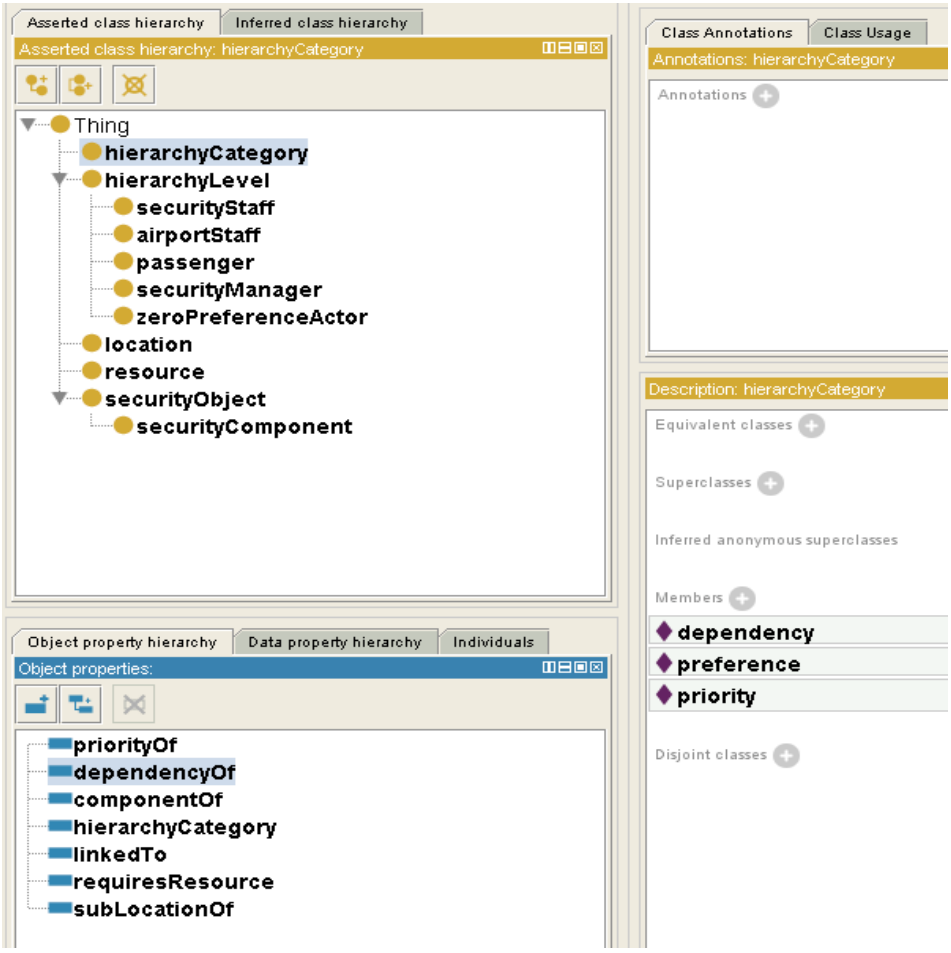


FIGURE 6. Structural Components of the OWL Template

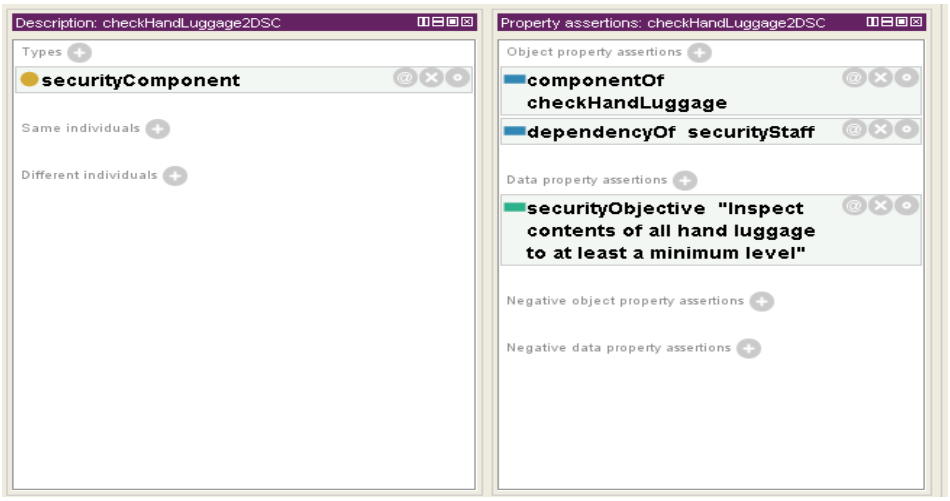


FIGURE 7. A Detail of the Security Component 'checkHandLuggage2DSC' in OWL

facilitates export of such data to a system model. Exploration of the system via this model allows utility calculations to be made and thus supports more effective decision making in the security context.

The ultimate aim of this research is to integrate this approach into a more seamless tool to replace the somewhat handcrafted stages that currently represent the state of the art. Our intent is to use the Gnosis modelling language to develop key components of this tool and explore more fully the impacts and benefits of creating system models of this kind. The creation of such models will allow further reasoning and experimentation to take place. The solicitation of knowledge and preferences to populate the hierarchy is a challenge that faces all areas of experimental economics and, although work is progressing in this area [5], it will still prove to be one of the more difficult aspects of utilizing this methodology. We will also further explore the use of tools such as OWL to provide bases for interfaces between our tools and the managers with whom they must eventually be deployed.

There are certain key areas that we would also like to expand on in future research. In particular, the rôle of Actors and the effects of interactions between multiple hierarchies. These two aspects are intrinsically linked as Actors are the conduit through which hierarchies exchange information. Exploring the rôle of Actors more thoroughly will also allow us to make improvements in user representation, in particular to account for users' effort and their compliance budget [4].

7. Acknowledgements

We are grateful to many colleagues from the disciplines of the computing sciences, economics, and information security — including Christos Ioannidis, Simon Parkin, Martin Sadler, Simon Shiu, Joe Swierzbinski, Aad van Moorsel, and Julian Williams — for their support of this work and their willingness to discuss it with us. The Systems Security Lab at HP Labs, Bristol has provided an ideal environment within which to formulate these ideas.

References

[1] R. Anderson. Why information security is hard: An economic perspective. In *Proc. 17th Annual Computer Security Applications Conference*, 2001.

[2] N. Ashford, H.P.M. Stanton, and C.A. Moore. *Airport Operations*. McGraw-Hill, 1997.

[3] A. Beautement, R. Coles, J. Griffin, C. Ioannidis, B. Monahan, D. Pym, A. Sasse, and M. Wonham. Modelling the Human and Technological Costs and Benefits of USB Memory Stick Security. In M. Eric Johnson, editor, *Managing Information Risk and the Economics of Security*, pages 141–163. Springer, 2008.

[4] A. Beautement, M.A. Sasse, and M. Wonham. The compliance budget: managing security behaviour in organisations. In *Proceedings of the 2008 Workshop on New Security Paradigms*, pages 47–58. ACM Digital Library, 2009.

[5] Y. Beres, D. Pym, and S. Shiu. Decision support for systems security investment. Manuscript, HP Labs, 2009.

[6] G.M. Birtwistle. *Discrete Event Modelling on SIMULA*. Springer-Verlag, 1987.

[7] F.A. Braz, E.B. Fernandez, and M. van Hilst. Eliciting security requirements through misuse activities. In *Proc. 19th. International Conference on Database and Expert System Applications*. IEEE Computer Society Press, 2008.

[8] British Standards Institution. BS ISO/IEC 27001:2005 — Information Technology — Security Techniques — Information Security Management Systems — Requirements. 2005.

[9] British Standards Institution. BS ISO/IEC 27001:2005 — Information Technology — Security Techniques — Information Security Management Systems — Requirements. 2005.

[10] M. Collinson, B. Monahan, and D. Pym. A logical and computational theory of located resource. *Journal of Logic and Computation*, 19(6):1207–1244, 2009. doi:10.1093/logcom/exp021.

[11] M. Collinson, B. Monahan, and D. Pym. A discipline of mathematical systems modelling. To appear: College Publications, London, 2010.

[12] M. Collinson, B. Monahan, and D. Pym. Semantics for structured systems modelling and simulation. In *Proc. Simutools 2010*. ICST: ACM Digital Library and EU Digital Library, 2010. ISBN: 78-963-9799-87-5.

[13] M. Collinson and D. Pym. Algebra and logic for resource-based systems modelling. *Mathematical Structures in Computer Science*, 19:959–1027, 2009. doi:10.1017/S0960129509990077.

[14] M. Collinson and D. Pym. Algebra and logic for access control. *Formal Aspects of Computing*, 2010. To appear. Erratum also to appear. Preprint (incorporating erratum): `http://www.hpl.hp.com/techreports/2008/HPL-2008-75R1.html`.

[15] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proc. of the Seventh Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in LNCS, pages 352–368. Springer-Verlag, 1994.

[16] Gnosis. `http://www.hpl.hp.com/research/systems_security/gnosis.html`.

[17] L.A. Gordon and M.P. Loeb. The Economics of Information Security Investment. *ACM Transactions on Information and Systems Security*, 5(4):438–457, 2002.

[18] L.A. Gordon and M.P. Loeb. *Managing Cybersecurity Resources: A Cost-Benefit Analysis*. McGraw Hill, 2006.

[19] M. Hennessy and G. Plotkin. On observing nondeterminism and concurrency. In *Proceedings of the 7th ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, 1980.

[20] J.C. Hersey, H.C. Kunreuther, and P.J. Shoemaker. Sources of bias in assessment procedures for utility functions. *Management Science*, 28:936–953, 1982.

[21] J. Hillston. Compositional Markovian modelling using a process algebra. In W. Stewart, editor, *Proceedings of the Second International Workshop on Numerical Solution of Markov Chains: Computations with Markov Chains*. Kluwer Academic Press, 1995.

[22] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[23] C. Ioannidis, D. Pym, and J. Williams. Information security trade-offs and optimal patching policies. Manuscript, 2009.

[24] C. Ioannidis, D. Pym, and J. Williams. Investments and trade-offs in the economics of information security. In Roger Dingledine and Philippe Golle, editors, *Proceedings of Financial Cryptography and Data Security '09*, volume 5628 of *LNCS*, pages 148–166. Springer, 2009. Preprint available at `http://www.cs.bath.ac.uk/~pym/IoannidisPymWilliams-FC09.pdf`.

[25] S.S. Ishtiaq and P. O'Hearn. **BI** as an assertion language for mutable data structures. In *28th ACM-SIGPLAN Symposium on Principles of Programming Languages, London*, pages 14–26. Association for Computing Machinery, 2001.

[26] J.Y. Jaffrey. Some experimental findings on decision-making under risk and their implications. *European Journal of Operational Research*, 38:301–306, 1989.

[27] A. Jimenéz, S. Ros-Insua, and A. Mateos. A decision support system for multi-attribute utility evaluation based on imprecise assignments. *Decision Support Systems*, 36:65–79, 2003.

[28] R.L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, 1976.

[29] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic Model Checking for Performance and Reliability Analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[30] M. McCord and R. de Neufville. Lottery equivalents: reduction of the certainty effect problem in utility assessment. *Management Science*, 32:56–61, 1986.

[31] R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.

[32] R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.

[33] R. Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.

[34] R. De Neufville and A.R. Odoni. *Airport systems: planning, design, and management*. McGraw-Hill Professional, 2003.

[35] Peter Neumann. *Computer-Related Risks*. Addison-Wesley, 1995.

[36] P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bull. of Symb. Logic*, 5(2):215–244, 1999.

[37] OWL. `http://en.wikipedia.org/wiki/Web_Ontology_Language`.

[38] Donn Parker. *Fighting Computer Crime: A New Framework for Protecting Information*. Wiley, 1992.

[39] S.E. Parkin, A. van Moorsel, and R. Coles. An information security ontology incorporating human-behavioural implications. In *Proceedings of the 2nd International Conference on Security of Information and Networks*, pages 46–55. ACM Digital Library, 2009.

[40] Proíegé. `http://protege.stanford.edu/`.

[41] D.J. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002. Errata and Remarks maintained at: `http://www.cs.bath.ac.uk/~pym/BI-monograph-errata.pdf`.

[42] D.J. Pym, P.W. O'Hearn, and H. Yang. Possible worlds and resources: The semantics of BI. *Theoretical Computer Science*, 315(1):257–305, 2004. Erratum: p. 285, l. -12: ", for some $P', Q \equiv P; P'$ " should be "$P \vdash Q$".

[43] J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. LICS '02*, pages 55–74. IEEE Computer Society Press, 2002.

[44] Francisco J. Ruge-Murcia. Inflation targeting under asymmetric preferences. *Journal of Money, Credit, and Banking*, 35(5), 2003.

[45] J.J.M.M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[46] Norman E.L. Shanks and Alexandre L.W. Bradley. *Handbook of Checked Baggage Screening: Advanced Airport Security Operation*. WileyBlackwell, 2004.

[47] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-theortic, Logical Foundations*. Cambridge University Press, 2009.

[48] Colin Stirling. *Modal and Temporal Properties of Processes*. Springer Verlag, 2001.

[49] Transport Security Administration. Recommended Security Guidelines for Airport Planning, Design, and Construction. `http://www.tsa.gov/assets/pdf/airport_security_design_guidelines.pdf`, 2006.

[50] Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, 5:297–323, 1992. doi:10.1007/BF00122574.

[51] H. Varian. A bayesian approach to real estate management. In S.E. Feinberg and A. Zellner, editors, *Studies in Bayesian Economics in Honour of L.J. Savage*, pages 195–208. North Holland, 1974.

[52] A.T. Wells and S. Young. *Airport Planning and Management*. McGraw-Hill Professional, 2004.

[53] A. Zellner. Bayesian prediction and estimation using asymmetric loss functions. *Journal of the American Statistical Association*, 81:446–451, 1986.

UCL AND HP LABS, BRISTOL
E-mail address: a.beautement@cs.ucl.ac.uk

HP LABS, BRISTOL AND UNIVERSITY OF BATH
E-mail address: david.pym@me.com