Assessing the Value of Investments in Network Security Operations: A Systems Analytics Approach♦

Jonathan Griffin, Brian Monahan, David Pym, Mike Wonham, Mike Yearworth
Trusted Systems Laboratory
HP Laboratories Bristol
HPL-2007-89
May 31, 2007*

modelling,
mathematics,
information security,
operations,
business value

Assessing the value of investments in network security operations remains a challenging problem. We suggest that an essential component of an analysis of this problem must be an account of the structure of the system/network and the services it is intended to deliver. We apply the methods of classical applied mathematics - using tools drawn from algebra, logic, probability theory, and theoretical computer science - to represent systems, services, and information flows in order to assess the value of network and security operations deployed in response to environmental threats and the requirements of business alignment. We use Monte Carlo experimentation to explore the levels of investment in, and trade-offs between, operations staff and security control devices necessary to maintain network availability of value determined by a given Service Level Agreement. We conclude that our methods deliver useful analyses and identify necessary future work required properly to integrate models of spatially distributed networks, stochastic environmental behaviour, and system value.

* Internal Accession Date Only
♦ Workshop on the Economics of Information Security, 7-8 June 2007, Carnegie Mellon University, Pittsburgh, PA, USA
Approved for External Publication

Assessing the Value of Investments in Network Security Operations: A Systems Analytics Approach

Jonathan Griffin, Brian Monahan, David Pym*, Mike Wonham, and Mike Yearworth

Trusted Systems Lab, HP Laboratories
Bristol BS34 8QZ, UK

Abstract. Assessing the value of investments in network security operations remains a challenging problem. We suggest that an essential component of an analysis of this problem must be an account of the structure of the system/network and the services it is intended to deliver. We apply the methods of classical applied mathematics — using tools drawn from algebra, logic, probability theory, and theoretical computer science — to represent systems, services, and information flows in order to assess the value of network and security operations deployed in response to environmental threats and the requirements of business alignment. We use Monte Carlo experimentation to explore the levels of investment in, and trade-offs between, operations staff and security control devices necessary to maintain network availability of value determined by a given Service Level Agreement. We conclude that our methods deliver useful analyses and identify necessary future work required properly to integrate models of spatially distributed networks, stochastic environmental behaviour, and system value.

1 Introduction

Computer networks that are connected to the internet are now a predominant feature of business and social life at all scales, from small businesses to multinational corporations, from individuals to governments. Indeed, to operate efficiently, particularly in a distributed manner, many organizations, both large and small, depend upon computer networking as an essential management tool through which most operational activity is either coordinated or mediated.

Because of this dependence, any disruption of an organization's network services can be extremely costly, and might even threaten the continued existence of the organization as a functioning business entity. Because of the improving reliability of the physical infrastructure, combined with the inherent resilience of high-level connections provided via packet switching and TCP/IP, a large proportion of network downtime arises from security issues at the local network's points of contact ('endpoints') with the outside world. These endpoints include not only internet connections themeselves but also local client systems, mail servers, web servers, database systems, and more.

Organizations typically respond to the occurrence of network security issues by investing in information security operations [15–17, 35]. Such investments are considered necessary to protect the confidentiality, integrity, and availability of an organization's information systems against attacks, and represent substantial investments in technologies, tools, and human resources. The objective of these investments is to allow the organization to maintain its operational (e.g., trading) activities and so security operations investment decisions must be consistent with the organization's overall operational and financial models. Indeed, estimating the return on ICT investments must be based on a desire to avoid losses arising from externalities, such as indiscriminate worms and viruses or highly focussed attacks. Such an analysis is challenging [3, 2] and requires an approach to monetizing the loss of availability, integrity, and confidentiality, and to estimating the risk of occurrence. Therefore, we should no longer regard ICT security as being purely a technology issue, at least within the sphere of corporate business. Rather, it is helpful to regard ICT security as a process — something that is done to create a smooth environment for the organization's operations — typically with specific cost objectives.

* Corresponding author. Email: david.pym@hp.com

In this paper, we explain an approach to assessing the value of investments in network security operations that is based on *systems analytics*. We begin, in § 2, with an explanation of our perspective, paying particular attention to the rôle of systems modelling, including the need to understand information security operations as a process, as opposed to a product or pure technology, and the influence of Activity-based Costing. Then, in §§ 3.1, 3.2, and 3.3, we explain our modelling philosophy and the conceptual and mathematical bases for our modelling approach, including an account of how our framework is realized in our modelling tool. In § 4, we discuss the nature of network/security operations and discuss how, in the context of availability, the value of security operations can be represented by the terms of the Service Level Agreement (SLA) associated with its management, and explain the ket aspects of (i) a basic model of the operational response to threats to a system, and (ii) a model of the operations required to support the placement of security controls (such as firewalls and similar devices). In § 5, we give a detailed description of our model, as implemented in our modelling tool, Demos2000. We explain the structure of the model in terms of our conceptual framework, and illustrate some key aspects of the code. In § 6, we outline some of our experimental results, rendered as the output of Monte Carlo simulations. Finally, in § 7, we summarize our achievements to-date and discuss the research directions arising from our work.

The scope of this article is rather wide. For that reason, for our present purposes, we make illustrative, rather than comprehensive, choices of citations of the literature.

2 Assessing the Costs and Benefits of Network Security Operations

2.1 The Systems Analytics View of Information Security Economics

In recent years, there has been a good deal of research activity addressing topics within what has become known as information security economics. The scope of this field is, perhaps, well-drawn by the contents of [9].

An examination of the contents of [9] reveals, however, a quite disparate collection of activities. We suggest that an appropriate unifying framework for addressing the concerns of the economics of information security is provided by an idea of *systems analytics*.

The key observations of systems analytics are, we suggest, the following:

- That the economics of information security is broadly concerned with two main issues:
  1. Modelling the various markets in vulnerabilities (for example, [10]) and in the delivery of information security operations, such as basic questions of strategy (for example, [21]) or the market for security outscourcing (for example, [18]); and
  2. Costing and valuing the investments in the security operations that are applied to the systems that deliver services based on information that requires, at least to some extent, protection;
- That the value of information security operations can only be understood in the context of the systems, and the services delivered by those systems, to which they are applied;
- That, therefore, an appropriate basis both for analysing the markets in the delivery of information security operations and valuing the investments in security operations in particular systems is provided a by systems modelling approach.

Implicit in each of these is the need to address, among many things, topics such as quantification and measurement of risk (e.g., [31]).

In the context of ICT systems — surely the leading example of interest — we would emphasize that our view of systems here includes (descriptions of) all of the following:

- The environment, including the threats, within which the system resides;
- The network and devices of the system;
- The services delivered by the system; and
- The users of the system, both external and internal.

In the context of information security economics, systems analytics is concerned, then, with providing mathematically rigorous models of systems, in the sense described above, that are able to capture those properties of the system (e.g., aspects of dynamics, of resource usage, of policies and protocols, of Service Level Agreements, and so on) that are pertinent to answering questions about the two main issues described above; that is, the markets in the delivery of information security operations, and the valuing of investments in the security operations that are applied to specific systems. In this paper, we shall be concerned with the latter issue.

2.2 Security Operations Processes

Information security operations — necessary to protect the confidentiality, integrity, and availability of an organization's information systems against attacks — represent substantial investments in technologies, tools, and human resources. The objective of these investments is to allow the organization to maintain its operational (e.g., trading) activities and so security operations investment decisions must be consistent with the organization's overall operational and financial models. Indeed, estimating the return on ICT investments must be based on a desire to avoid losses arising from externalities, such as indiscriminate worms and viruses or highly focussed attacks. Such an analysis is challenging (Anderson 2001), and requires an approach to monetizing the loss of availability, integrity, and confidentiality, and to estimating the risk of occurrence. Therefore, we should no longer regard ICT security as being purely a technology issue, at least within the sphere of corporate business. Rather, it is helpful to regard ICT security as a process that is executed in order to create a smooth environment for the organization's operations. In order to understand the value of operational processes, modern corporate management makes essential use of metrics to assess impact upon business performance, so enabling an understanding of the consequences of operational decisions for key indicators, such as shareholder value. For security, the question that then arises is that of how to measure the level of security performance that these security-related processes achieve [17]. In order to formulate this question precisely, it is necessary to understand the goals of security operations and to have mechanisms for predicting the outcomes of specific processes. Then it becomes possible to integrate ICT security investment decisions within an organization's overall operational and financial processes.

2.3 Activity-based Costing (ABC)

Traditional management accounting methods were typically not able to accurately manage and separate *direct costs* (e.g., expenditure identified specifically with a particular business project) from *indirect costs* (e.g., expenditure on services which cannot be economically identified with a particular project). The indirect costs would be allocated across the identifiable projects, perhaps on a pro-rata overhead basis. Such an allocation places burdensome overheads upon some projects, potentially distorting overall performance assessment (e.g., projects that had otherwise adequate performance would appear to be marginal due to the taxation imposed by allocation of indirect costs). As a result, management decision making would be based upon a distorted view of expenditure, probably leading to poor performance, poor profitability, or even operational failure.

ABC, introduced by Kaplan and Bruns, and Cooper, [20] and subsequent papers in the *Harvard Business Review*, is in part a response to this confusing situation. The main idea is simple: determine what are the operational (e.g., business) activities and what are their outputs (e.g., products, services) and accurately determine their resource usage. Typically, the benefits result from the outputs of business activities and the costs arise as a result of resource usage by those activities. Thus, more refined and accurate cost structures are developed within which a more accurate allocation of costs can be achieved. In this way, ABC identifies cause and effect relationships to which costs can be more accurately assigned and so eliminates many sources of indirect cost. A substantial benefit is that management can more clearly identify the sources and causes of high costs within the business. In some cases, costs may well be deemed to be justified and accepted as the cost of doing business; in others, however, costs may exceed the benefits achieved, suggesting possible re-evaluation and reorganization.

3 The Systems Analytics Approach

In this section, we explain informally the theoretical basis of our modelling technology by describing our semantic analysis of the structure and dynamics of the systems that we aim to model.

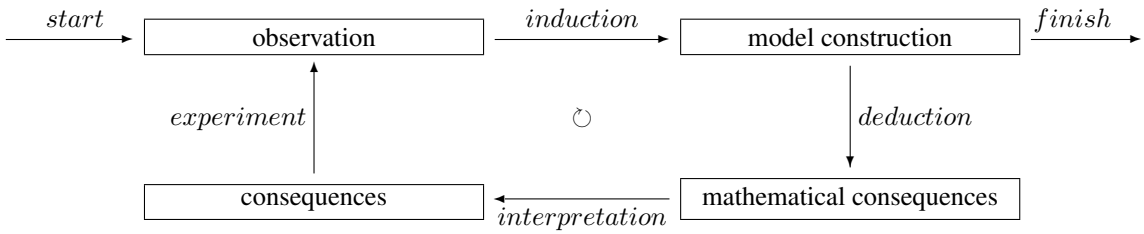
We begin, in § 3.1, with an explanation of the style — that of classical applied mathematics in engineering — used to construct our models, paying particular attention to the influence of Activity-based Costing (ABC). In § 3.2, we give a brief account of the basic concepts of interest to us; then, in § 3.3, we give a brief description of its mathematical realization in the synchronous calculus of resources and processes (SCRP, for short) and an associated modal logic (MBI) [26, 27]. Next, in § 3.4, we introduce the language of current modelling tool, Demos2000; then, in § 3.5 we give a brief summary of the extent to which our framework is partially realized by the Demos2000.

Later, in § 5, we explain how each of the ideas we have explained in this section arises in a particular model of interest.

3.1 The Modelling Process and Style

The use of mathematical models in engineering has a long and distinguished record of success. From earthworks to suspension bridges, from bicycles to spacecraft, mathematical models are used to predict behaviour and give confidence that necessary properties of the constructions — such as capacity, resilience, and cost — obtain.

Such applications of applied mathematics in engineering are useful, and usable, by virtue of the scientifically rigorous modelling methodology, summarized conveniently in the following diagram, in which the left-hand side inhabits the physical world and the right-hand side the mathematical world:



The modelling process begins with *observations* of the world, inducing the definition of a mathematical *model*, of which properties are *deduced*. These mathematical consequences of the definition of the model are then interpreted in the physical world and, experimentally, the extent of their validity is observed. The cycle is repeated until a model that is judged to be sufficiently accurate is obtained. A critical aspect of this approach concerns the choice of level of abstraction: it is important to represent in a model just those features of the phenomenon of interest that are pertinent to the properties and questions of interest.

In the worlds of traditional (mechanical, civil/environmental and even, to a large extent, electrical and electronic) engineering, the mathematical methods used are mainly those of calculus and differential equations.

Our approach to systems modelling is to apply the classical modelling cycle to understanding large, complex networks of information processing devices. In this world, the appropriate mathematical methods are more discrete, being drawn from algebra, logic, theoretical computer science, and probability theory.

In order to apply these methods, we require a conceptual analysis of the relevant aspects of the systems of interest. Our analysis, which distinguishes concepts of *resource*, capturing the essentially static components of the system, of *process*, capturing the dynamic components of the system, of *location*, capturing the spatial distribution and connectivities of the system, and of *environment*, within which the system functions, is described in some detail in §§3.2 and 3.3.

From the perspective of addressing economic, or at least business-value, questions, a certain aspect of our modelling style is important. Specifically, when deciding how to capture the relevant dynamics of a system — such as the activities of a network's security operations staff — we structure the representation so as to reflect the activity-based costing (ABC) approach recalled in § 2.3 above.

Our use of stochastic, probabilistic, methods is critical. Systems inhabit complex environments within which pertinent events occur. Typically, there is no realistic hope of our being able to model all of pertinent environmental events exactly. Indeed, typically that will evidently be impossible. Instead, we capture the incidence of events upon our systems using probability distributions. For example, we might represent the arrival of requests at a web-server by a negative exponential distribution with a rate parameter that gives a realistic representation of the load on the server. In general, such Markov-style methods set up systems of queues within the structural framework of resources, processes, and locations.

Finally, we must validate our mathematical models against the external physical world. Our approach is essentially that of Monte Carlo simulation: our models, though mathematically rigorous in a precise sense described in the sequel, are coded in an executable language (described in §§3.4, 3.5, and Appendix A).

3.2 The Conceptual Framework

Our conceptual framework for modelling the integrity and performance of systems is based on an analysis of four key concepts.

Resource: Informally, we consider resources to be the consumable, static components of a system. Examples include computer memory, processor cycles, docking bays in a port, and money, among many other similarly natural examples. Considering these examples, it is natural to require that it should be possible to combine resources. For one example, two pots of money may be combined to form another (larger) pot of money. For another, two regions of computer memory maybe combined provided they do not overlap. Conditions such as this latter one are known as separation conditions and have been studied in some detail in, for example, ([29, 19, 30]). It is also natural to be able to compare resources. For example, we might compare pots of money, as above, or the size of a region of computer memory. For another example, resources might be regions of a file system ordered by an access control régime such as the Bell-La Padula protection model.

Process: We consider processes to be the dynamic parts of systems, which manipulate — for example, consume, move, combine — resources. For instance, the basic functions of an operating systems, or the movements of a boat around a dock as it arrives, is loaded or unloaded, and departs. Fortunately, there is a well-developed mathematical theory of processes [22, 23, 26, 27] which is well-suited to our purposes. The key aspects of [26, 27] are sketched very briefly below, without theorems.

Location: The idea of location is an intuition derived from the physical concept of place. For one example, regions of memory to which an operating system must write may be located on different chips. For another, a port may have several docking bays and the choice of which to use for any given boat may depend on the widths and depths of the access channels to the different bays. Considering the various examples of location, it seems that a model should capture the following features: there should be a notion of sub-location, a notion of connection between locations, and a notion of zooming in and out to consider more or less detail of the 'map'. For more technical reasons, we also need a notion of the product of locations. In the security domain, a concept of location arises naturally and essentially. For example, an infection starts at some place(s) then spreads around the system following an epidemiological pattern that depends, among other things, on the connectivity of the system components. Understanding the topological properties of the system, such as connectivity, is therefore essential to determining the appropriate deployment of countermeasures.

Environment: Systems, be they IT, social, business, or physical, are intended to perform functions within an environment of events. In our framework, we capture the environment of events stochastically. That is, we assume that given classes of events occur — that is, are incident upon the system of interest — with given probability distributions. For example, we might assume that boats arrive

at a port according to a negative exponential distribution with parameter. Given such an assumption, it is then natural to understand the flow of resources between locations, described as a collection of processes, as a system of queues.

3.3 The Mathematical Framework

Our mathematical framework reflects the conceptual structure outlined above. Each of the key concepts, resources, process, and location, is captured mathematically by considering the basic mathematical properties that we expect of it. So, our methods are those of classical applied mathematics but using mathematical tools drawn from logic, theoretical computer science, and probability theory.

Resource: We have seen that the key aspects of resource that we want to capture are unit, composition, and comparison. Mathematically, we capture these properties by requiring that resources carry the structure of a pre-ordered, commutative, partial monoid (subject to some mild algebraic conditions). Specifically, we write

$$\mathcal{R} = (\mathbf{R}, \circ, e, \sqsubseteq)$$

to denote the evident quadruple, consisting of an underlying set of resources ($r, s \,\varepsilon\, \mathbf{R}$), a composition operation, $r \circ s$, on resource elements, a unit, e, (i.e., the identity element for composition), and a comparison relation, $r \sqsubseteq s$, so defining the ordered monoid.

An example, corresponding to money, is given by the natural numbers combined using addition, with unit zero, and ordered by less than or equals.

For SCRP, we take sets of resources R, S ($\varepsilon\, \wp(\mathbf{R})$, the powerset monoid over $\mathbf{R}$) over such an underlying monoid, lifting the monoidal structure in a straightforward way.

Composition: The composition of a set of resources is given by the set of compositions of elements of the underlying set. The unit is then e.

Comparison: There are some choices, one example being $R \sqsubseteq S$ if, for all $r \,\varepsilon\, R$, there is an $s \,\varepsilon\, S$, such that $r \sqsubseteq s$.

Process: Our model of process is based on a development, SCRP, of Milner's SCCS (Milner 1983) which integrates our model of resource. The basic components of SCRP can be summarized as follows:

Actions: which carry the structure of a monoid — so any two actions a and b can be combined to form a composite action, $a\#b$, with a unit, $\mathbf{1}$;

Combinators: which allow processes to be built out of actions. The combinators we take are action prefix, which allows the sequencing of actions, non-deterministic choice between processes, concurrent composition of processes, a local binding of resources to processes, and recursion (via constant definitions). Formally, the grammar of processes, E, is given as follows, following the order of introduction above:

$$E ::= a \mid a : E \mid E + F \mid E \times F \mid (\nu R)E \mid C{:=}E.$$

The connection between actions and processes is expressed using a modification function, μ, a partial function that maps an action together with a resource to a resource. The meaning of this syntax is given by an operational semantics, expressed as a collection of inference rules. For example, the rule for action prefix

$$\frac{\mu(a, R) \quad \text{defined}}{R\,,\; a : E \xrightarrow{\;a\;} \mu(a, R), E},$$

says that, provided the effect of the action a on the resource R be defined, then, with resources R the process $a : E$ can evolve by the action a to become the process E with resources $\mu(a, R)$ as specified by μ. Another example is given by the rule for concurrent composition,

$$\frac{R\,,\; E \xrightarrow{\;a\;} \mu(a, R)\,,\; E' \quad S\,,\; F \xrightarrow{\;b\;} \mu(b, S), F'}{R \circ S\,,\; E \times F \xrightarrow{\;a\#b\;} \mu(a\#b, R \circ S), E' \times F'},$$

provided the modification $\mu(a\#b, R \circ S)$ is defined. This rule expresses how we form a concurrent process from its components. Notice that the composite resource is determined by the composition in the resource monoid, where our separation conditions can be imposed. The rules for the other combinators are expressed similarly. Finally, we remark that equality between SCRP processes is given by bisimulation [22, 23]) relative to a given resource and modification function.

Location: Our model of location, capturing the conceptual requirements described above, is very simple. We require a mathematical structure that supports the following:

- A collection of locations, L, L', M, M', etc;
- Substitution of locations, $M[L'/L]$, of a location L' for a sub-location L of M . This idea requires a suitable notion of arity of location, so that substitution suitably preserves connectivity;
- A notion of connection between locations L and M;
- Finally, a product of locations.

Directed graphs provide a simple and natural example of a model of location.

Environment: As we have seen, we treat the behaviour of the environment stochastically. The simplest way to understand the relationship the stochastic environment and the process-theoretic structure of our models is via actions. An example will make things clear. Consider again the example of boats in port. We have a process that describes the movements of a boat around the port. But how do we initiate such a process? When a boat arrives at the port, according to the probability distribution for such arrivals, such as negative exponential or Poisson, the first action in a boat process is initiated. This happens for each incidence of a boat, and multiple boats execute in the system as concurrent processes (see above). Note that this gives us an example of a separation condition; concurrent boats may not share the same docking bay; their respective docking-bay resources must be separate.

Our mathematical framework admits also the possibility, not exploited in this paper, of providing a system of logic, tailored to our model of resources and processes, in which system properties can be expressed. The basic idea is to set up a logical judgement relating resources, processes, and their logical properties, expressed as

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

and read as the property ϕ holds — that is, is true of — of process E relative to the set R of resources. The relationship of truth between propositions, resources, and processes is defined inductively on the structure of propositions, with a case of the induction for each of the logical connectives that is supported by the semantics. A full discussion of the basics of this logic, called MBI, is provided in [26, 27] and is beyond our present scope. For now, however, we remark that logical assertions might be used, for example, to express access control policies for the system (model) described by the (SCRP) resource-process R, E, so that the judgement $R\,,\, E \models \phi$ will hold just in case the system (model) is compliant with the policy. MBI, standing for 'modal BI', is based on the logic of bunched implications, BI, introduced in [25, 28, 29, 14], and Hennessy-Milner logic [32]. The main point about BI is that it permits both 'additive' and 'multiplicative' logical connectives at the same level of abstraction as one another, thereby admitting, when combined with modalities in the sense of Hennessy-Milner logic, a logical analysis of the global and local properties of systems. This point may be seen quite quickly by considering the difference between the additive and multiplicative conjunctions, $\wedge$ and $*$, respectively. We have

$$R\,,\, E \models \phi \wedge \psi \quad \text{iff} \quad R\,,\, E \models \phi \quad \text{and} \quad R\,,\, E \models \psi,$$

which refers to the resource and process only locally. On the other hand, we have

$$R\,,\, E \models \phi * \psi \quad \text{iff} \quad S\,,\, F \models \phi \quad \text{and} \quad T\,,\, G \models \psi,$$

where R is the composition of S and T, and where E is bisimilar (the appropriate notion of equality for processes [22, 26, 27]) to $F \times G$. Notice here that the definition of the truth condition for $\phi * \psi$ refers to the global structure of the model of the system and, as a consequence, provides a logical characterization of concurrent composition [26].

As in Hennessy-Milner logic [23], the connection between the logic and the dynamics of the processes is facilitated by the modalities, $[a]$ and $\langle a \rangle$. In MBI, these modalities have both additive and multiplicative forms, analogous to the conjunctions described above. The additives may be seen as 'temporal' and the multiplicatives as 'spatial' or 'resourceful'. Specifically, we obtain a theorem [26, 27] to the effect that the appropriate notion of equality for (SCRP's) resource–processes corresponds exactly to logical equality in MBI. The details of this theorem — a key part of our framework — are beyond the scope of the present discussion.

All these logical ideas can be further enriched with our notion of location, leading to a logical judgement of the form $L\,,\,R\,,\,E\,\models\,\phi$ which, when given with an account of how resources are located using a modification function that is parametrized on locations, is read as property is true of the process E relative to resources R located at L. An example of a logical formula of interest would be an expression of an access control policy, with the overall judgement capturing the assertion that the system is compliant with the policy.

The operational counterpart, depending on the same parametrized modification function, then takes the form

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

where $\mu(a, L, R) = (L', R')$, with suitably parametrized rules for each of the combinators. The remaining active strand of research in this area concerns the key security concept of access control. We can consider a user or a group of users, or rôle, that is to say, a principal, to be a process that is being executed on a system; that is, relative to some resources that are situated at locations. Building on some ideas introduced by [1] we are able to make mathematically precise their conception that impersonation — hence groups, rôles, etc. — is a form of concurrent composition of the impersonated and impersonating principals. Using the ideas from logic to which we have alluded above, we can integrate specifications of access control régimes.

3.4 The Demos2000 Modelling Language

Demos2000 (henceforth Demos2k) [12] is a semantically justified discrete event modelling/simulation language implemented in Standard ML (see, for example, [24]) and freely available for download under an experimental licence from HP (www.demos2k.org). Appendix A contains some further discussion of Demos2k and the basic modelling concepts it supports directly. This work builds on [4].

In the next section we explain how Demos2k partially captures our conceptual framework.

3.5 Realizing the Conceptual Framework in Demos2k

The modelling language used in our examples, Demos2k, provides a partial realization and implementation of our mathematical model of our conceptual framework. Summarized below is the extent to which Demos2k (see Appendix A) captures our framework, concentrating on just the key points.

Resource: Demos2k has several notions of resource, primarily bins and resources. Bins provide a basic form of resource that is essentially counters: processes put elements in and may (then) take them out. They have no in-built notion of composition or comparison beyond numerical ordering. Demos2k's resources provide stocks of elements for which concurrent processes (called entities) may compete. Just as for bins, resources also have no in-built notion of composition or comparison.

Process: The basic notion of process in Demos2k is the entity. Entities may be thought of as (recursively defined) sequences of actions (cf. the basic (recurisve) construct $a : E$ in the definition of process terms in § 3.3). Entities manipulate resources.

Location: There is no inherent notion of location in Demos2k. Location must be represented implicitly in models. In particular, the model discussed in § 4.2 represents different locations implicitly. In more complex models, the complexity introduced in this way would become difficult to manage.

Environment: Demos2k supports a very wide range of probability distributions for representing the behaviour of the environment and, indeed, stochastic aspects of the internal construction and operation of models.

A process-theoretic semantics for Demos2k has been provided in [5–8], where theorems characterizing precisely the interpretation of Demos2k's constructs in mathematical theories of processes are given. Formally, the semantics requires a stochastic extension of process algebra, such as that given in [33], although the structural and stochastic parts of the semantics can be separated. Similar analyses, exploiting SCRP's built-in resource semantics, can be given in our setting (see [27] for a discussion).

4 Network Operations, Security Operations, and Systems Value

Network security operations are activities engaged in by network managers in order to protect the availability, confidentiality, and integrity of networks, including the constituent computer systems, and the information that they process. From a certain perspective, availability may be seen as the primary concern: if systems and the services they deliver are non-functional, questions of confidentiality and integrity do not arise. In the remainder of this paper we are concerned with availability. The availability of the network itself limits the availability of all items on the network.

Network security operations require investments in staff, devices, and software. The value of the availability of a network can be regarded as being determined by the cost of lack of availability as specified by, say, a Service Level Agreement (SLA); that is, the supplier of a network security service will be required to pay the customer (i.e., the user of the network) specified levels of compensation for specified levels of lack of availability. In this paper, then, we study how network and security operations impact upon system availability and we assume that *value* of the availability of a system is adequately represented by the terms of (penalties for poor performance, etc.) the SLA associated with the operation of the network. Availability, therefore, is a proxy for value in this work. Note that SLAs are applicable not only in the context of contracts *between* organizations (such as in IT outsourcing) but also *within* organizations. We suggest, therefore, that SLAs provide a both convenient and effective measure of value for our present purposes.

In order to provide an example context within which to develop our analytic method, we have chosen a model of a (hypothetical) system with 20,000 devices which equates to a high-end small- to medium-sized enterprise (SME) or institution similar in size to a medium-sized European university. Given that the parameters in the model have not been calibrated, we have chosen a 'reasonable' starting set and then have first, in a basic threat model introduced in § 4.1 explored the number of security operations staff required to give achievable utilization rates when exploring increased threat and reduced numbers of operational staff. We then, in § 4.2, broaden and extend this model by explicitly introducing aspects of network management related to security operations.

4.1 Sources of Costs

Sources of downtime, or lack of availability, of an ICT system can be characterized as follows [36]:

1. Fix time: the time taken to repair specific components of the system that have been compromised or damaged by attack. Typically this would be cleaning or rebuilding a computer including the time taken to recover data. This source of downtime arises from vulnerabilities;
2. Misconfiguration: lack of access to a significant business component of the system (e.g., a business application, such as an enterprise relationship management system). By misconfiguration we mean any configuration of the system incompatible with operation of business processes (e.g., authentication failure, incorrect assignment of rôle) which would require security operations resources to resolve. Also note the principle of constant change, there is never a correct system configuration. This source of downtime arises from lack of alignment;

3. Intrinsic reliability: the intrinsic reliability of the components that make up the system, the way in which they are connected; and their dependency on operational level agreements in place for break/fix constitute the usual domain of availability analysis and typically the focus of SLAs in current IT Outsourcing (ITO) and application management contracts;
4. Network: a catch all for lack of availability caused by the network itself being swamped by non authorized, non-business-related traffic, perhaps due to a Denial of Service (DoS) or worm attack. This source of downtime arises from vulnerabilities.

We make an essential distinction between downtime arising from vulnerabilities (1, 4) and alignment (2). Typical analyses of threat environments and attack profiles have tended to concentrate on the former. Both, however, must be monetized for any derived SLA to be considered meaningful.

4.2 Classification of Security Controls

Security controls can be divided into a number of categories, according to the job they do, the threats they protect against, and the data or protocol levels they examine (all of which are related). These controls can be used to provide various forms of defensive measure for a particular network, with the defensive effect they have depending upon their *configuration* and their *network location*. Most of these controls can be implemented either on end hosts (such as individual computers) — host-based — or on network equipment (on either dedicated devices or as enhancements of other devices, such as routers) — network-based.

We give below a summary of the categories of network security controls whose behaviour we wish to capture in our models:

Firewalls: These can be either host- or network-based. They provide filtering capabilites to restrict network access to certain categories of traffic on certain 'ports';
IDS devices: Intrusion detection systems monitor the network, using a signature to identify undesirable traffic;
IPS devices: Intrusion protection systems protect regions of the network — that is, identified collections of locations — from undesirable traffic identified by IDS devices by taking appropriate inhibiting actions (typically by blocking the identified traffic).

Other network security controls, that are beyond the scope of our present work, include application-level gateways (such as web proxies), anti-virus tools (e.g., active scanning of computer memory for malware), scanners (probe devices attached to the network to search for vulnerability to known attacks), and so-called 'pukkaware' (particular bundles of hardware and software, chosen and configured to meet particular policy standards).

Security controls devices have, essentially, fixed unit acquisition costs and their use incurs installation, configuration, and operational (including servicing) costs. The security controls placement problem, then, is to determine the optimal location for placing a limited collection of such devices on a network in order to achieve the desired protection. Some discussion, in the absence of any consideration of the structure of systems, has been given by, for example, [13, 11]. In making such an assessment, the following factors, among others, must be considered:

- The location of the endpoints of the system at which exposures to threats can occur;
- Which are the locations on the network that are critical to organizational value owing, for example, to the nature of their information throughput;
- The effects of necessary reconfigurations of the system, in response to threats and to realign with changing organizational objectives.

Building on our basic security operations model [36], in this paper, we focus on the last of these three concerns.

5 A Description of the Demos2k Models

We now turn to a more detailed discussion of our Network and Security Operations model as implemented in Demos2k. This will follow our standard conceptual pattern of *resource*, *process*, *location*, and *environment* that we introduced in § 3.2. We will further split this into an initial discussion of the basic model (see [36]), followed by a discussion of our extended operations model introducing elements of network management.

We illustrate our description with fragments of the Demos2K code for the model.

5.1 The Basic Security Operations Model

Broadly, our model captures the *high-level* operations and activities performed by the IT division of a large organization, such as a medium-sized European university, to defending its IT assets (hardware, software and information) from external attacks exploiting vulnerabilities in software deployed on the endpoint devices (e.g., computers, desktops, etc.). The parameters for the Demos2k model corresponding to Figure 5 are given in Appendix B.1, structured in accordance with the Demos2k principles presented in Appendix A.

The diagrams given in Figures 5 and 3: both represent a high-level process view of causative spawning dependencies in their respective models. The thick arrows represent event flows between processes and queues. Processes decorated with circular arrows indicate 'perpetual' processes that continually execute, consuming resources.

Resource: The principal resource of interest is security operations staff. Staffing levels is of interest because of the interaction of regular activities (such as regular patching and signature distribution) with more sporadic events (such as emergency patching and crisis handling).

Broadly, there are two kinds of staff resource represented in our model:

- *General operations staff:* These are systems security experts and administrators whose responsibility is to oversee security-related business processes such as systems patching, 3rd-party signature deployment, and secure repair. We envisage that such processes are largely automated but nonetheless could require *manual intervention* involving *human initiative* to handle the difficult corner cases that may arise from time to time. Thus, despite extensive automation, expert judgement will be required on occasion to resolve issues and ensure adequate completion of tasks;
- *Analyst operations staff:* These are security strategy experts whose responsibility is to assess incoming threat intelligence to coordinate and assess security response to those threats. In particular, analyst staff must be knowledgeable about the current IT assets in the organization in order to decide the level of threat posed by particular vulnerabilities and to decide what to do about this. Typically this amounts to filtering the patches to be applied and thus expediting relevant patches to ensure rapid application (prior to attack). Given the number and variety of systems available, it is important to filter out the many patches that are not relevant to the organization and to avoid wasting valuable time deploying them.

Note that we are also exploiting the *full-time equivalent* staffing model to abstract away neatly from representing various personnel management issues such as staffing rotas, holidays, absences, etc..

Process: The following internal *activities* consume security operations staff resources:

- *Business alignment:* Continuous change management (relationship between users, resources, and processes) which has been aggregated into a single *business alignment* services process;
- *Secure repair:* Repair/reinstatement of compromised or damaged machines;
- *Patch management:* including testing and deployment;

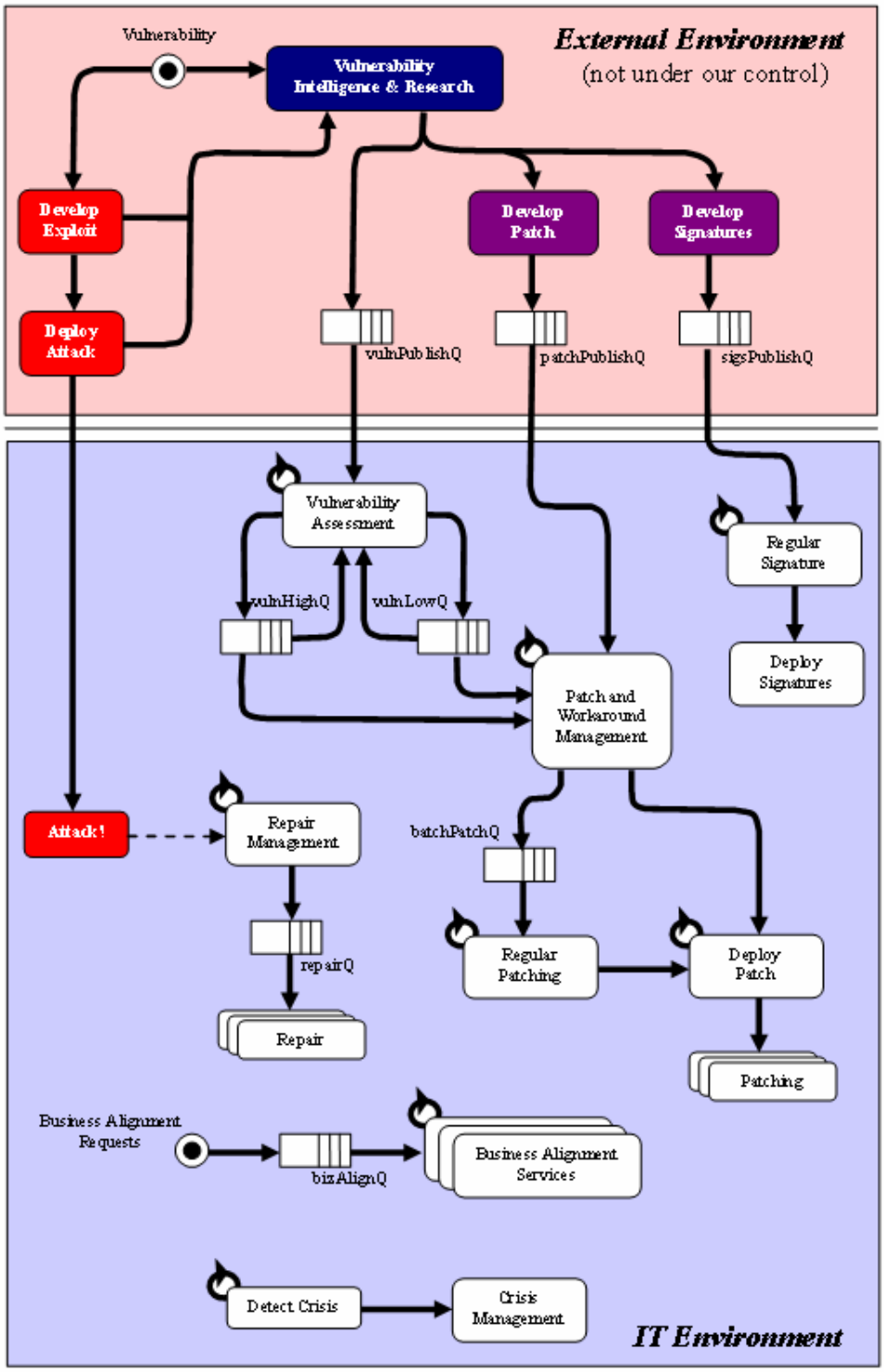


Fig. 1. Basic operations model [36]: High-level process flow diagram

- *Workarounds:* Implementing ad hoc 'workarounds' to provide further mitigating actions as necessary;
- *3rd-party signature deployment:* Deployment of 3rd-party signatures that are used when detecting anomalous behaviour. As such, these signatures have to be provided and renewed on a regular basis if they are to remain effective.
- *Vulnerability assessment and threat analysis:* Determining corporate patching deadlines and associated risk levels.

We illustrate how these processes are represented by discussing the `repair` process in terms of its formal description in Demos2k; Figure 2 presents the Demos2k code.

The `repair` process consists of a perpetual process consuming repair items from the queue `repairQ`. Staff are then allocated and the repair performed taking time, for example, `hold(repairTime)`, where `repairTime` is a stochastic quantity drawn from a normal distribution. The repaired system is then rebooted. Finally, various counts such as `availSys` are updated to reflect that a system has been repaired.

Location: The basic operations model pays little attention to location. Rather, it is concerned essentially with the incidence of threats upon the system as a whole and the response of the system as whole. As such, the basic model concentrates on modelling security management activities in terms of aggregated quantities such as `availSys`, effectively assuming that spatial distributional aspects have been safely incorporated. However, in our extended model, we explicitly introduce aspects of network management and thus begin to acknowledge the rôle that networks and hence location plays in security management.

Environment: A model for security management would be vacuous without some source of threat and challenge. Accordingly, we introduce processes representing the development of exploits based upon vulnerabilities in systems software. To counterbalance this, there are vulnerability intelligence activities underway as well as 3rd-party organizations developing patches and various kinds of signatures. All of these activities are external to the large organization in question and are thus beyond its direct control.

We have made the following simplifying assumptions about vulnerabilities, exploits and the effect of patching:

- Vulnerabilities are naturally present in all systems — they do not need to be 'caught' or 'infected';
- There is effectively an arbitrarily large number of vulnerabilities — we assume an endless supply (valid because non-security updates always occur);
- Patching can never eliminate all *future* vulnerabilities — only the ones that are currently known about by patch developers;
- Viruses, worms, and trojans (VWT) exploit vulnerabilities that remain on currently unpatched systems. These VWT are the infection agents;
- Typically, there has to be some external access (e.g., via email or internet access) for VWT to become active;
- The process of patching systems will eliminate certain vulnerabilities (hopefully prior to the attack) and also repair the effects of the infective agents following an attack. This assumption represents a substantial simplification: typically, the process will both failire to find all of the vulnerable devices and, at least potentially, introduce new vulnerabilities.

We acknowledge that this is a radical simplification of the known threat space. For example, the vulnerability intelligence capability could be split into, say, predictive and responsive (e.g., for 0-day attacks) operations — but the present model suffices for our present purposes.

```
...

cons NsecOps    = 4;       // number of security operations staff
cons Nanalysts  = 2;       // number of security and network analysts

...

cons repairTime = normal(3*hrs, 10*mins);

...

res(lock,  1);
res(opsStaff,  NsecOps);

...

bin(repairQ,          0);

...

class repair = {
  local var repairDelaytime = 0;
  local var staffNeeded = 0;

  repeat{
    req [ online == 1 ];
    getB(repairQ, 1);

    staffNeeded := staffForRepair;
    getR(opsStaff, staffNeeded);
      hold(repairTime);
      hold(rebootTime);
    putR(opsStaff, staffNeeded);

    getR(lock, 1);
      cRepaired := cRepaired + 1;
      availSys  := availSys + 1;
      vulnerableSystems := vulnerableSystems - 1;
      sync(check_range);
    putR(lock, 1);
  }
}

...

do repairStreams { entity(repair, repair, 0); }     // Launch repair teams ...
```

Fig. 2. Repair process class in Demos2k

5.2 The Network & Security Operations Model

We assumed in our previous model that networks behave in a transparent and neutral manner with respect to security matters. The security actions taken into account there focused upon the devices at the endpoints — these actions typically involved systems patching and the deployment of 3rd-party signatures. We now broaden and extend our earlier model by explicitly introducing aspects of network management related to security operations.

Specifically, we add into our earlier model the activities involved with reconfiguration of the network and model its impact upon the business. Network reconfiguration is concerned with changes in network connectivity between internal clusters of endpoint devices (e.g., computers, servers, desktops).

Networks may be reconfigured for several reasons:

Business alignment: As a part of some persistent realignment of business units and their processes, the network may need to be reconfigured in order to provide additional access to some services or to restrict access to others;

Security isolation: Temporary isolation measures to protect certain software and information assets, perhaps from damage owing to an anticipated or ongoing networked attack. Such actions will typically partition the network in ways that are disadvantageous to some business activities but which protect the integrity of the overall network. Clearly, such measures are expensive in terms of reduced business activity (i.e., availability) and as such are regarded as extreme measures, taken as a *least undesirable* option. Because these measures are temporary, there will be *two* periods of network disruption: firstly, to enforce the isolation; and, secondly, to deliver it into its desired configuration;

Optimizing and reprioritizing access: Reconfiguring access in response to prevailing conditions, perhaps as a response to some attack.

As such, there is inevitably some degree of short-term disruption to business users due to changes in connectivity. Thus the survival security benefit due to a partitioning of the network has to be balanced against the impact of the short-term disruption it causes.

As we have already seen in § 4, the security of the network itself involves placement of a number of network security controls (e.g., firewalls) to both monitor network activity and also to protect it. Any reconfiguring of the network raises the question of how to position network security controls to perform these functions in an effective manner.

To be clear, we again follow our standard conceptual pattern of *resource*, *process*, *location*, and *environment* used earlier. The parameters for the Demos2k model corresponding to Figure 3 are given in Appendix B.2, structured as before in accordance with the principles from Appendix A.

Resource: As in our earlier model, the resources of interest are the operations staff, to which we now add:

- *Network operations staff:* These are network security expert engineers and administrators whose responsibility is to perform network reconfiguration operations, which may be significantly disruptive activities. In particular, network security controls may need to be reconfigured and repositioned within the network. This will hopefully be done virtually, but could potentially involve physical relocation of equipment.

We further assume that the existing pool of analyst staff takes on the strategic activities of planning the network operations activities, rather than to introduce a separate pool of staff. From a modelling perspective, this just means we can increase the size of the analyst pool to allow for this.

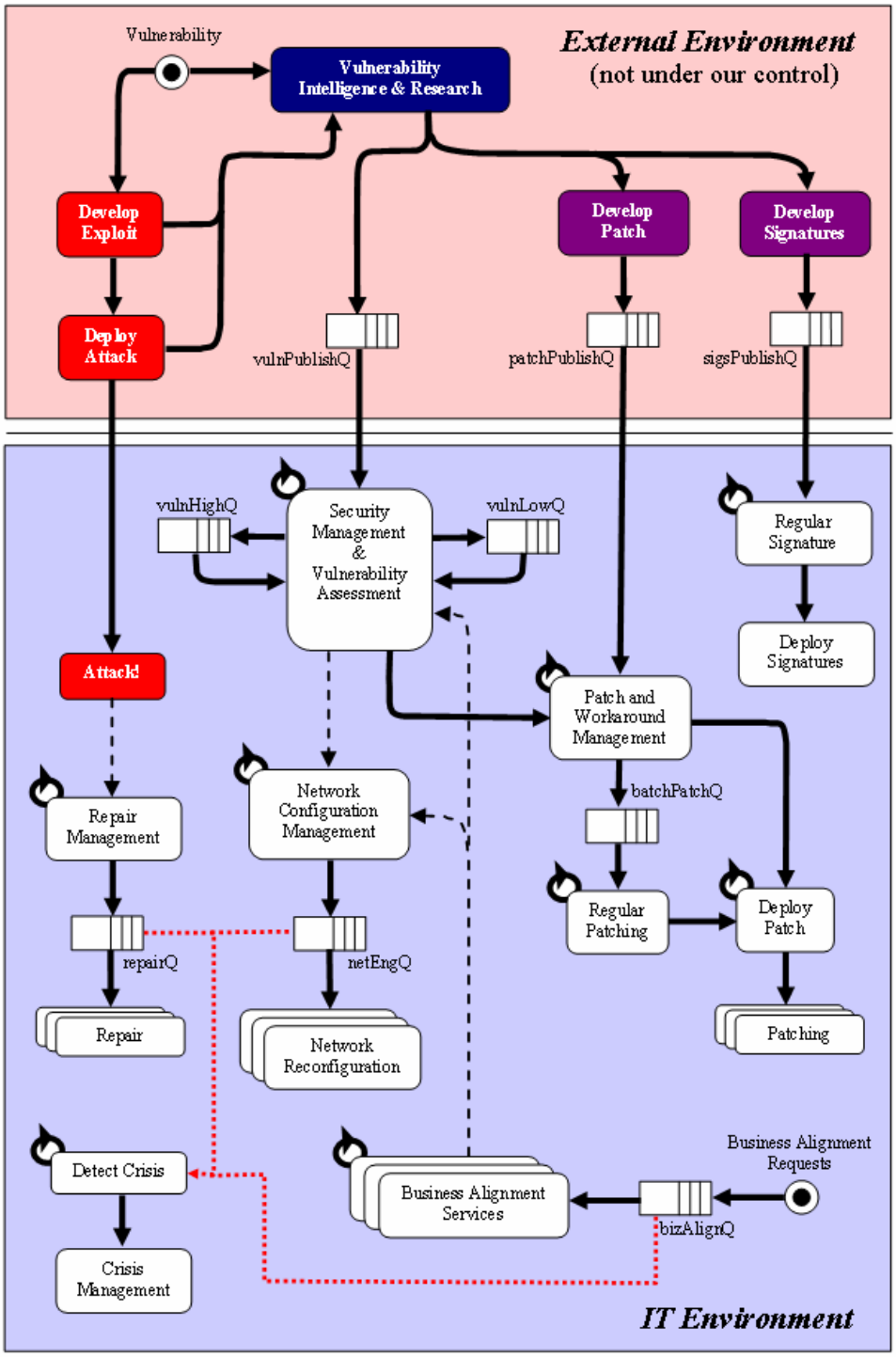


Fig. 3. Extended model (incl. network management): High-level process flow diagram

```
class networkManagement = {
  repeat{
    req [ online == 1 ];

    req [ networkTasks > 0 ];  // there are network tasks to be done ...

    getR(lock, 1);
      networkTasks := networkTasks - 1;
    putR(lock, 1);

    getR(analysts, 1);
      hold(networkOpnsPlanning);
      putB(netOpnsQ, 1);
    putR(analysts, 1);
  }
}


class networkOperations = {
  local var networkDelayTime   = 0;
  local var netEfficencyFactor = 0;

  repeat {
    req [ online == 1 ];

    getB(netOpnsQ, 1);

    syncV(calc_network_delay, [], [networkDelayTime]);
    hold(networkDelayTime);

    getR(netStaff, staffForNetworkOpn);
       hold(networkOperationTime);
    putR(netStaff, staffForNetworkOpn);

    getR(lock, 1);
      cNetworkOpns       := cNetworkOpns+1;
      netEfficencyFactor := (1 - networkEfficiencyIncr);
      networkEfficiency  := netEfficencyFactor * networkEfficiency;
      sync(check_range);
    putR(lock, 1);

    entity(networkRestore, networkRestore(#netEfficencyFactor),
           networkDisruptionTime);
  }
}

class networkRestore(factor) = {
    getR(lock, 1);
      networkEfficiency := networkEfficiency / factor;
      sync(check_range);
    putR(lock, 1);
}
```

Fig. 4. Networking management classes in Demos2k

Process: The new model introduces aspects of network management to handle and deal with network reconfiguration tasks as described above. This model is a fairly direct extension of the previous model; this will involve introducing some additional process elements, together with a number of aspects that cut across some of the existing process elements. We shall focus here on the additions required and their overall structure and shape. The diagram in Figure 3 presents a high-level process view of the extended model.

Clearly we cannot represent the network in a direct or literal way; so we instead need to capture its *effects* in terms of the *coupling* with the activities we have already modelled. This choice is a very clear example of our need to choose levels of abstraction for our models that are appropriate for the questions of interest.

The insight here is to introduce two numerical variables whose values will impact overall systemic behaviour in our model:

- `networkEfficiency`: This numerical variable lies in the range strictly between 0 and 1, and represents a 'frictional' effect acting upon all IT operations. The way this manifests itself is through the introduction of longer delays into a number of the existing activities e.g., repair, patching, etc..

  Thus, a lower network efficiency means a slower, less responsive network having greater delay. In business terms, this translates to a reduced level of business performance. A higher network efficiency means lower delays and thus, potentially higher business performance. To be clear, network efficiency is only one of several factors that would limit business performance in practice and certainly cannot by itself *determine* it.

  We use network efficiency in the model to capture the effects of network disruption due to network reconfigurations. As such, this is a dynamic quantity that can increase and decrease as a result of these activities. For simplicity, we assume that the effect of network disruption is transient and can thus recover after some amount of time.

- `securityControlEffectiveness`: This numerical variable lies in the range strictly between 0 and 1 and represents the overall, aggregated effectiveness of all the security controls deployed within the network. The way this manifests itself is by qualifying the 'potential attack impact' upon the endpoint devices.

  What in practice influences and determines the effectiveness of security controls? There are simply too many details to consider in depth here so we shall radically simplify the issue to a consideration of the following two major aspects:

  - Strategic placement within the network to monitor, filter and control traffic, in accordance with corporate security policies. We reflect the *quality* of such placement in terms of the *maximum value* that the security control effectiveness attains.
  - Keeping up to date with the prevailing threat situation, in terms of 3rd-party detection and filtration signatures. We model this aspect by allowing the security control effectiveness to *decay* at a daily rate over time as a result of the increasing staleness of the 3rd-party signature information. Naturally, whenever the 3rd-party signatures are renewed and deployed, we can restore the security control effectiveness variable back to its maximum value.

The following Demos2k code fragment is added into a number of model processes whose activities would be sensitive to network delay e.g., repair, patching, etc..

```
...
syncV(calc_network_delay, [], [networkDelayTime]);
hold(networkDelayTime);
...
```

where the class `calcNetworkDelay` represents a convenience function to calculate the network delay time based upon the current value of `networkEfficiency`.

```
class calcNetworkDelay = {
  local var delay = 0;
  repeat{
    getSV(calc_network_delay, [], true);

    getR(lock, 1);
       delay := networkDelay*(1/networkEfficiency - 1);
    putR(lock, 1);

    putSV(calc_network_delay, [delay]);
  }
}
```

The formula $\texttt{networkDelay} * (1/\texttt{networkEfficiency} - 1)$ is used to specify the delay because as the networkEfficiency decreases towards zero, the delay effect increases unboundedly. In the other direction, as the networkefficiency tends towards 1, the delay becomes zero i.e. as the network efficiency improves, the delay becomes shorter. In particular, note that the constant `networkDelay` is equal to the delay value when the network efficiency is $1/2$.

We now briefly discuss the Demos2k code used to represent the network management processing, presented in Figure 4. The perpetual class `networkManagement` repeatedly waits for a network reconfiguration request (i.e., when $\texttt{networkTasks} > 0$ to enter the planning stage for network operations. This consumes some analyst time followed by placing a planned job on the queue `netOpnsQ`.

The perpetual class `networkOperations` represents the activity of a network operations workforce team responding to network operations tasks from the queue `netOpnsQ`. Note that the effect of network delay also affects `networkOperations` itself. We model the work being done here by allocating network operations staff and then holding for a certain amount of time specified by the stochastic constant `networkOperationTime`. Following this, various counts and values are computed and updated. Finally, an instance of `networkRestore` class is launched which waits a randomised period of time determined by `networkDisruptionTime` before restoring the network efficiency.

A subtlety here is that `networkRestore` doesn't merely reset back to the prior value of `networkEfficiency` but instead performs an inverse operation (namely division) to the *present value* of `networkEfficiency` at the time that it eventually executes. Since there may be other such restoration processes with other randomised times, this randomises the time and the order in which `networEfficiency` is restored. Because of this, it would be quite wrong to simply update with the previous value of `networkEfficiency`. By performing an inverse operation we can guarantee that the newly updated value is functionally linked to its immediately previous value. Because the division operations can be reordered, we know that the intermediate values of `networkEfficiency` always remains less than 1 and, more importantly, will return to its original value.

Location: The inclusion of network reconfiguration into our model clearly brings with it some concern for location-dependent issues. As shown above, the effect of using and depending upon a network is encoded in terms of the values of a couple of numerical variables. Such a model is useful and adequate for performance related matters in which the systems behaviour and characteristics are typically represented in terms of aggregated stochastically determined quantities.

However, there are spatial distribution questions involving location that cannot be answered purely in this fashion - but which nonetheless have a significant impact upon the economic effectiveness and efficiency of the network. One such issue concerns the effective placement of network security controls within the network.

The problem is that network security controls need to be placed so as to maximize their impact in reducing potentially harmful traffic and in controlling security access to different regions of the network. Now, reconfiguring the network involves changing access permissions and the major routing

taken through the network. Both of these impacts will affect how the network security controls are themselves configured and additionally, where they should be logically positioned in the network.

Furthermore, placing these network security controls effectively will require some appreciation of where the business critical assets and traffic are. Thus, in some sense, it is necessary to prioritise the placement of network security controls somehow according to the greatest need for network defense and protection.

Once some estimate of this asset information is known, standard graph-theoretical techniques (from operations research) might be applicable to assist in locating where best to place the controls on the network to realize security protection value to business IT assets. The downside of not placing network controls in line with business-asset value is that the wrong assets will get protected, leading to poor value for money spent.

Environment: The environmental considerations for the earlier model apply equally for this model.

6 Experimental Results

We have used our model of network and security operations to conduct a number of simulations. The model is generic; that is, it is not based on any specific organization, although quite clearly based on the processes and resources one would expect to find in any organization of sufficient size to require them. The value of developing the model is to make explicit certain relationships and dependencies between internal IT functions (e.g., repairs, patching) and external event drivers (e.g., exploits). The simulations performed are purely illustrative of the kind of effects and results that would be observed in a modelling activity based on a specific organization tasked with providing data into decision making. Thus the results obtained from the simulations presented here constitute *conjectures* or *open questions* relating to how a particular organisation *might* behave in a certain situation; that is, the results have been generated in isolation, rather than as part of an operational context designed to elicit data for hypothetical analyses that feed into a decision making process.

The principal effect that we demonstrate in our model is the propensity for 'crisis' events to arise owing to the IT systems of large organizations becoming overloaded by a backlog of work to be performed — in our model this is represented by the growths of the repair and business alignment queues. When, in executions of our model, these queues become too long and the general availability of the organization suffers, a crisis situation is deemed to have arisen. At this point, the general operation of the organization is suspended and a clean-up activity takes place — in our case, we have assumed this to take just two days. The occurrence of such crisis events was observed in both models — and is illustrated graphically in Figure 5.

Crises occur because IT service organizations could become overstretched (perhaps because of a rapid sequence of attacks) and hence incapable of fulfilling their basic service obligations. The event precipitating a crisis need not by itself be large in magnitude or scale — it could easily be a small event which leads to the crisis itself, although it is also typically preceded by a gradual reduction of availability up to that point.

6.1 The Basic Security Operations Model

In this brief section, we summarize the results obtained in our previous work (see [36]). In that paper, we are mainly concerned about availability SLAs and the effect of security upon them. We have assumed that, should an organization decide to use an availability SLA, then loss of availability can be adequately monetized — thus linking availability to business value.

The model has been used to investigate two key questions: firstly, the impact, on predicted availability of a system, of reducing the number of security operations staff resources; and, secondly, the impact on availability and utilization of changes in the threat environment. We have chosen a system with 20,000 devices which equates to a high end small- to medium-sized enterprise (SME) or institution similar in size

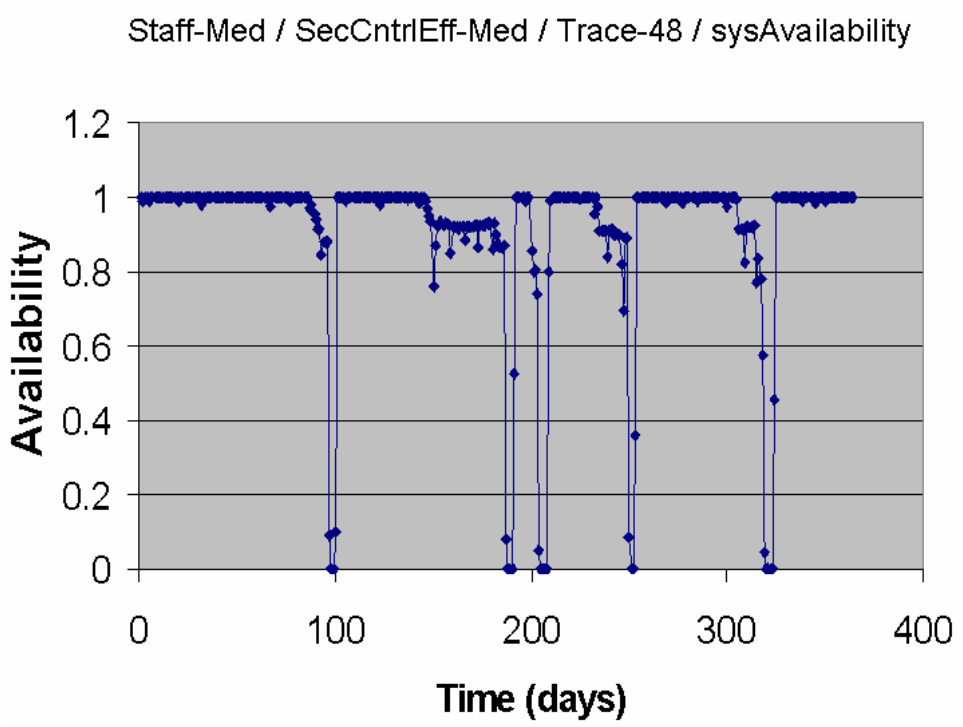


Fig. 5. Graph of systems availability showing 5 crisis dips

to a medium-sized European university. Given that the parameters in the model have not been calibrated we have choosen a 'reasonable' starting set and have explored the number of security operations staff that gave achievable rates of utilization of staff when exploring increased threat and reduced numbers of operational staff. This gives a baseline model in which 3 security operations FTE staff are required, assuming single-site operation. In order to acquire reasonable statistics, each simulation run of 365 days is repeated 100 times. Using the model as shown in Appendix B as a baseline we obtain an average predicted availability of 98.15%. The corresponding average utilization of the security operations staff is 57.45%.

The broad outcome that can be observed is that the number of crises arising (captured in availability terms) is inversely proportional to the numbers of FTE security staff available, but also proportional to the vulnerability discovery rate (i.e., number of vulnerabilities discovered per unit time). Some further details concerning these (indicative) results may be found in § 5 of [36].

6.2 The Network and Security Operations Model

We now briefly investigate, using our enhanced model, how the number of crises per year changes when both the maximum value of security control effectiveness and the numbers of network and security operations staff available vary. Recall that the *maximum value* of the *security control effectiveness* parameter is indicative of how well the overall security controls are placed, maintained, and operated within the organization. This maximum value is used to initialize and periodically refresh the security control effectiveness parameter over time.

As with the previous study discussed above, our simulation runtime is taken to be 1 year (i.e., 365 days) and for each set of parameters (e.g., maximum value of security control effectiveness, numbers of network staff), we again performed 100 independent simulation runs (i.e., a simulated century) for the purpose of gathering acceptable statistics. Naturally all other variables are held constant. In particular, our organization has 20000 systems under management and the rate at which vulnerabilities are discovered is 36 per 365 days — approximately 1 every 10 days.

Our synthetic experiment has two main sets of parameters: the maximum value of security control effectiveness; and the numbers of network and security operations staff available. To explore the effects of varying these parameters, we shall cluster each of these parameters as follows:

*Maximum value of security control effectiveness **[Max SCE]***:

LOW	**MEDIUM**	**HIGH**
0.05	0.5	0.95

*Numbers of Network and Security Operations Staff **[N. & S. Staff]***:

LOW	**MEDIUM**	**HIGH**
2 network, 2 security	4 network, 4 security	6 network, 6 security

Thus, there are 9 different combinations of results, presented in Table 1. The outcomes are presented in terms of the most likely number of crises, and its probability, together with the probabilities of there being fewer/more crises respectively.

N. & S. Staff	Max SCE	Most likely no. of crises (prob.)	Prob. of fewer crises	Prob. of more crises
2N, 2S (LOW)	0.05 (LOW)	5 (0.23)	0.16	0.61
	0.5 (MEDIUM)	4 (0.22)	0.15	0.63
	0.95 (HIGH)	2 (0.43)	0.36	0.21
4N, 4S (MEDIUM)	0.05 (LOW)	5 (0.21)	0.22	0.57
	0.5 (MEDIUM)	5 (0.21)	0.42	0.37
	0.95 (HIGH)	1 (0.47)	0.14	0.39
6N, 6S (HIGH)	0.05 (LOW)	3 (0.27)	0.08	0.65
	0.5 (MEDIUM)	4 (0.29)	0.39	0.32
	0.95 (HIGH)	1 (0.48)	0.20	0.32

Table 1. Number of crises vs max. security effectiveness and staffing levels

The clear message from these results is that it is clearly important to maximize the effectiveness of security controls, even though doing so does not eliminate the risk of crises. This table also shows that it is useful to employ sufficiently many of the right people. Specifically, for example:

- The difference between the first and second rows of the third column represents a shift of relative impact from being against high SCE to being against medium and low SCE; and
- The difference between the second and third rows of the third column represents an overall reduction in the number of crises (as a result of the larger staff numbers).

7 Discussion

We have shown how a mathematically rigorous conceptual modelling framework, partially captured in executable form by the Demos2k modelling tool, can be used to represent and explore the levels of investment in, and trade-offs between, operations staff and security control devices (such as firewalls, IDSs, and IPSs) by analyzing the consequences of such decisions for system availability. We have assumed that the underlying value of system availability is captured by a Service Level Agreement.

Our experimental work has given good exemplary data, suggesting strongly that our approach is valuable, and has raised some interesting questions:

- What are effective methods for designing experiments with which to explore complex models of the kind we construct?
- Are there effective heuristics for determining the appropriate level of abstraction for a model? How are we to resolve reliably the tension between our desire — indeed, policy — to avoid representing detail and the level of detail sometimes necessary in the security domain?
- How can we effectively integrate models of spatially distributed networks, stochastic environmental behaviour, and system value?

We have demonstrated a generic approach using an illustrative model. In practical application, the modelling activity would take place in response to a specific requirement from an organization's CIO or CISO to provide analyses of business decisions about information security investments. For example, if we take the simulation results presented in Figure 5 and Table 1 and assume that they were generated in response to a previous crises similar to the sequence shown, then a reasonable conclusion might be that the organization must ensure that Max SCE is always high but no more than 4N and 4S staff are required because the most likely number of crises in a year will always be close to 1. If this is unacceptable, however, a decision would perhaps be to redesign some of the operational processes, in which case the model would need to be changed and a similar set of simulations run. An excellent operational state, the embodiment of systems analytics, would then be the continual checking between predicted behaviour and actual with updating of the model as necessary.

More broadly, two other questions, at least, remain:

- Can we integrate our work, focussed on system/network value, with models of, for example, markets in vulnerabilites and security operations services?
- Can we use our methods to assess the value of investments intended to protect not only availability but properties such as confidentiality and integrity?

References

1. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.
2. R. Anderson. Why information security is hard: An economic perspective. In *Proc. 17th Annual Computer Security Applications Conference*, 2001.
3. R. Anderson and T. Moore. The economics of information security. *Science*, 314:610–613, 2006. Extended version available at `http://www.cl.cam.ac.uk/~rja14/Papers/toulouse-summary.pdf`.
4. G. Birtwistle. *Demos — discrete event modelling on Simula*. Macmillan, 1979.
5. G. Birtwistle and C. Tofts. An operational semantics of process-orientated simulation languages: Part I πDemos. *Transactions of the Society for Computer Simulation*, 10(4):299–333, 1993.
6. G. Birtwistle and C. Tofts. An operational semantics of process-orientated simulation languages: Part II μDemos. *Transactions of the Society for Computer Simulation*, 11(4):303–336, 1994.
7. G. Birtwistle and C. Tofts. Getting Demos Models Right — Part I Practice. *Simulation Practice and Theory*, 8(6-7):377–393, 2001.
8. G. Birtwistle and C. Tofts. Getting Demos Models Right — Part II ... and Theory. *Simulation Practice and Theory*, 8(6-7):395–414, 2001.
9. L. Jean Camp and Stephen Lewis, editors. *Economics of Information Security*. Kluwer Academic Publishers, 2004.
10. L. Jean Camp and Catherine Wolfram. Pricing Security. In L. Jean Camp and Stephen Lewis, editors, *Information Security Economics*, pages 17–34. Kluwer Academic Publishers, 2004.
11. Huseyin Cavusoglu. Economics of IT Security Management. In L. Jean Camp and Stephen Lewis, editors, *Information Security Economics*, pages 71–83. Kluwer Academic Publishers, 2004.
12. A. Christodolou, R. Taylor, and C. Tofts. Demos2000. `www.demos2k.org`.

13. W. Lee et al. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1–2):5–22, 2002.
14. D. Galmiche, D. Méry, and D. Pym. The Semantics of **BI** and Resource Tableaux. *Mathematical Structures in Computer Science*, 15:1033–1088, 2005.
15. L.A. Gordon and M.P. Loeb. The Economics of Information Security Investment. *ACM Transactions on Information ad Systems Security*, 5(4):438–457, 2002.
16. L.A. Gordon and M.P. Loeb. The Economics of Information Security Investment. In L. Jean Camp and Stephen Lewis, editors, *Information Security Economics*, pages 105–127. Kluwer Academic Publishers, 2004.
17. L.A. Gordon and M.P. Loeb. *Managing Cybersecurity Resources: A Cost-Benefit Analysis*. McGraw Hill, 2006.
18. S. Gritzalis, A.N. Yannacopoulos, C. Lambrinoudakis, P. Hatzopoulos, and S.K. Katsikas. A probabilistic model for optimal insurance contracts against security risks and privacy violation in it outsourcing environments. In press, 2006.
19. 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.
20. Robert S. Kaplan and W. Bruns. *Accounting and Management: A Field Study Perspective*. Harvard Business School Press, 1987.
21. Carl Landwehr. Improving Information Flow in the Information Security Market. In L. Jean Camp and Stephen Lewis, editors, *Information Security Economics*, pages 155–163. Kluwer Academic Publishers, 2004.
22. R. Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25(3):267–310, 1983.
23. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
24. R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.
25. P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.
26. David Pym and Chris Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18(4):495–517, 2006. Erratum for this paper and [27] submitted to *Formal Aspects of Computing*.
27. David Pym and Chris Tofts. Systems Modelling via Resources and Processes: Philosophy, Calculus, Semantics, and Logic. In L. Cardelli, M. Fiore, and G. Winskel, editors, *Electronic Notes in Theoretical Computer Science (Plotkin Festschrift)*, volume 172, pages 545–587, 2007. Erratum for this paper and [26] submitted to *Formal Aspects of Computing*.
28. D.J. Pym. On bunched predicate logic. In *Proc. LICS'99*, pages 183–192. IEEE Computer Society Press, 1999.
29. 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`.
30. J.C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *Proc. LICS '02*, pages 55–74. IEEE Computer Society Press, 2002.
31. S. Schechter. *Computer Security Strength and Risk: A Quantitative Approach*. PhD thesis, Harvard University, 2004.
32. Colin Stirling. *Modal and Temporal Properties of Processes*. Springer Verlag, 2001.
33. C. Tofts. Processes with probability, priority and time. *Formal Aspects of Computing*, 6(5):536–564, 1994.
34. C. Tofts. Efficiently modelling resource in a process algebra. Technical Report HPL-2003-181, HP Laboratories, Bristol, 2003.
35. Hal Varian. System Reliability and Free Riding. In L. Jean Camp and Stephen Lewis, editors, *Information Security Economics*, pages 1–15. Kluwer Academic Publishers, 2004.
36. M. Yearworth, B. Monahan, and D. Pym. Predictive modelling for security operations economics (extended abstract). In *Proc. I3P Workshop on the Economics of Securing the Information Infrastructure*, 2006. Proceedings at `http://wesii.econinfosec.org/workshop/`.

A A Brief Summary of Demos2k

Demos2k is two things — firstly, it is a semantically justified discrete-event systems modelling language (see § 3); and, secondly, it is a simulation-based environment to support the examination and exploration of systems so described.

An advantage of this semantic justification activity is that, having done this work, it contributes extensively to the integrity of the implementation, thus ensuring accuracy and fidelity of the simulation results obtained using the tools. Broadly speaking, this means that Demos2k users can be highly confident in the numbers produced during and resulting from their simulations and in the patterns of behaviour observed.

With this underpinning, we can be much more certain that our results are genuine consequences of the model and not mere artefacts of simulation.

The Demos2k environment has been designed to support the precise examination of simulation oriented descriptions of systems. These can be compiled or automatically rewritten into multiple representations dependent upon the questions that must be asked of the model such as correctness, performance, availability, or agility, etc..

Systems descriptions written in Demos2k tend to be high-level, pleasingly short and to the point. The modelling philosophy supported is very much akin to 'extreme modelling', where the systems analyst/modeller can rapidly construct high-level models representing the customer's core business concerns. A key contribution to this capability is the exploitation of probability theory to abstract away from extraneous details.

We now outline the basic 'shape' of a typical Demos2k definition of a model. Although not syntactically mandated in any sense, as a general rule Demos2k models/programs pragmatically adopt the following pattern:

Constant definitions: Demos2k constants are special in that they may be defined in terms of probability distributions — each time such 'constants' are evaluated during simulation, a fresh sample is taken from the specified distribution. The probability distributions supported include standard distributions such as Uniform, Binomial, Geometric, Negative Exponential, Normal, Poisson, and Weibull, as well as arbitrary point/discrete distributions;

Global variable definitions: "Variables are typically used to count the number of events of a certain kind, or the number of times a particular process is activated;

Resource definitions: Resources represent pure synchronizations (in the process-calculus sense) and can be claimed and released by means of 'getR' and 'putR' expressions;

Bin definitions: Bins represent synchronizable entities (note that the term 'resource' is used in the rest of the paper to encompass both the Demos2k notion of 'resource' and the Demos2k notion of 'bin', as described here) into which some quantity of material may be placed and retrieved. These may be used to provide the effect of one entity making a synchronous, concurrent process call on another;

Class definitions: Each entity is a concurrently executing instance of some class. Classes thus represent the behaviour of entities in conventional procedural terms, by manipulating resources in some fashion and by 'holding' (letting time pass) for defined periods of time;

Initial model population, and entity creation: Run length control, typically a hold of some fixed duration;

The (all-important) close statement: Ends the simulation run.

In this form, we may regard Demos2k descriptions as defining behaviour in terms of a Dijkstra-like guarded command language. All active commands test the current system state. If the condition they represent can be met then they are executed — otherwise they are blocked until such time as the condition holds, if at all. Note that Demos2k simulations will typically run for a specified length of time. If either deadlock or livelock arise during simulation runs then these situations are checked for pragmatically. The major difference between process oriented simulation languages (like Demos2k) and pure guarded command languages is that the conditions may have side effects, principally due to the assignment of resource to the active entity. Hence change of state is mediated not only by assignments to variables, and by the claim of resource, but also by entities becoming resources themselves.

Demos2k has been given a simple, elegant, and informative semantics [5–8], abstracting away from the stochastic data collection, in the process calculi SCCS and CCS [22, 23]. It can be argued that the representation of resource in the synchronous semantics (SCCS) is superior to that in the asynchronous semantics (CCS) ([34]).

B The Demos2k Models

We include here the key parameter declarations required for our Demos2k models. These, taken together with the code fragments used to illustrate § 5, should be sufficient for the reader to establish an understanding of the nature of our models.

B.1 Basic Security Operations Model

```
(*
   Security operations model - Brian Monahan, David Pym & Mike Yearworth
   version 1.5 - HYBRID of v1 & 2
   19 July 2006 - update a
*)

//////////////////
// Constants
//////////////////

// Timescaling constants
cons days = 1;
cons hrs  = days/24;
cons mins = hrs/60;
cons secs = mins/60;


// Simulation
cons holdTime   = 365;    // int: sim. period -- in days


// Parameters
cons totalSys   = 20000;   // number of desktop systems under management
cons NsecOps    = 2;       // number of operations staff
cons Nanalysts  = 1;       // number of security risk analysts

cons patchTeams     = 4;   // number of patch teams
cons repairStreams  = 4;   // number of repair streams
cons serviceTeams   = 4;   // number of biz align service teams

cons sysForAlign    = 1;   // number of systems needed per alignment request

cons maxPatchLimit              =  30;     // maximum number of systems patched concurrently by single team
cons lowerAvailLimit            =  90/100;  // lower limit of acceptable availability
cons max_days_unavailable_limit  =  3;       // maximum number of days consecutively unacceptable availability
cons max_repairs_limit          =  850;     // max acceptable length of repair queue

cons clearUpTime             =   2;             // number of days needed to clear up crisis

cons rebootTime         =   normal(10*mins, 1*mins);   // avg. reboot time
cons patchDeployTime    =   normal(1*hrs, 15*mins);    // avg. time to apply a patch to system
cons sigsDeployTime     =   normal(10*mins, 1*mins);   // avg. time to update signatures


// Environment related
// ===================

cons vulnerabilityInterval  =   negexp(365/24);   // Avg. time between vulnerability discovery
cons isExploitable          =   binom(1, 40/100); // Prob. of vuln. being exploitable
cons devExploitTime         =   negexp(19*days);  // Avg. time to develop exploit
cons vulnDiscoveryTime      =   negexp(14*days);  // Avg. time to discover/expose intell. on vuln.

cons sigsEffectiveness      =   uniform(10/100, 20/100);   // level of effectiveness of signatures
cons attackEffectiveness    =   uniform(05/100, 85/100);   // normalised measure of attack capability ...

cons attackDeploymentTime   =   negexp(20*days);  // Avg. time to deploy an attack, given an exploit exists
cons isZeroDayAttack        =   binom(1, 1/10);   // prob. of exploit being immediately deployable
cons isDeployableAttack     =   binom(1, 8/10);   // prob. of exploit being eventually deployable
cons isExposable            =   binom(1, 9/10);   // prob of vulnerability being exposable (i.e. subject of intell.)

cons patchDevTime           =   negexp(15*days);  // Avg. time to develop patch
cons sigDevTime             =   negexp(5*days);   // Avg. time to develop signatures

cons attackRounds           =   puni(1, 3);              // Avg. number of attack attempts.
cons attackInterval         =   normal(2*days, 16*hrs);   // Avg. time between attack attempts

cons dropStuffInterval      =   negexp(300*days);   // Avg. time between environment dropping info.
cons canDropItem            =   binom(1, 1/1000);   // prob. of environment dropping information


// Business related
// ================

// staffing limits
cons staffForPatching       = 1;    // number of staff needed for patching activity
cons staffForSigs           = 1;    // number of staff needed for updating sugnatures
cons staffForAlign          = 1;    // number of staff needed for alignment request
cons staffForRepair         = 1;    // number of staff needed for repairing systems

// biz alignment
cons bizAlignInterval       =   negexp((1/30)*days);     // avg. time between requests
cons bizAlignTime           =   negexp(60*mins);         // avg. time taken to complete biz align tasks.

cons someReassessment       =   puni(0, 3);       // avg. number of vulnerability reassessments performed
cons vulnAssessTime         =   negexp(2*hrs);    // avg. time taken to assess vulnerabilities
cons patchAssessTime        =   negexp(2*hrs);    // avg. time taken to assess suitability of patches
cons assessmentInterval     =   negexp(1*days);   // avg. time between vulnerability assessments

cons isVulnHigh             =   binom(1, 2/10);   // prob of vulnerability being urgent
cons isVulnLow              =   binom(1,  7/10);  // prob of vulnerability being non-urgent, but useful
```

```
cons repairQCheckInTime     =   normal(8*mins, 1*mins);     // time taken to check in systems to repairQ
cons canQuickFix            =   binom(1, 1/10);             // prob. of system being fixable quickly
cons repairTime             =   normal(3*hrs, 10*mins);     // avg. repair time

cons patch_is_irrelevant    =   binom(1, 1/10);   // prob. of patch being neither relevent nor useful

cons patchMaintenanceInterval  =  normal(14*days, 1*days);  // avg. time between patch maintenance
cons sigMaintenanceInterval    =  normal(7*days,  1*days);  // avg. time between sig. defence maintenance

cons systemsNeedingPatching    =  uniform(10/100, 95/100);  // proportion of systems that need patching
cons sysNeedSigs               =  uniform(50/100, 90/100);  // proportion of systems requiring signatures


////////////////////
// Variables
////////////////////

var day                 = 0;
var vulnerableSystems   = 0;
var attackImpact        = 0;

var sysAvail            = 1;
var cSysAvail           = 0;
var cStaffUtil          = 0;

var availSys            = totalSys;
var needsRepairs        = 0;
var needsPatching       = 0;

var online              = 1;

var daysUnavailable     = 0;
var manpowerUsedToday   = 0;
var systemsAvailToday   = 0;

...

////////////////////
// Resources
////////////////////

res(lock,     1);
res(analysts,  Nanalysts);
res(opsStaff, NsecOps);

bin(vulnAssessQ,      0);
bin(patchPublishQ,    0);
bin(sigPublishQ,      0);
bin(bizAlignQ,        0);
bin(vulnHighQ,        0);
bin(vulnLowQ,         0);
bin(batchPatchQ,      0);
bin(repairQ,          0);


////////////////////
// Classes
////////////////////

// External processes/activities
// ==============================

class vulnerable = { ... }
class devExploit = { ... }
class vulnIntelligence = { ... }
class devPatch = { ... }
class devSig = { ... }
class deployAttack = { ... }


// Corporate activities
// ====================

class bizAlignRequests = { ... }
class bizAlignService = { ... }
class vulnerabilityAssessment = { ... }
class assessment = { ... }

// Patch Management
class patchManagement = { ... }
class maintainPatch = { ... }
class maintainSigDefence = { ... }
class deployPatch = { ... }
class patchApplication = { ... }
class deploySig = { ... }
class attack = { ... }
class repairManagement = { ... }
class repair = { ... }


// Management
// ==========

class detectCrisis = { ... }
class crisisManagement = { ... }
class accounting = { ... }
class reporting = { ... }


////////////////////
// Entities
////////////////////

entity(vuln,  vulnerable,        vulnerabilityInterval);  // generate vulnerabilities ...
entity(drop,  dropStuff,         dropStuffInterval);      // allow environmental interference ...
entity(biz,   bizAlignRequests,  0);                      // generate biz alignment requests ...

entity(vulnAssess,  vulnerabilityAssessment,  0);   // vulnerability assessment ...
entity(patchMgmt,   patchManagement,          0);   // patch management ...
```

```
entity(patchMtn,    maintainPatch,            0);   // maintain patching process ...
entity(sigDefMtn,   maintainSigDefence,       0);   // maintain signature defence process ...
entity(patchDep,    deployPatch,              0);   // deploy patches ...
entity(sigDefDep,   deploySig,                0);   // deploy defensive signatures ...
entity(repairMgmt,  repairManagement,         0);   // repair Management ...

do serviceTeams  { entity(bizService,  bizAlignService, 0); }  // service biz alignment requests ...
do repairStreams { entity(repair,      repair,          0); }  // repair teams ...

// business mgmt & reporting
entity(range,       checkRange,     0);   // function to check ranges

entity(check,       detectCrisis,   0);   // check for crisis conditions
entity(accounts,    accounting,     0);   // perform accounting ...
entity(reports,     reporting,      0);   // perform reporting ...

// Run the simulation ...
hold(holdTime);
close;
```

B.2 The Network and Security Operations Model

```
(* Security & Network Operations model

   by Brian Monahan, Jonathan Griffin, David Pym, Mike Wonham, Mike Yearworth

   version 0.2, 26th February 2007
*)


//////////////////
// Constants
//////////////////

// Timescaling constants
cons days   = 1;                // time unit = days
cons hrs    = days/24;
cons mins   = hrs/60;
cons secs   = mins/60;
cons msecs  = secs/1000;

cons weeks      = 7 * days;
cons months     = 4 * weeks;
cons years      = 365 * days;
cons centuries  = 100 * years;


// Simulation
cons holdTime   = 1*years; // simulation period


// Parameters
cons totalSys   = 20000;   // number of desktop systems under management
cons NsecOps    = 4;       // number of security operations staff
cons NnetOps    = 4;       // number of network operations staff
cons Nanalysts  = 2;       // number of security and network analysts

cons patchTeams     = 2;   // number of patch teams
cons repairStreams  = 4;   // number of repair streams
cons serviceTeams   = 4;   // number of biz align service teams
cons networkTeams   = 2;   // number of network operations teams

cons sysForAlign    = 1;   // number of systems needed per alignment request

cons maxPatchLimit               =  30;      // maximum number of systems patched concurrently by single team
cons lowerAvailLimit             =  90/100;  // lower limit of acceptable availability
cons max_days_unavailable_limit  =  3;       // maximum number of days consecutively unacceptable availability
cons max_repairs_limit           =  850;     // max acceptable length of repair queue

cons clearUpTime                 =  2;                // number of days needed to clear up crisis

cons rebootTime         =   normal(2*mins,  20*secs);  // avg. reboot time
cons patchDeployTime    =   normal(20*mins, 5*mins);   // avg. time to apply a patch to system
cons sigsDeployTime     =   normal(5*mins,  1*mins);   // avg. time to update signatures

cons minNetEff          =   0.001;                   // Minimum Network Efficiency
cons maxNetEff          =   1 - minNetEff;           // Maximum Network Efficiency
cons networkDelay       =   negexp(2*hrs);           // Network Delay at 1/2 network efficiency

cons securityControlDecay             =  1/100;        // percentage decay of security control "quality" per day
cons maxSecurityControlEffectiveness  =  0.95;         // maximum security control effectiveness


// Environment related
// ===================

cons vulnerabilityInterval  =   negexp(365/36);   // Avg. time between vulnerability discovery
cons isExploitable          =   binom(1, 45/100); // Prob. of vuln. being exploitable
cons devExploitTime         =   negexp(19*days);  // Avg. time to develop exploit
cons vulnDiscoveryTime      =   negexp(14*days);  // Avg. time to discover/expose intell. on vuln.

cons sigsEffectiveness      =   uniform(10/100, 30/100);   // level of effectiveness of signatures
cons attackEffectiveness    =   uniform(05/100, 85/100);   // normalised measure of attack capability ...

cons attackDeploymentTime   =   negexp(20*days);  // Avg. time to deploy an attack, given an exploit exists
cons isNoWarningAttack      =   binom(1, 1/10);   // prob. of exploit being immediately deployable
cons isDeployableAttack     =   binom(1, 8/10);   // prob. of exploit being eventually deployable
cons isExposable            =   binom(1, 9/10);   // prob of vulnerability being exposable (i.e. subject of intell.)

cons patchDevTime           =   negexp(15*days);  // Avg. time to develop patch
cons sigDevTime             =   negexp(5*days);   // Avg. time to develop signatures

cons attackRounds           =   puni(1, 3);               // Avg. number of attack attempts.
cons attackInterval         =   normal(2*days, 16*hrs);   // Avg. time between attack attempts


// Business related
// ================

// staffing limits
cons staffForPatching       = 1;       // number of staff needed for patching activity
cons staffForSigs           = 1;       // number of staff needed for updating sugnatures
cons staffForAlign          = 1;       // number of staff needed for alignment request
cons staffForRepair         = 1;       // number of staff needed for repairing systems
cons staffForNetworkOpn     = 2;       // number of staff needed for network operations

// biz alignment
cons bizAlignInterval       =   negexp((1/30)*days);     // avg. time between requests
cons bizAlignTime           =   negexp(60*mins);         // avg. time taken to complete biz align tasks.
cons bizNeedsNetReconfig    =   binom(1, 5/100);         // prob. of biz alignment needing reconfig of network

cons someReassessment       =   puni(0, 3);       // avg. number of vulnerability reassessments performed
cons vulnAssessTime         =   negexp(2*hrs);    // avg. time taken to assess vulnerabilities
cons patchAssessTime        =   negexp(2*hrs);    // avg. time taken to assess suitability of patches
cons assessmentInterval     =   negexp(1*days);   // avg. time between vulnerability assessments

cons isVulnHigh             =   binom(1, 2/10);            // prob of vulnerability being urgent
cons isVulnLow              =   binom(1, 7/10);            // prob of vulnerability being non-urgent, but useful
```

```
cons vulnNeedsNetWorkAround =   binom(1, 10/100);            // prob of response to vuln. needing reconfig of network

cons repairQCheckInTime     =   normal(15*mins, 2*mins);    // time taken to check in systems to repairQ
cons canQuickFix            =   binom(1, 3/100);            // prob. of system being fixable quickly
cons repairTime             =   normal(3*hrs, 10*mins);     // avg. repair time

cons networkOperationTime   =   negexp(2*days);             // time taken to perform network operations
cons networkDisruptionTime  =   negexp(15*days);            // duration of disruptive effect upon network
cons networkEfficiencyIncr  =   uniform(1/100, 5/100);      // change in efficiency due to disruption
cons networkOpnsPlanning    =   negexp(1*days);             // time taken to plan network operations

cons patch_is_irrelevant    =  binom(1, 1/10);              // prob. of patch being neither relevent nor useful

cons patchMaintenanceInterval  =  normal(14*days, 1*days);  // avg. time between patch maintenance
cons sigMaintenanceInterval    =  normal(7*days,  1*days);  // avg. time between sig. defence maintenance
cons patchDeployIntv           =  normal(1*days,  3*hrs);   // time between patch deployment phases

cons systemsNeedingPatching    =  uniform(10/100, 95/100);  // proportion of systems that need patching
cons sysNeedSigs               =  uniform(50/100, 90/100);  // proportion of systems requiring signatures


//////////////////
// Variables
//////////////////

var day                 = 1;
var vulnerableSystems   = 0;              // number of systems potentially at risk
var attackImpact        = 0;              // potential for damage from an attack (variable)

var networkEfficiency   = maxNetEff;      // Efficiency of network (minNetEff <= networkEfficiency <= maxNetEff)

var securityControlEffectiveness
                        = maxSecurityControlEffectiveness;
                                          // Effectiveness of network controls (between 0 and 1)

var sysAvailability     = 1;              // availability of systems (per day)
var cSysAvailability    = 0;              // cumulative/averaged availability of systems

var secStaffUtil        = 0;              // security operations staff utilisation (per day)
var cSecStaffUtil       = 0;              // cumulative/averaged security operations staff utilisation

var netStaffUtil        = 0;              // network operations staff utilisation (per day)
var cNetStaffUtil       = 0;              // cumulative/averaged network operations staff utilisation

var availSys            = totalSys;       // number of systems available
var needsRepairs        = 0;              // number of outstanding repair tasks to be done
var needsPatching       = 0;              // number of systems that need patching
var networkTasks        = 0;              // number of oustanding network tasks (not yet started/processed)

var online              = 1;              // indicator variable asserting if business is online

var daysUnavailable     = 0;              // count of consecutive days unavailable
var totalDaysUnavail    = 0;              // total number of days unavailable

...

//////////////////
// Resources
//////////////////

res(lock,      1);
res(analysts,  Nanalysts);
res(opsStaff,  NsecOps);
res(netStaff,  NnetOps);

bin(vulnAssessQ,      0);
bin(patchPublishQ,    0);
bin(sigPublishQ,      0);
bin(bizAlignQ,        0);
bin(vulnHighQ,        0);
bin(vulnLowQ,         0);
bin(batchPatchQ,      0);
bin(repairQ,          0);
bin(netOpnsQ,         0);


//////////////////
// Classes
//////////////////

// External processes/activities
// =============================

class vulnerable = { ... }
class devExploit = { ... }
class vulnIntelligence = { ... }
class devPatch = { ... }
class devSig = { ... }
class deployAttack = { ... }


// Corporate activities
// ====================

class bizAlignRequests = { ... }
class bizAlignService = { ... }
class secMgmtVulnAssessment = { ... }
class assessment = { ... }


// Network reconfiguration

class networkManagement = { ... }
class networkOperations = { ... }
class networkEffect(factor) = { ... }
class calcNetworkDelay = { ... }

// Patch Management
```

```
class patchManagement = { ... }
class maintainPatch = { ... }
class maintainSigDefence = { ... }
class deployPatch = { ... }
class patchApplication = { ... }
class deploySig = { ... }
class attack = { ... }
class repairManagement = { ... }
class repair = { ... }


// Management
// ==========

class detectCrisis = { ... }
class crisisManagement = { ... }
class accounting = { ... }
class reporting = { ... }


////////////////////
// Entities
////////////////////

// Generators
entity(vuln,  vulnerable,        vulnerabilityInterval);  // generate vulnerabilities ...
entity(biz,   bizAlignRequests,  0);                      // generate biz alignment requests ...

// Perpetual processes
entity(vulnAssess,  secMgmtVulnAssessment,    0);   // security management and vulnerability assessment ...
entity(patchMgmt,   patchManagement,          0);   // patch management ...
entity(patchMtn,    maintainPatch,            0);   // maintain patching process ...
entity(sigDefMtn,   maintainSigDefence,       0);   // maintain signature defence process ...
entity(patchDep,    deployPatch,              0);   // deploy patches ...
entity(sigDefDep,   deploySig,                0);   // deploy defensive signatures ...
entity(repairMgmt,  repairManagement,         0);   // repair Management ...
entity(networkMgmt, networkManagement,        0);   // network configuration Management ...

do serviceTeams  { entity(bizService,  bizAlignService,   0); }  // service biz alignment requests ...
do repairStreams { entity(repair,      repair,            0); }  // repair teams ...
do networkTeams  { entity(networkOpns, networkOperations, 0); }  // network operations

// business mgmt & reporting
entity(check,       detectCrisis,   0);   // check for crisis conditions
entity(accounts,    accounting,     0);   // perform accounting ...
entity(reports,     reporting,      0);   // perform reporting ...

// Run the simulation ...
hold(holdTime);
close;
```