Modelling and Simulating Systems Security Policy

Tristan Caulfield
University College London
Department of Computer Science
Gower Street, London WC1E 6BT
t.caulfield@ucl.ac.uk

David Pym
University College London
Department of Computer Science
Gower Street, London WC1E 6BT
d.pym@ucl.ac.uk

ABSTRACT

Security managers face the challenge of designing security policies that deliver the objectives required by their organizations. We explain how a rigorous modelling framework and methodology—grounded in semantically justified mathematical systems modelling, the economics of decision-making, and simulation—can be used to explore the operational consequences of their design choices and help security managers to make better decisions. The methodology is based on constructing executable system models that illustrate the effects of different policy choices. Models are compositional, allowing complex systems to be expressed as combinations of smaller, complete models. They capture the logical and physical structure of systems, the choices and behaviour of agents within the system, and the security managers' preferences about outcomes. Utility theory is used to describe the extent to which security managers' policies deliver their security objectives. Models are parametrized based on data obtained from observations of real-world systems that correspond closely to the examples described.

Categories and Subject Descriptors

I.6 [**Simulation and modelling**]: General; F.3 [**Logics and meanings of programs**]: Miscellaneous; H.1 [**Models and principles**]: Systems and information theory; K.6 [**Security and Protection**]; K.6.0 [**General**]: Economics; K.6.1 [**Project and People Management**]: Systems analysis and design

General Terms

design, economics, experimentation, languages, management, modelling, security, theory

Keywords

composition, decision, location, logic, modelling, policy, process, resource, security, semantics, simulation

1. INTRODUCTION

A system's security policies—and, indeed, a system's management policies more generally—must be formulated so as to deliver the declarative objectives of the system's managers in the context of the system's architecture and operational objectives.

When formulating, implementing, and monitoring security policies, managers are often faced with deciding between achieving varying degrees of security and achieving varying degrees of operational effectiveness. That is, more effective security controls will often more severely impede operations. A key challenge for system managers is to engineer trade-offs between concerns such as these that is acceptable to their organizations.

It is an increasingly commonplace observation that mathematical modelling—of the system, of its operational and security policies, and of the behaviour of its users—can provide useful decision-support tools [2, 3, 4, 5]. But the systems about which these decisions must be made are usually complex socio-technical systems of systems for which analytic solutions to policy-system co-design problems are rarely available. Instead, simulation-based models [18] supporting experimental mathematical modelling methods offer a useful approach [2, 4, 8, 5].

In this paper, we argue for and demonstrate the value of developing a systems and policy modelling framework with key structural, economic, and experimental features:

- Explicit structural models of the structure of systems. That is, following the classical approach to understanding distributed systems [11], we consider models based on concepts of location, resource, process, and environment. Locations are the places within a system at which its resources reside. Resources are the components of a system that are manipulated (e.g., consumed, created, moved, read, etc.) by the processes; we assume that resource elements of the same type can be combined and compared. Processes are the concept that describes the dynamics of the system, manipulating located resources in order to deliver services. Environment is the context within which the system exists, to and from which services may be delivered and received by the system;
- Simulation-based techniques for experimental mathematical investigations of the consequences of different system–policy co-design decisions for their benefit-cost analysis;
- Use of economic models of agents' decision-making and managers' valuations of policies [13, 14]. Production functions [15] are used to characterize the values of the different choices available to agents during the execution of a mode. Utility functions [25, 20] characterize the overall value of a (security) policy to a system's manager. We use multi-attribute utility functions to capture the trade-offs between different

attributes, such as productivity and security, that a system's managers make when designing and implementing policies;
- Compositionality at all levels. Structure and experiments: the underlying mathematical treatments of process, resource, and location are naturally compositional—models can very simply be joined together using a simple concept of interface, and experiments can then run through composed models. Agents' decisions: the expected value of an agent's decision at any point in a model is expressed in such way that the expected value of composite decisions, as maybe derived in composed models, is calculated additively. Managers' utility: the form of the objective function describing the system's managers' expected utility is such that the expected utility of a composed model is, essentially, the sum of the expected utilities of the component models.

Thus our approach represents a compositional synthesis of semantically-justified systems modelling, information security (and policy) economics, and simulation.

Although our approach is unusual in its use of tools from logic and semantics, the modelling methodology is simply that of classical applied mathematics and simulation, as illustrated in Figure 1. Note

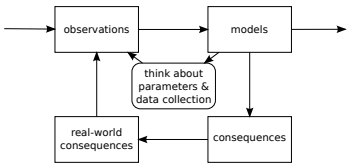


Figure 1: The Classical Modelling Cycle

that we emphasize the importance of integrating the exploration of the parameter space with the construction of models.

In Section 2, we explain the semantic structures that provide the basis for our approach to systems modelling. It is organized according to the distributed systems-inspired view described above. Then, in Section 3, we described the (simplified) class of models that we implement, and explain how this implementation is achieved in the julia language [19]. Then, in Section 3.4.1, we explain how agents and their decision-making are captured in our implemented models, and also consider how system managers' utility functions can be used to assess the value of policies in both stand-alone and composite models.

To illustrate our methods, we base our examples on three models. First, we look at access control to a building and consider the problem of tailgating at the entrance barrier. Second, we consider how employees within an organization decide how to share confidential documents. Third, we consider loss of devices containing confidential information by an organization's employees. Having established these three models, we then illustrate a key feature of our approach; that is, composition. We show how these models, and their experimental simulations, can be composed to yield a model of information loss from an organization that has access control to its building and a document-sharing policy. In Section 5, we explore experimentally some key properties of these three models.

We provide, in Section 6, a summary of the key features of our work and give a brief discussion of some research directions, primarily concerned with model checking via a modal logic that is naturally associated with our modelling approach.

Throughout this short paper we seek to maintain an informal, conceptual style of presentation. Much of the required underlying mathematics, when presented in full detail, is both quite abstract and quite complex to articulate. Here our primary concern is to articulate a methodology—incorporating structural, economic, and computational aspects—and its use as tool to support policy decision-making. Accordingly, we aim give just enough of the mathematical and, indeed, implementation detail to support an explanation of the story. Many readers will have sufficient familiarity with logic, process algebra, and stochastic and experimental methods to appreciate what would be required for full detail to be provided. The underlying mathematical set-up as sketched in Section 2.

2. A SEMANTIC BASIS

We give a brief summary of the process-theoretic and logical theory that underpins the semantics-based approach to modelling and simulating systems security policy that we have described above. This work is developed in detail in [7, 8, 9].

2.1 Processes and Resources

Milner's synchronous calculus of communicating systems, SCCS [21], perhaps the most basic of process calculi, provides our first point of departure. This, together with the theory of resources provided by the semantics of bunched logic [22, 9], provides the basis for a calculus of resources and processes. The key components for our purposes are the following:

- A monoid of actions, Act, with a composition ab of elements a and b and unit 1;
- The following grammar of process terms, E, where $a \in \mathsf{Act}$ and X denotes a process variable:

$$E ::= a : E \mid \sum_{i \in I} E_i \mid E \times E \mid X \mid \mathrm{fix}_i X.E \mid (\nu R)E$$

Most of the cases here, such as action prefix, sum, concurrent product, and recursion (in the fix_i case, X and E are tuples, and we take the ith component of the tuple), will be quite familiar to theorists. The term $(\nu R)E$, in which R denotes a resource, is called hiding, and is available because we integrate the notions of resource and process; it generalizes restriction.

Our notion of resource—which encompasses natural examples such as space, memory, and money—is based mathematically on ordered, partial, commutative monoids (e.g., the non-negative integers with the monoid of addition with unit zero and order less-than-or-equals), and captures the basic properties of resources discussed briefly in Section 1: each type of resource is based on a basic set of resource elements, which can be combined (and the combination has a unit) and compared. Formally, we take preordered, 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 preorder on $\mathbf{R}$.

The basic idea is that resources, R, and processes, E, co-evolve, $R, E \xrightarrow{a} R', E'$, according to the specification of a partial 'modification function', $\mu : (a, R) \mapsto R'$, that determines how an action a evolves E to E' and R to R'.

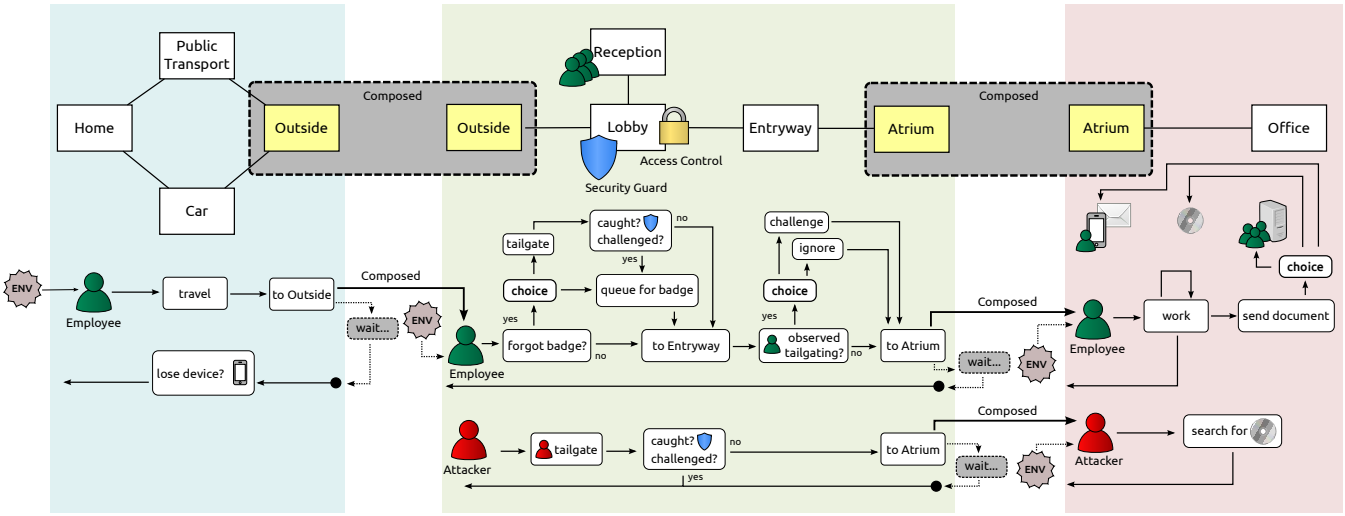


Figure 2: The three example models, composed

The base case of the operational semantics is given by action prefix:

$$\frac{}{R, a : E \xrightarrow{a} R', E} \qquad \mu(a, R) = R'.$$

Concurrent composition, $\times$, exploits the monoid composition, $\circ$, on resources,

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

Modification functions are required to satisfy some basic coherence conditions: $\mu(1, R) = R$, where 1 is the unit action, and, if $\mu(a, R) \circ \mu(b, S)$ and $R \circ S$ are defined, then $\mu(ab, R \circ S) \simeq \mu(a, R) \circ \mu(b, S)$, where $\simeq$ is Kleene equality, for all actions a and b and all resources R and S. In certain circumstances, additional equalities may be required [7, 9].

Sums and recursion are formulated in familiar ways:

$$\frac{R, E_i \xrightarrow{a} R', E'}{R, \sum_{i \in I} E_i \xrightarrow{a} R', E'} \quad \text{and} \quad \frac{R, E_i[E/X] \xrightarrow{a} R', E'}{R, \mathrm{fix}_i X.E \xrightarrow{a} R', E'},$$

where I is an indexing set and E_i is the ith component of the tuple E of processes. Other combinators are handle similarly.

2.2 Location

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 [9], including a collection of atomic locations—the basic places—which generate a structure of locations, and a notion of (directed) connection between locations—describing the topology of the system.

We can also consider a notion of sublocation (which respects connections) and 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. The notions of sublocation and substitution are intimately related, the former being a prerequisite for the latter. We do not develop or support substitution in the current version of our implemented models.

We remark briefly that treating location as a first-class citizen in this way does not lead to a process calculus with operational behaviour that is more expressive in absolute terms. It does, however, lead to greater pragmatic expressiveness and simplifies the construction of models of a wide range of systems.

The resulting calculus has a transition system given by a judgement of the form $L, R, E \xrightarrow{a} L', R', E'$, where a is an action (in the usual process sense), L, L' are location environments, R, R' are resource environments and E, E' are processes used to control the evolution. As sketched above, we define a modification function, μ, which, for each action a, location L, and resource R, determines the evolved location L' and resource R': that is, $\mu : (a, L, R) \mapsto (L', R')$, with corresponding versions of the coherence conditions. In a state L, R, E, the component L carries the relevant information about the topology of the model and R carries the relevant information about the distribution of the resources around the model's topology.

The operational semantics extends to locations very naturally. The following is the rule for action prefix:

$$\frac{}{L, R, E \xrightarrow{a} L', R', E'} \quad (L', R') = \mu(a, L, R)$$

The details of the operational semantics with location and its implementation may be found in [9].

2.3 Environment

The systems that we model do not exist in isolation, but rather interact with an environment that we choose not to model in detail. Rather, we represent an environment by capturing the incidence on the model of events from the environment stochastically. For example, the arrival rate of agents at entrance that marks the outer boundary of a model may be captured using a negative exponential distribution [24, 18]. Similarly, a model may capture outputs from the system that is modelled to the surrounding environment.

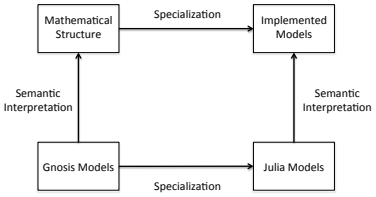
Within a complex model there may be components that we do not need to model in detail, and we can capture the interaction between such components and the rest of the model probabilistically.

2.4 Gnosis: A Proof-of-concept Tool

The semantic basis for system modelling described above has been implemented in a proof-of-concept modelling tool called Gnosis [8, 9] that closely captures the semantic structure of processes, resources, and locations. In [8, 9], a formal semantics of Gnosis is given in the semantic structures described above. Since the stochastic aspects of models are not captured directly in these structures, Gnosis's scheduler is given a denotational definition within them.

In the implementation described in this paper, we employ not Gnosis, but rather a new framework written julia [19]. julia is well-adapted to describing the simplified implemented models that we explain in detail in the next section and is supported by well-engineered programming and graphics environments. Our use of julia is explained in Section 3.4.

The relationship between the abstract semantic structures for processes, resources, and locations, the implemented models, Gnosis models, and julia models can summarized by the following diagram: Each of the arrows can be given a precise mathematical definition; for example, the interpretation of Gnosis models in the semantic structures is spelled out in [8, 9], and the interpretation of julia models in our implemented models is similar.



3. IMPLEMENTED MODELS

The mathematical structure of models as described above provides the basis for the class of models that we implement. We employ well-motivated simplifcations of the general semantic set-up.

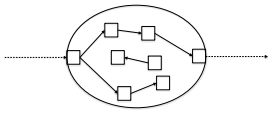


Figure 3: The basic structure of implemented models

Models in this methodology are designed to be composed with other models (Figures 3 and 4). Composition allows us to join two or more models together and see what effects their interaction has. When models are composed there are interactions at the location, process, and resource levels, and the role of their intended environments is critical. Processes transition and resources are moved between models at locations shared between the models.

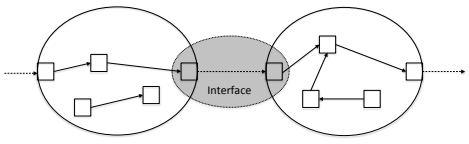


Figure 4: The basic structure of composed models

To enable composition, models contain interfaces, which define the locations where models fit together and which actions, defined at appropriate locations within the interface, are party to the composition. Actions in the interface will nevertheless be able to execute only if the resources they require are available.

3.1 Definitions of Implemented Models

The locations and resources of a model are represented using a location graph, $\mathcal{G}(\mathcal{V}[\mathcal{R}], \mathcal{E})$, with a set of vertices, $\mathcal{V}$, representing the locations of the model, and a set of directed edges, $\mathcal{E}$, giving the connections between the locations. Vertices are labelled with resources $\mathcal{R}$.

As explained above, actions evolve the processes and resources of a model. However, rather than thinking of actions evolving processes, it is convenient to think of a process as a trace of actions—the history of actions that have evolved a process during the execution of the model. All of the actions in a model are contained in a set, $\mathcal{A}$, and process traces are comprised of these.

The environment a model sits inside causes actions within the model to be executed, at a particular location. A model contains a set of located actions, $\mathcal{L}$, and a located action, $l \in \mathcal{L}$, is given by an ordered pair $l = (a \in \mathcal{A}, v \in \mathcal{V})$. The environment associates these located actions with probability distributions: $Env : \mathcal{L} \to ProbDist$. During the execution of the model, the located actions are brought into existence by sampling from these distributions.

Models also need to have interfaces in order to support composition. An interface $I \in \mathcal{I}$ on a model is a tuple (In, Out, L) of sets of input and output vertices, where $In \subseteq \mathcal{V}$ and $Out \subseteq \mathcal{V}$, and a set of located actions $L \subseteq \mathcal{L}$.

The sets of input vertices and output vertices in interfaces must be disjoint; that is

$$\bigcap_{i \in \mathcal{I}} In_i \in In = \emptyset \quad \text{and} \quad \bigcap_{i \in \mathcal{I}} Out_i \in Out = \emptyset.$$

DEFINITION 1. *A model $M = (\mathcal{G}(\mathcal{V}[\mathcal{R}], \mathcal{E}), \mathcal{A}, \mathcal{P}, \mathcal{L}, \mathcal{I})$ is a tuple that consists of a location graph $\mathcal{G}$, a set of actions $\mathcal{A}$, a set of processes $\mathcal{P}$, a set of located actions $\mathcal{L}$, and a set of interfaces $\mathcal{I}$.*

3.2 Composition of Implemented Models

Two models, M_1 and M_2 are composed with specific interfaces $I_{1,1}, \ldots, I_{1,j}, \ldots, I_{1,n} \in \mathcal{I}_1$ and $I_{2,1}, \ldots, I_{2,k}, \ldots, I_{2,m} \in \mathcal{I}_2$ using the composition operator, $M_1{}_{I_{1,j}}|_{I_{2,k}} M_2$, which is defined using an operation, $\oplus$, on each of the elements of a model.

First, we define the $\oplus$ operator for vertices and edges,

$$\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathcal{V}_1 \cup \mathcal{V}_2$$

and, for each $v \in \mathcal{V}_1 \oplus \mathcal{V}_2$,

$$v[\mathcal{R}_1 \oplus \mathcal{R}_2] = \begin{cases} v[\mathcal{R}_1] & \text{if } v \in \mathcal{V}_1 \wedge v \notin \mathcal{V}_2 \\ v[\mathcal{R}_2] & \text{if } v \in \mathcal{V}_2 \wedge v \notin \mathcal{V}_1 \\ v[\mathcal{R}_1 \cup \mathcal{R}_2] & \text{otherwise.} \end{cases}.$$

Composition of edges, actions, and proceeses are straightforward: $\mathcal{E}_1 \oplus \mathcal{E}_2 = \mathcal{E}_1 \cup \mathcal{E}_2$, $\mathcal{A}_1 \oplus \mathcal{A}_2 = \mathcal{A}_1 \cup \mathcal{A}_2$, and $\mathcal{P}_1 \oplus \mathcal{P}_2 = \mathcal{P}_1 \cup \mathcal{P}_2$.

To define the $\oplus$ operator for locations and interfaces, we first need to introduce some notation. The interfaces on a model are a set of

tuples; for example, the interfaces of M_1:

$$\mathcal{I}_1 = \{(In_1, Out_1, L_1)_i\}$$

A particular interface from $\mathcal{I}_1$ is referred to as $I_{1,i}$, and the input locations from that interface are referred to as $In_{1,i}$, the outputs as $Out_{1,i}$, and the located actions as $L_{1,i}$.

When models are composed, the located actions in the interface that were executed by the environment in the uncomposed model are now executed as a consequence of the other model instead. As such, the composition of located actions is the union of both sets of located actions, minus those that are in interfaces used in the composition: $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathcal{L}_1 \cup \mathcal{L}_2 \setminus \{L_{1,j}, L_{2,k}\}$.

Interfaces can be used in just one composition, and the input and output locations of the interfaces from the two models must correspond, so their composition is $\mathcal{I}_1 \oplus \mathcal{I}_2 = (\mathcal{I}_1 \cup \mathcal{I}_2) \setminus \{I_{1,j}, I_{2,k}\}$, where we require $\bigcup_{j=1}^{n} In_{I_{1,j}} = \bigcup_{k=1}^{m} Out_{I_{2,k}}$ and $\bigcup_{j=1}^{n} Out_{I_{1,j}} = \bigcup_{k=1}^{m} In_{I_{2,k}}$. Models must be composed completely: any location that is in both of the models must belong to the interfaces used in the composition.

DEFINITION 2. *With the data as established above, the composition of models M_1 and M_2 is given by*

$$M_1 {}_{I_{1,j}}|_{I_{2,k}} M_2 = (\mathcal{G}((\mathcal{V}_1 \oplus \mathcal{V}_2)[\mathcal{R}_1 \oplus \mathcal{R}_2], \mathcal{E}_1 \oplus \mathcal{E}_2), \mathcal{A}_1 \oplus \mathcal{A}_2, \mathcal{P}_1 \oplus \mathcal{P}_2, \mathcal{L}_1 \oplus \mathcal{L}_2, (\mathcal{I}_1 \oplus \mathcal{I}_2)),$$

with the constraint that $\mathcal{V}_1 \cap \mathcal{V}_2 = In_{1,j} \cup In_{2,k}$.

PROPOSITION 1. *$M_1 {}_{I_{1,j}}|_{I_{2,k}} M_2$ is a model.*

PROOF. The composition of each of the components of the models is well-defined:

- $\mathcal{G}_1 \oplus \mathcal{G}_2((\mathcal{V}_1 \oplus \mathcal{V}_2)[\mathcal{R}_1 \oplus \mathcal{R}_2], \mathcal{E}_1 \oplus \mathcal{E}_2)$ is immediately a location graph with vertices $\mathcal{V}_1 \cup \mathcal{V}_2$, resources $\mathcal{R}_1 \oplus \mathcal{R}_2$, and edges $\mathcal{E}_1 \cup \mathcal{E}_2$;
- $\mathcal{A}_1 \oplus \mathcal{A}_2$ and $\mathcal{P}_1 \oplus \mathcal{P}_2$ are trivially the unions of the actions and processes, respectively;
- $\mathcal{L}_1 \oplus \mathcal{L}_2$ is trivially a set of located actions.

The sets of input vertices and output vertices in the composed model must both be disjoint. With the constraints that $\mathcal{V}_1 \cap \mathcal{V}_2 = In_{1,j} \cup In_{2,k}$ and that $\bigcup_{j=1}^{n} In_{I_{1,j}} = \bigcup_{k=1}^{m} Out_{I_{2,k}}$ and $\bigcup_{j=1}^{n} Out_{I_{1,j}} = \bigcup_{k=1}^{m} In_{I_{2,k}}$ then any locations in both of the models are present in the interfaces used for composition and $(\mathcal{I}_1 \cup \mathcal{I}_2) \setminus \{I_{1,j}, I_{2,k}\}$ is a well-defined set of interfaces. □

PROPOSITION 2. *Composition of models is commutative:*

$$M_1 {}_{I_{1,j}}|_{I_{2,k}} M_2 = M_2 {}_{I_{2,k}}|_{I_{1,j}} M_1$$

PROOF. Each component of the model is commutative.

Trivially, $\mathcal{A}_1 \oplus \mathcal{A}_2 = \mathcal{A}_2 \oplus \mathcal{A}_1$, $\mathcal{L}_1 \oplus \mathcal{L}_2 = \mathcal{L}_2 \oplus \mathcal{L}_1$, and $\mathcal{P}_1 \oplus \mathcal{P}_2 = \mathcal{P}_2 \oplus \mathcal{P}_1$.

Next, location graphs: $\mathcal{G}_1 \oplus \mathcal{G}_2((\mathcal{V}_1 \oplus \mathcal{V}_2)[\mathcal{R}_1 \oplus \mathcal{R}_2], \mathcal{E}_1 \oplus \mathcal{E}_2) = \mathcal{G}_2 \oplus \mathcal{G}_1((\mathcal{V}_2 \oplus \mathcal{V}_1)[\mathcal{R}_2 \oplus \mathcal{R}_1], \mathcal{E}_2 \oplus \mathcal{E}_1)$. The edges are straightforward: $\mathcal{E}_1 \oplus \mathcal{E}_2 = \mathcal{E}_2 \oplus \mathcal{E}_1$. Vertices are trivial—$\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathcal{V}_2 \oplus \mathcal{V}_1$—but resources require more care: $(\mathcal{V}_1 \oplus \mathcal{V}_2)[\mathcal{R}_1 \oplus \mathcal{R}_2] = (\mathcal{V}_2 \oplus \mathcal{V}_1)[\mathcal{R}_2 \oplus \mathcal{R}_1]$.

For resources, we must show that any $v \in \mathcal{V}_1 \oplus \mathcal{V}_2$ gets the same assignment of resources from $\mathcal{R}_1 \oplus \mathcal{R}_2$ as $\mathcal{R}_2 \oplus \mathcal{R}_1$.

Let $v \in \mathcal{V}_1 \oplus \mathcal{V}_2$.

$$v[\mathcal{R}_1 \oplus \mathcal{R}_2] = \begin{cases} v[\mathcal{R}_1] & \text{if } v \in \mathcal{V}_1 \wedge v \notin \mathcal{V}_2 \\ v[\mathcal{R}_2] & \text{if } v \in \mathcal{V}_2 \wedge v \notin \mathcal{V}_1 \\ v[\mathcal{R}_1 \cup \mathcal{R}_2] & \text{otherwise.} \end{cases}$$

$$v[\mathcal{R}_2 \oplus \mathcal{R}_1] = \begin{cases} v[\mathcal{R}_2] & \text{if } v \in \mathcal{V}_2 \wedge v \notin \mathcal{V}_1 \\ v[\mathcal{R}_1] & \text{if } v \in \mathcal{V}_1 \wedge v \notin \mathcal{V}_2 \\ v[\mathcal{R}_2 \cup \mathcal{R}_1] & \text{otherwise.} \end{cases}$$

For the remaining case, $v[\mathcal{R}_1 \cup \mathcal{R}_2] = v[\mathcal{R}_2 \cup \mathcal{R}_1]$. In the other cases, then v is either in $\mathcal{V}_1$ or $\mathcal{V}_2$ so, regardless of order, $v[\mathcal{R}_1 \oplus \mathcal{R}_2] = v[\mathcal{R}_2 \oplus \mathcal{R}_1] = v[\mathcal{R}_1]$ if $v \in \mathcal{V}_1$ and $v[\mathcal{R}_2]$ if $v \in \mathcal{V}_2$.

Finally, interfaces: $\mathcal{I}_1 \cup \mathcal{I}_2 = \mathcal{I}_2 \cup \mathcal{I}_1$ and $\{I_{1,j}, I_{2,k}\} = \{I_{2,k}, I_{1,j}\}$, so $\mathcal{I}_1 \oplus \mathcal{I}_2 = \mathcal{I}_2 \oplus \mathcal{I}1$. We know from Proposition 1 that the composition of interfaces is well-defined. □

For any pair of models M_1 and M_2, let $I_{1\to 2}$ be the subset of interfaces in $\mathcal{I}_1$ that compose with M_2.

PROPOSITION 3. *Composition of models is associative:*

$$(M_1 {}_{I_{1\to 2}}|_{I_{2\to 1}} M_2) {}_{I_{1\to 3} \cup I_{2\to 3}}|_{I_{3\to 1} \cup I_{3\to 2}} M_3 = M_1 {}_{I_{1\to 2} \cup I_{1\to 3}}|_{I_{2\to 1} \cup I_{3\to 1}} (M_2 {}_{I_{2\to 3}}|_{I_{3\to 2}} M_3)$$

PROOF. The proof unpacks the definition of composition in a style similar to the proof of Proposition 2. The details are straightforward. □

3.3 Environment

As we have previously explained, in Section 2.3, the environment represents things outside the scope of the model: events and details that have an effect on the model, but are too complicated or of not enough interest to model directly. The modeller chooses a distribution to represent the likelihood of these events occurring, and these distributions are sampled to determine when to execute located actions within the model.

When models are composed, some of these located actions are removed, meaning that the distributions previously chosen for them are not sampled and the actions are no longer a consequence of the environment. Instead, the located actions are initiated by a process in the other model, but they are still, at some level, dependent on the environment because the other model still samples from it.

Consider Figure 5, which shows two processes, A and B, from two different models, which are represented as a sequences of actions: $A = input \to a1, a2, a3, a4 \to output$ and $B = input \to b1, b2, b3, b4$.

In separate models, the initial actions of both processes are started by the environment, meaning that both $a1$ and $b1$ are located actions. If the models are composed, then the sequence of actions

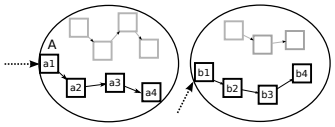


Figure 5: Models with processes started by the environment

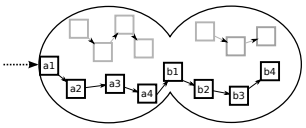


Figure 6: The two models have been composed. The second process is started by the first instead of the environment

might look like Figure 6. The sequence of actions is now $input \rightarrow a1, a2, a3, a4, b1, b2, b3, b4$

Now, only one process is started by the environment. What was once the second process, B, is now started at the end of A.

A more complicated example occurs when the composition does not occur at the end of a process. In this case, the initial processes are shown in Figure 7. The sequence of actions for process A is $A = input \rightarrow a1, a2, delay, a3, a4 \rightarrow output$ and the sequence for process B is $B = input \rightarrow b1, b2, b3, b4 \rightarrow output$.

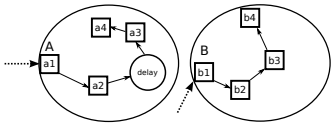


Figure 7: Two models with processes started by the environment. Process A has a special action, `delay`

In A, there is a special action, $delay$ which is where composition would occur. In an uncomposed executing model, this stops the process for a period of time before continuing. When composed, the process looks like Figure 8. The sequence of actions is now $input \rightarrow a1, a2, b1, b2, b3, b4, a3, a4 \rightarrow output$.

The delay in process A simulates the time that would have elapsed if the model were more detailed. This detail is present in the composed model and so the delay is no longer needed.

3.4 Implementation in julia

The modelling framework is implemented in the julia language, a new language designed for scientific computation. The features of the language make it relatively straightforward to implement the concepts required for modelling.

Processes, for example, are represented naturally with julia's Tasks—commonly known as coroutines. In combination with a scheduler, this allows processes to run, wait for a period of time, and resume execution. In the example below, the very simple process starts, suspends for 5 hours of simulation time, and then terminates.

```
function example_process(proc :: Process)
    hold(proc, 5hours)
end
```

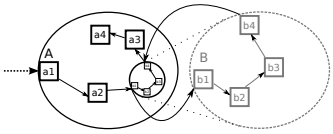


Figure 8: The two models have been composed. Process B has now been substituted for the `delay` action in process A

Resources are simply types in julia that are subtypes of the abstract `Resource` type. For example, `Agent` is an abstract subtype of `Resource` and `Employee` is a subtype of `Agent`. The `Employee` type has a field, `data`, which is used for storing information about this agent: preferences, what it is carrying, etc.

```
abstract Agent <: Resource
type Employee <: Agent
    data :: AgentData
end
```

Locations are created and then linked together.

```
local loc_atrium = Location("Atrium")
local loc_office = Location("Office")
link(loc_atrium, loc_office)
```

In addition to these basic elements, the framework also comprises a library of useful components, such as doors, access control mechanisms, and function libraries useful for building new models.

3.4.1 Agents

Each agent in the system is modelled using bundle of a process, a resource, and one or more locations. The resource marks the agent's physical location in the model. The agent's process moves this resource between the locations in the model, and also interacts with other resources as needed. The locations bundled with the agent are used to model concepts like holding or remembering. To model an agent picking up another resource, the agent's process would move that resource into the location that represents the things the agent is carrying; dropping the resource would move it back to a physical location in the model.

3.4.2 Agent Decision Making

In addition to moving around and interacting with resources, agents must also make decisions. These decisions occur at specific points in the agent's process and the agent decides between a number of alternative choices. In theory, any method may be used to decide between the choices, but in practice the decision is based on the current state of the model and the agent's preferences.

In these models, we use a straightforward approach to decision-making: for each choice C, the agent calculates

$$val(C) = p_{sec} f_{sec}(C) + p_{prod} f_{prod}(C)$$

where p_{sec} and p_{prod} are the agents preferences towards security and productivity, respectively. The functions $f_{sec}(C)$ and $f_{prod}(C)$ determine the current value to the agent, based on the state of the model, of the choice C in terms of security and productivity. The agent then selects the highest valued choice.

The agent's preferences for security and productivity are drawn stochastically from distributions at the start of the execution of the model. The distributions should be chosen to reflect the population of agents in the system being modelled.

3.4.3 Composition

In the julia code for models, composition is implemented in a way that matches the mathematical definitions closely. For example, Figure 9 shows the code used to create the interfaces for the tailgating model. As can be seen in Figure 2, the tailgating model composes on two interfaces.

The first interface is for the Outside location, where the model composes with the device loss model. This is an input location—a process is started from the other model rather than the environment, when composed. The code creates the interface, then the input location, sets an environment processes responsible for starting the agents in the model when it is not composed, and defines which function should be executed from another model when composed with this model. The `agent_process` function corresponds to a located action in the mathematical definition of an interface above. The environment process, `start_agents` corresponds to the mapping of a probability to a located action; in fact, the function just calls `agent_process` for each agent at random times drawn from a distribution.

```
iface_outside = Interface()
il = InputLocation()
push!(il.env_processes,
  Process("start_employees", start_agents))
il.functions[Agent] = agent_process
iface_outside.input_locations["Outside"] = il
model.interfaces["Outside"] = iface_outside

iface_atrium = Interface()
ol = OutputLocation()
ol.functions[Agent] = in_atrium
iface_atrium.output_locations["Atrium"] = ol
model.interfaces["Atrium"] = iface_atrium

push!(model.env_processes,
  Process("start_attackers", start_attackers))
```

Figure 9: Code for creating interfaces in the tailgating model

The second interface is for the Atrium location, where the tailgating model composes with the document-sharing model. It is an output location: the agent and attacker processes from this model continue into the document-sharing model when it is composed. When not composed, the processes execute the `in_atrium` function as set in the code. This function just pauses the processes for a set amount of time—it is the $delay$ action from above. When composed, the actions in the other model are executed instead of this.

Finally, composition itself is similar to the mathematical definition. The `compose` function takes two models and specified interfaces and returns the composed model. Figure 10 shows the code for composing all three of the models together. First the device loss and tailgating models are composed together, and then the resulting model is composed with the document-sharing model.

4. SYSTEM MANAGERS' UTILITY

This modelling methodology is designed to allow system managers to explore the effects and consequences of various policy decisions on a a system. A necessary component of this is the ability to express how good the performance of a system is given certain particular policy choices and the system manager's preferences.

```
m1 = create_device_loss_model()
m2 = create_tailgating_model()
m3 = create_document_sharing_model()
temp_m = compose(m1, "Outside", m2, "Outside")
m = compose(m3, "Atrium", temp_m, "Atrium")
```

Figure 10: Code for composing the three models together

We can express this value—the utility of a policy choice—in terms of a set of attributes that the system manager cares about [17, 20]:

$$Utility = \sum_{a \in A} w_a f_a(v_a - \bar{v}_a)$$

Each attribute, a, has a function f_a that determines the system manager's response to the deviation of the value of the attribute, v_a from its target value, $\bar{v}_a$. For example, for one attribute the system manager might not care about missing the target by a small amount, but for another attribute a small deviation from the target could be critical. Each attribute also has a weighting, w_a, describing its relative importance compared to others. When models are composed, the overall utility simply adds together the component utilities.

The target values, weightings, and deviation functions for the attributes are obtained from the system manager. To evaluate the performance of current policy, the values for each of the attributes can be obtained from the real world. To evaluate changes to policy, the values of the attributes can be obtained from models.

The values of the attributes can depend on the decisions that agents make during an execution of the model. A decision, D, made by an agent can be written in Cobb-Douglas form [15] as

$$D = C_1^{\lambda_1} C_2^{\lambda_2} \dots C_n^{\lambda_n}$$

where $C_1, C_2, \dots, C_n$ are the values to the system manager of the choices available for that decision. The exponents $\lambda_1, \lambda_2, \dots, \lambda_n$ take the value 0 for the choices not selected, and the value 1 for the choice that was selected.

Each agent makes a series of decisions as the model executes, and the set of the decisions made by all of the agents during the execution of a model can be used when calculating the values of the attributes in the system manager's utility function. Once those values have been obtained, the utility for that execution of the model can be determined.

4.1 Expected Utility

Knowing the utility of a single execution of a model is not that useful if the model contains stochastic elements—it is just one of many possible outcomes. Instead, we must find the expected utility

$$E[Utility] = E\left[\sum_{a \in A} w_a f_a(v_a(\Phi_a) - \bar{v}_a)\right]$$

where the expected value for each attribute a depends on the stochastic process Φ_a. Again, expected utilities for composed models simply add those of the components.

The values of these processes also depend on the decisions individual agents make. However, we need to consider the expected value of these decisions. A standard transformation of the Cobb-Douglas form [25] gives

$$E[D] = \lambda_1 ln(C_1) + \lambda_2 ln(C_2) + \dots + \lambda_n ln(C_n)$$

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are now the probabilities of the agent selecting each choice, and sum to 1.

These probabilities, along with each Φ_a, depend on the state of the model, which in turn depends on stochastic elements of the model and its environment. In theory, these values can be calculated starting with the known probability distributions and using queueing theory to work out the subsequent states of the model. In practice, this is extremely difficult and an approximation of these values must be used. This is done by executing the model a large number of times, to get an approximate distribution of values. The accuracy of this estimation increases with the number of executions.

5. MODELS AND EXAMPLES

5.1 Models

We will use three different models that examine different areas of security policy as examples. The first of these models is a tailgating model, which looks at the behaviour of employees and attackers at the entrance to an office building. The second model looks at the behaviour of employees inside the office, when they are making decisions about how to share confidential documents with other employees. The final model is extremely simple, and looks at the loss of devices, possibly containing confidential data, outside of the office. These three models are depicted in Figure 2.

5.1.1 Tailgating

The tailgating model looks at how different security policy choices at the entrance to the building affect the behaviour of employees and attackers. This model is shown in the centre of Figure 2. In the model, employees need an ID card in order to get through the security door. When employees forget their cards, they must make a decision: to queue at the reception desk to get a temporary ID card for the day, or to tailgate through the door. Employees that have passed through the security door observe if others tailgate behind them; if so, the employee has another choice: to ignore the tailgater and continue straight to the office, or to stop the tailgater and send them back to reception. There are also attackers in this model, who always try to tailgate and do not have any decisions. They can be stopped by security guards and by employees' challenges.

5.1.2 Document Sharing

The document-sharing model looks at the behaviour of employees inside the office when they must decide how to share confidential documents with each other. This model is shown on the right side of Figure 2. Normally, employees share documents using a shared drive that restricts access to each document to employees with the appropriate authorization. However, the system is often unavailable and employees must use an alternative means to share the documents. In the model, there are three choices. First, is to use a global share, which is accessible by all employees. Second, is to email the documents to the recipients. Finally, employees can use portable media, such as a CD or USB stick to share the data. Each of the three options has a drawback: documents on the global share can be access by any employee, documents sent to email end up on the devices which employees carry to and from work, and the portable media used to share documents gets left lying around the office. In this model, attackers walk around the office and pick up any portable media they find lying around.

5.1.3 Device Loss

The device-loss model is very simple: employees that return home from work have a random chance to lose devices they are carrying, which possibly contain confidential data. By itself, this model does not do very much. When composed, however, it becomes more useful: the number of confidential documents on the devices is determined by the behaviour of employees in the document-sharing model. This model is shown on the left side of Figure 2.

5.1.4 Composed Model

The three models are designed to compose together at specific interfaces. The Outside locations in the device-loss model and the tailgating model are part of the interfaces for those models. When composed, the models are joined at that location. The same is true for the tailgating model and the document-sharing model: they are composed at the Atrium location in each of the models.

Some of the processes in each model are also part of the composition. When the tailgating model is not composed, the Employee processes are started by the environment, and agents that make it through to the Atrium simply wait there for the duration of the work day before leaving. When the model is composed to the device loss model, the Employee processes start in that model, and then continue to this one. When the tailgating model is composed with the document -haring model, the employee and attacker processes in the document-sharing model are executed as consequences of the tailgating model, rather than started by the environment.

5.1.5 Validation

The parameters used in our experiments have been chosen to illustrate our methodology simply and clearly. However, the formulation of each of the example models is based directly upon extensive empirical studies of organizational behaviours (e.g., both employees and managers) in a large critical infrastructure company. Moreover, the experimental results of the simulations closely match observed behaviours in the parts of the company upon which the example scenarios are based.

5.2 Examples

We will use two fictional companies with different attitudes towards security for our examples. The first company places a high value on security and the confidentiality of its information; this could be a corporate research lab in a competitive field, or a national infrastructure company. The second company values security much less, and is more concerned with allowing people to get work done than keeping information secret.

The different attitude towards security in these companies is present in both the preferences of the employees and those of the organizations' security managers. In these examples, the distributions of employee preferences are chosen to reflect what is likely to be found in the respective companies. In reality, these distributions would be determined by sampling the preferences of actual employees at the organizations. The distributions we are using are shown in Figure 11. The highly-secure company contains employees who for the most part value security very highly over productivty. In the lower-security company, the employees are more likely to have a higher preference for productivity than for security.

In this example, the security manager of the high-security company is concerned about the number of confidential documents being shared on the globally-accessible shared drive, which can be accessed by all employees instead of being restricted to those who should have access. The manager is considering telling employees that, for security, they should share the documents with the intended recipients either by email or by using portable media. Both

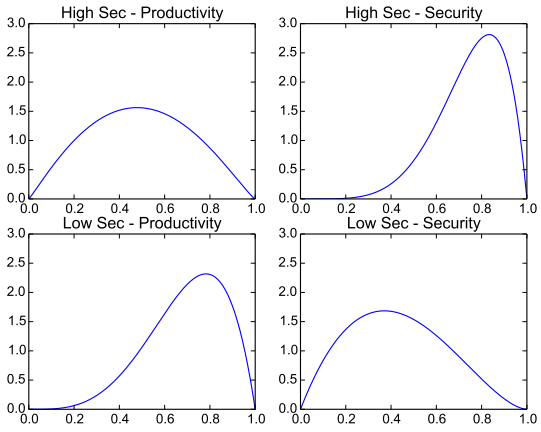


Figure 11: Employee preference distributions

options are less efficient (and so less productive) for the employees than using the global share; portable media is the least productive.

Table 2 shows the results for the document-sharing model. In the high-security company, employees have high preferences for security; most follow the instructions to use either email or portable media. In contrast, employees in the low-security company are much more concerned with productivity; the instructions have little effect. This shows how the security culture of a company and its employees can have an effect on the performance of policy.

Company	a	w_a	$\bar{v}_a$	$f_a(x)$
High Sec	Share	0.25	20	$-(e^{0.01x} + 0.01x - 1)$
	Found	1	0	$-e^x + 1$
	Lost	1.1	0	$-e^x + 1$
	Rcpt Time	0.01	3000	$-.002x$
	Tgate	1.5	0	$-e^x + 1$
Low Sec	Share	0.1	200	$.003x$
	Found	0.3	0	$-e^x + 1$
	Lost	0.4	0	$-e^x + 1$
	Rcpt Time	0.4	2000	$-.005x$
	Tgate	0.2	0	$-e^x + 1$

Table 1: Utility parameters

The weights, targets, and deviation functions for each attribute used in the utility calculations are shown in Table 1. For the document-sharing model, only the first two attributes (the number of documents on the globally accessible share and the number of documents on portable media found by attackers in the office) are used. The security managers of the high and low security companies have different weightings, targets, and deviation functions. The high security manager is trying to reduce the number of documents shared globally and so has a low target value. A Linex function [26] is used for the deviation function for this attribute: beating the target results in a linear increase in utility; missing it results in an exponential loss. The low security manager actually prefers that employees use the global share as it is more efficient and confidentiality is not an issue. Both managers have an exponential response to attackers finding documents on portable media inside the office, but have different weightings.

In this case, the high-security manager would prefer the email policy, as it produces the lowest number of documents on the global share, and does not leave media devices lying around the office to be found by attackers. The low-security manager prefers no change of policy: employees should keep using the global share.

Company	Policy	Email	Share	Media	Found	$E[U]$
High Sec	Email	151.73	7.75	0.00	0.00	0.059
	None	0.00	167.32	0.00	0.00	-1.209
	Media	0.00	54.42	105.33	2.43	-10.543
Low Sec	None	0.00	167.29	0.00	0.00	-0.010
	Email	54.87	107.96	0.00	0.00	-0.028
	Media	0.00	154.15	12.24	0.07	-0.035

Table 2: Results from the document-sharing model

The managers would like to know that these policies are still optimal when considered as part of the larger system. Composing the document-sharing model with the tailgating and device loss models allows the managers to see the interactions between policy choices in each model. In the tailgating model, the managers must choose the number of security guards and receptionists, which can affect how many attackers gain access to the office. The managers also care about more attributes from the other two models: the number of documents on lost devices, the number of times attackers successfully tailgate, and the amount of time employees spend at reception. The weightings, targets, and functions for these attributes are shown in Table 1. The high security manager cares less about the time taken at reception and more about the number of tailgates and devices lost. In contrast, the low security manager cares more about reducing the time employees spend at reception.

Table 5.2 shows the results from the composed model for some parameter combinations. For the high-security manager, the email policy is still the best option as long as there is at least one guard. The difference in expected utility between the email and media policies is much smaller than in just the document-sharing model as the guards and challenges by employees prevent attackers from accessing the office. For the low-security manager, having two receptionists is the most important factor: it greatly reduces the time employees spend at reception. Using two guards instead of one actually results in less expected utility, as the guards catch more employees tailgating, who must then spend the time at reception. The influence of employee preference can be seen again in this composed model: for the same number of guards in the low- and high-security companies, there are more tailgates in the low-security company because the employees do not challenge tailgaters.

6. CONCLUSIONS AND FURTHER WORK

We have presented a rigorously grounded, compositional modelling framework and methodology capable of capturing the consequences and values of systems security policy design decisions. We have demonstrated this framework in some simple, but empirically supported, examples of concern to practising security managers that illustrate all aspects of the methodology and its value in supporting their decision-making.

Much further work is possible. Our framework would be enriched by employing better models of agents' decision-making, taking account of the relevant literature in psychology [12]. Similarly enriching would be to introduce the ability of agents to adapt to changing circumstances during the execution of a model [6].

Company	Policy	#Grd	#Rcpt	Email	Share	Media	Found	Lost	Rcpt Time	Tgate	$E[Utility]$
High Sec	Email	2	2	174.20	8.28	0.00	0.00	0.14	2756.80	0.09	-0.241
	Email	2	1	174.60	8.12	0.00	0.00	0.14	4469.21	0.09	-0.278
	Email	1	2	173.92	8.35	0.00	0.00	0.16	2754.11	0.23	-0.513
	Email	1	1	173.67	8.35	0.00	0.00	0.14	4532.56	0.23	-0.523
	Media	2	2	0.00	65.62	129.38	0.23	0.00	2782.38	0.09	-0.661
Low Sec	None	1	2	0.00	203.79	0.00	0.00	0.00	1754.50	0.74	0.273
	Media	1	2	0.00	187.34	14.99	0.05	0.00	1745.54	0.74	0.270
	Email	1	2	67.21	130.53	0.00	0.00	0.06	1734.02	0.74	0.267
	None	2	2	0.00	203.48	0.00	0.00	0.00	1837.37	0.29	0.258
	Media	1	1	0.00	187.55	15.25	0.04	0.00	2403.60	0.74	-1.043

Table 3: Results from the composed model

Concerning experimental methodology, we need to develop a more systematic approach to analyzing the sensitivity of model executions and utility to parametrization choices [18]. More generally, a systematic methodology for exploring how managers' elicited preferences [1, 4] can be represented in model structure and parametrization should be explored.

Associated with the process-algebraic tools described in Section 2 is a modal logic, in the tradition of Hennessy–Milner logic [16, 9], in which propositions assert properties of system states. The logic is given semantically by a satisfaction relation, $L, R, E \models \phi$, inductively defined over the logic's connectives, and read as 'the process E, executing at location L with available resources R has property ϕ'. The logic's connectives include additives and multiplicatives of the bunched logic BI [22] and both additive and multiplicative modalities [9], so rendering it capable of expressing both structural and dynamic properties of models. Model checking—with some initial work described in [9], but much more work required—would provide a framework within which stateful properties of systems, as well as utility-theoretic properties of policies, such as optimality, can be verified. A further step would be to incorporate the stochastic aspects of implemented models into model-checking tools, perhaps building on ideas in, for example, PRISM [23].

7. REFERENCES

[1] *AI Magazine: Preferences*. J. Goldsmith and U. Junker (editors). Volume 29, No 4, Winter 2008.

[2] A. Beautement et al.. Modelling the Human and Technological Costs and Benefits of USB Memory Stick Security. In *Managing Information Risk and the Economics of Security*, M. Eric Johnson (ed.), Springer, 2008. 141–163.

[3] A. Beautement, M. Angela Sasse, and M. Wonham. The Compliance Budget. Proc. New Security Paradigms Workshop, 2008 (NSPW '08), 47–58. ACM New York, 2008. doi: 10.1145/1595676.1595684

[4] Y. Beres, D. Pym, and S. Shiu. Decision Support for Systems Security Investment. In *Proc. Business-driven IT Management (BDIM)*, IEEE Xplore, 2010, 118–125. doi: 10.1109/NOMSW.2010.5486590

[5] T. Caulfield, D. Pym, and J. Williams. Compositional Security Modelling: Structure, Economics, and Behaviour. LNCS 8533:233–245, 2014.

[6] J. Cohen, S. McClure, and A. Yu. Should I stay or should I go? How the human brain manages the trade-off between exploitation and exploration. *Phil. Tran. Roy. Soc. B: Biol. Sci.* 362(1481), 933–942, 2007.

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

[8] M. Collinson, B. Monahan, and D. Pym. Semantics for structured systems modelling and simulation. *Proc. SIMUTools 2010*, 34:1–34:10. ACM Digital Library. doi:10.4108/ICST.SIMUTOOLS2010.8631

[9] M. Collinson, B. Monahan, and D. Pym. *A Discipline of Mathematical Systems Modelling*.College Publications,2012.

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

[11] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 3rd ed., 2000.

[12] C. Fang and D. Levinthal. Near-Term Liability of Exploitation: Exploration and Exploitation in Multistage Problems. *Organization Science* 20(3), 538–551, 2009.

[13] L. Gordon and M. Loeb. The Economics of Information Security Investment. *ACM TISSEC*, 5(4):438–457, 2002.

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

[15] D. Heathfield. *Production Functions*. Macmillan Press, 1971.

[16] M. Hennessy and G. Plotkin. On observing nondeterminism and concurrency. LNCS 85, 299–309, 1980.

[17] C. Ioannidis, D. Pym, and J. Williams. Information security trade-offs and optimal patching policies. *European Journal of Operational Research* 216(2):434–444, 2011.

[18] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley and Sons, 1991.

[19] julia. `http://julialang.org`.

[20] R.L. Keeney and H. Raiffa. *Decisions with multiple objectives*. Wiley, 1976.

[21] R. Milner. Calculi for synchrony and asynchrony. *Theoret. Comp. Sci.*, 25(3):267–310, 1983.

[22] P. O'Hearn and D. Pym The Logic of Bunched Implications. *Bulletin of Symbolic Logic* 5(2), June 1999, 215–243.

[23] The PRISM Model Checker. `http://www.prismmodelchecker.org`

[24] S. Ross. *Introduction to Probability Models*. Eighth Edition, Academic Press, 2003.

[25] H. Varian. Intermediate Microeconomics. W.R. Norton & Company, 2014

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