Semantics for Structured Systems Modelling and Simulation

Matthew Collinson
HP Labs, Bristol, UK
collinson@hp.com

Brian Monahan
HP Labs, Bristol, UK
brian.monahan@hp.com

David Pym
HP Labs, Bristol, UK
University of Bath, UK
david.pym@hp.com

ABSTRACT

Simulation modelling is an important tool for exploring and reasoning about complex systems. Many supporting languages are available. Commonly occurring features of these languages are constructs capturing concepts such as process, resource, and location. We describe a mathematical framework that supports a modelling idiom based on these core concepts, and which adopts stochastic methods for representing the environments within which systems exist. We explain how this framework can be used to give a semantics to a simulation modelling language, Core Gnosis, that includes basic constructs for process, resource, and location. We include a brief discussion of a logic for reasoning about models that is compositional with respect to their structure. Our mathematical analysis of systems in terms of process, resource, location, and stochastic environment, together with a language that captures these concepts quite directly, yields an efficient and robust modelling framework within which natural mathematical reasoning about systems is captured.

Categories and Subject Descriptors

I.6 [**Simulation and modelling**]: General; F.3 [**Logics and meanings of programs**]: Miscellaneous; H.1 [**Models and principles**]: Systems and information theory

General Terms

Theory, languages, experimentation, design

Keywords

Simulation, modelling, semantics, process calculus, logic

1. INTRODUCTION

Executable modelling languages are important tools in science and engineering, providing methods for exploring systems that are too complex to be usefully described in simple, analytical terms.

It is very often difficult to validate such models of complex systems, and there are important questions about faithfulness of representation of the underlying system and of the degree to which such models can be predictive. A possible source of errors lies in the modelling language itself, because (contrary to the beliefs of many) languages are themselves complicated artifacts. It is very important to use a modelling language which is well-understood, both by its authors and by its users. This points towards the disciplined use of small, expressive, languages that have a formal semantics, that are implemented with a high-degree of integrity, and which employ constructs that naturally support the modelling idiom.

A landmark achievement was the construction of process modelling languages, particularly Simula [13] (which extends Algol for modelling), which use the notion of concurrent processes to structure models. This was distilled into a small, expressive language called Demos [4] by Birtwistle, which emphasizes the disciplined use of further structure, namely resource, by the processes.

In fields such as program logic, programming language semantics, and concurrency, the introduction of mathematical semantic methods has led to significant insights in expressiveness and improved reliability properties. In the field of modelling and simulation, however, semantics has made relatively little impact. One significant and elegant exception is the work of Hillston and her colleagues (see, for example, [16, 19, 20]), in which a process calculus is enriched with stochastic components, together with an account of its stochastic properties in terms of Markov chains. Hillston et al's framework has been explored in detail, has tool support, and has been deployed in wide range of examples. The Markov chain point of view is also the basis of the Möbius system [28]. Our approach differs in that we separate system semantics and modelling language, interpreting the latter in the former.

While the notion of process has been explored in some detail by the semantics community, concepts like resource have almost always been treated as second class. There are many advantages to doing this, from the point-of-view of a theorist. We take the opposite view. That is, we try to see what can be gained by developing an approach in which the structures present in applied modelling languages are given a rigorous treatment as first-class citizens in a theory. This has allowed us to develop our own disciplined approach to applied modelling and an associated tool Core Gnosis.

It is useful to argue — see, for example, [8] — that the key structural aspects of systems are the ones discussed below. This point of view is consistent with the classical view of distributed systems, as described, for example, in [12].

Process. A synthetic system exists in order to deliver services, and services can be conveniently understood as processes that execute on the system's architecture. Typically, it will be necessary for many services to execute, concurrently and sequentially, in order for a service to be delivered. Similarly, natural processes execute relative to natural substrates. Our main focus here is on synthetic systems; in particular, large-scale information systems.

SIMUTools 2010 March 15–19, Torremolinos, Malaga, Spain.
Copyright 2010 ICST, ISBN 78-963-9799-87-5.

For example, we might wish to model the handling of boats by some docks. A boat arrives at the docks, is tugged to a jetty, is then unloaded and/or loaded, and then tugged away from the jetty before being ready to depart the docks. We can consider the boat's passage through the docks to be a process, understood as a sequence of actions, that executes relative to the infrastructure (i.e., the tugs, the jetties, the cranes, the stevedores) of the docks. The service delivered by the docks might consist of concurrently processing several boats, and might require a number of subsidiary processes.

There are number of familiar aspects of the intuitive notion of process that we might naturally want our model of process, discussed mathematically below in Section 2, to capture. These include, among others, sequencing, choice, concurrency, and recursion.

Resource. The infrastructure of a system, relative to which the system's processes execute, consists of a collection of resources that may be utilized by the processes in order to achieve their intended purposes.

For example, various resources are required to be available within the docks in order for boats to be handled: tugs are needed to take boats to jetties, cranes and stevedores are required to unload/load boats, and tugs are required to take boats away from jetties.

As with processes, there are some basic properties of the intuitive notion of resource that we would like our model of resource to capture. Recent work on resource semantics (see, for example, [33, 34]) suggests that capturing the idea that resource elements may be combined and compared is sufficient for a great deal of progress to be made. This too is developed mathematically in Section 2.

Location. In general, the architectures of systems are highly distributed, logically and/or physically. The system's resources are distributed around a collection of places, and these places have (directed) connections between them.

For example, suppose we wish to model a system of docks that handles both low-security boats, perhaps carrying produce or fossil fuels, and high-security boats, perhaps carrying nuclear fuels [8]. The docks may be divided into two zones, or locations, one low-security and one high-security. Specialized resources may be associated with each zone, high-security jetties, tugs, and stevedores being separated from their low-security counterparts. It may, however, be the case that low-security traffic, perhaps being much more frequent, puts a much greater burden on the low-security zone than does less frequent high-security traffic on the high-security zone. Accordingly, the docks' operators may wish to be able to divert high-security tugs and stevedores to low-security jetties in a controlled fashion. So, transition from high-security to low-security may be permitted only via an intermediate 'debriefing' location that applies appropriate controls to the information flows. In this example, we assume that direct connection from the low-security zone to the high-security zone is permitted since no high-security information flows in that direction.[1] In Section 3, for variety, we give a related 'dual' example, in which movements from low- to high-security is controlled by a 'vetting' location.

The notion of location provides more than a way to understand information flows, however. It also provides a basis for mechanisms for abstraction, where a portion of the structure of locations and connections is replaced with a less detailed version, and refinement, where a portion of the structure of locations and connections is replaced with a more detailed version. These ideas will be discussed mathematically in Section 2.

Environment. Systems exist within external environments, from which events are incident upon the system's boundaries. Typically, the environment is insufficiently understood and too complex to be represented in the same, explicit, form as the system itself.

For example, when modelling the handling of the arrival, unloading/loading, and departure of boats at a dock, we might choose not to model the patterns of trade in the surrounding seas, but rather simply represent the arrival of boats at the entrance to the docks as a stochastic process.

In general, then, we choose to represent the impact of the environment on the system of interest just as random events that are incident upon the system. There is, however, another important rôle for stochastic methods in our approach. Even within the system of interest, there may be (perhaps quite complex) components that we do not need to model in (process, resource, location) detail. The impact of such components on the operation of the overall system can often be handled stochastically.

Our integration of the stochastic and structural aspects of our models also differs from the approach of Hillston et al [16, 19, 20]. Whereas PEPA is a language within which stochastic constructs are internalized, we reside our stochastic capability wholly within our associated tool, Core Gnosis.[2] Core Gnosis, which is described in Section 3, also implements notions of process, resource, and location, and its semantics is given denotationally, as in the semantics of programming languages (see, for example, [40], and particularly [35]), in terms of the process calculus (L)SCRP [10, 8], which includes appropriate mathematical models of resource and location, and which is described in Section 2.

Section 4 provides a substantive sketch of the denotational semantics of Core Gnosis. Here the distinction between our approach and that of Hillston et al. is very clear. Whereas they work intensionally within a process calculus that includes a stochastic component, our approach is to give a denotational semantics to a modelling language (Core Gnosis) that includes process, resource, location, and stochastic components. A complete treatment of this semantics would entail giving an interpretation, in (L)SCRP's language of processes, resources, and locations, of Core Gnosis's event-scheduler. Such a treatment, including a precise representation of the timing of events, is a very substantial undertaking, beyond the scope of this short paper. Instead, we give a substantial sketch of a version of such a semantics in which we map Core Gnosis models (L)SCRP processes, preserving the scheduler's order of major events.

In Section 5, we sketch a modal logic, MBI, which is related to SCRP in the sense of Hennessy–Milner logic [18, 38]. MBI is a bunched logic, with both additive and multiplicative connectives and quantifiers at the same level of abstraction [29]. As a result, MBI provides logical characterizations of some key structural aspects of our (L)SCRP's representation of system models.

2. A SEMANTIC BASIS

Each of the components of a system model that we have described in Section 1 can be captured mathematically. Processes are usually captured in a process calculus, such as SCCS [24], CCS [25], or the pi-calculus [26]. Resources can be captured using resource monoids, as in bunched logic [29, 33] and its applications, such as Separation Logic [21, 34]. Location can be captured using (hyper-) graph-like structures, and all of this can be combined with probability distributions to capture environments.

2.1 Processes and Resources

We give a brief review of the process calculus SCRP [10] of

[1]Of course, in practice this would require strong assumptions about the security 'clearance' of the resources, such as the stevedores, involved.

[2]*gnosis: Gk, knowledge to influence and control.*

resources and processes (which builds on and consolidates [31, 30]) and its extension to locations [8].

Our starting points are Milner's synchronous calculus of communicating systems, SCCS [24], perhaps the most basic of process calculi, and the resources semantics of bunched logic [29, 33]. The key components for our purposes are the following:

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

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

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

Mathematically, our notion of resource — which encompasses natural examples such as space, memory, and money — is based on (ordered, partial, commutative) monoids (e.g., the non-negative integers with addition, zero, and less-than-or-equals), and captures the basic properties of resources discussed briefly in Section 1:

- Each type of resource is based on a basic set of resource elements;
- Resource elements can be combined (and the combination has a unit);
- Resource elements can be compared.

Formally, we take pre-ordered, partial commutative monoids ,

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

where $\mathbf{R}$ is the carrier set of resource elements, $\circ$ is a partial monoid composition, with unit $\mathbf{e}$, and $\sqsubseteq$ is a pre-order on $\mathbf{R}$.

The basic idea is that resources, R, and processes, E, co-evolve,

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

according to the specification of a partial 'modification function', $\mu : (a, R) \mapsto R'$, that determines how an action a evolves E to E' and R to R'.

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

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

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

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

A modification function is required to satisfy some basic coherence conditions:

- $\mu(1, R) = R$, where 1 is the unit action, and
- if $\mu(a, R) \circ \mu(b, S)$, and $R \circ S$ are defined, then $\mu(ab, R \circ S) \simeq \mu(a, R) \circ \mu(b, S)$,

where $\simeq$ is Kleene equality, for all actions a and b and all resources R and S. In certain circumstances, additional equalities may be required [10, 8].

Sums and recursion are formulated in familiar ways:

$$\frac{R, E_i \xrightarrow{a} R', E'}{R, \sum_{i \in I} E_i \xrightarrow{a} R', E'},$$

where I is an indexing set, and

$$\frac{R, E_i[E/X] \xrightarrow{a} R', E'}{R, fix_i X.E \xrightarrow{a} R', E'},$$

where E_i is the ith component of the tuple E of processes.

Of more interest is hiding,

$$\frac{R \circ S, E \xrightarrow{a} R' \circ S', E'}{R, (\nu S)E \xrightarrow{(\nu S)a} R', (\nu S')E'},$$

in which the resource S becomes bound to the process E (we elide the definition of the action $(\nu S)a$ [10, 8]). This construction replaces, and generalizes, restriction in calculi such as SCCS.

2.2 Location

If one wishes to construct models of systems with location in the SCRP calculus, then this must be done using the resource and/or process components. For some simple situations, the notion of location can be treated as resource — for example, this would be sufficient to give a process algebraic account of Separation Logic (where location is thought of as resource). For more complex notions, an encoding into the process component must be used.

This section describes how to extend SCRP to handle system states with three components: location, resource and process. Thus location will be treated as a first-class citizen in the formalism. We name the resulting (family of) calculi LSCRP.

Just as our treatment of resources begins with some basic observations about some natural and basic properties of resources, so we begin our treatment with the following basic requirements of a useful notion of location [8, 9]:

- A collection of atomic locations — the basic places — which generate a structure of locations;
- A notion of (directed) connection between locations — describing the topology of the system;
- A notion of sublocation (which respects connections);
- A notion of substitution (of a location for a sublocation) that respects connections — substitution provides a basis for abstraction and refinement in our system models;
- A product (again, monoidal) of locations (an inessential but useful technical property), suitably coherent with the other products [8].

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

Treating location as a first-class citizen in this way does not lead to a process calculus with operational behaviour that is more expressive in absolute terms. It does, however, lead to greater pragmatic expressiveness: we claim that it simplifies the construction of models of a wide range of systems. It also makes it easier to write specifications about located resource in the logical language. In some circumstances, such as those that obtain in Separation Logic

[34, 21], location can be treated as a form of resource (in such settings, the topology of locations plays no role).

The resulting calculus has transition systems with dynamic behaviour of the following form:

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

where a is an action (in the usual process sense), L, L' are location environments, R, R' are resource environments and E, E' are processes used to control the evolution. Following the approach sketched above, we define a modification function, μ, which, for each action a, location L, and resource R, determines the evolved location L' and resource R'. In a state L, R, E, the component L carries the relevant information about the topology of the model and R carries the relevant information about the distribution of the resources around the model's topology.

The operational semantics of LSCRP extends that of SCRP in the evident way. Modification functions are extended to include locations, $\mu : (a, L, R) \mapsto (L', R')$, with corresponding versions of the coherence conditions. Then the following is the rule for action prefix:

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

where $(L', R') = \mu(a, L, R)$.

The following quite general form of the product rule in the presence of locations makes use of a notion of product of locations:

$$\frac{L, R, E \xrightarrow{a} L', R', E' \quad M, S, F \xrightarrow{b} M', S', F'}{L \bullet M, R \circ S, E \times F \xrightarrow{a \cdot b} L' \bullet M', R' \circ S', E' \times F'} \; Product$$

where $\bullet$ is the product of locations. Various simpler forms, such as taking a fixed location, make sense in absence of a product of locations [8, 9]. We can also take a Frame rule (with respect to resources):

$$\frac{L, R, E \xrightarrow{a} L', R', E'}{L, R \circ S, E \xrightarrow{a} L', R' \circ S', E'}$$

provided $\mu(a, L, R \circ S))$ is defined.

Alternative semantic approaches to systems modelling, in addition to the work of Hillston et al., include the work of Milner on bigraphs [27], in which, essentially, a single language is used to capture process, resource, and location. A detailed comparison with our approach is beyond the scope of this short paper.

2.3 Environment

We have explained that, in our approach, the environment — that is, those aspects of the environment that we do not wish to model explicitly — is handled stochastically. Yet neither SCRP nor LSCRP is a stochastic calculus. How does this work? In contrast to the intensional approach of, for example, Hillston et al., our approach is in the spirit of the denotational semantics of programming languages (see, for example, [40]). We develop our semantic structures in parallel with our modelling language, Core Gnosis, the two being related by an interpretation of the modelling language in the structures, about which we seek to establish certain properties.

We describe such an interpretation in some detail in Section 4. The interpretation of the structural components — process, resource, and location — is relatively direct. The interpretation of Core Gnosis' stochastic capabilities is, however, essential to the semantics of control in this context.

2.4 Basic Meta-theory

This is not a paper about the mathematical theory of processes, resources, and locations. That has been presented at length elsewhere [10, 8, 11]. Rather, it is an overview of how that mathematics provides a rigorous basis for systems modelling. Nevertheless, it seems worthwhile to give a brief indication of the key properties that are require to support its use both as a conceptual framework and as a semantic domain for the Core Gnosis modelling language.

In a process algebra, it is desirable to have a notion of equality that goes beyond syntactic identity and identifies processes with similar behaviour. The usual notion is that of an equivalence relation called *bisimulation*. In (L)SCRP, where the decomposition of state into the three components introduces a degree of delicacy, there are two natural notions of equivalence: a *local* one and a *global* one.

The local equivalence is on states, and is written $L, R, E \approx L, R, F$. It is a bisimulation relation that takes account of the resource and its location. The global relation, written $E \sim F$, quantifies the equivalence over all locations and resources. The global equivalence is a congruence, but the local equivalence is not. Nevertheless, the local equivalence has many useful conceptual and theoretical properties [30, 31] (but see the erratum). We shall return to this point in Section 5. The details of these equivalences and their key theoretical properties may be found in [10, 8].

In the sequel we make use of a few additional pieces of notation. Rather than using the *fix* notation, we use an equivalent formulation using *process constants* for recursive definitions. Thus for example, processes $E_0, \ldots, E_n$ may be defined by a sequence of equations $C_i := F_i$, for $0 \leq i \leq n$, where each F_i is a process term (but possibly containing C_j, for $0 \leq j \leq n$). We further abbreviate by simply writing the equations for E_i in the form $E_i := \ldots E_j \ldots$. As a first example, there is the *unit* or *tick* process 1, given by $1 = 1 : 1$. For a second example, for any process E, there is the *delayed* process $\delta(E)$, given by $\delta(E) := E + 1 : E$. The *asynchronous prefix* operator '.' may be defined from the delay operator δ, by taking $a.E$ to be $a : \delta(E)$, for all a and E.

3. THE CORE GNOSIS LANGUAGE

We introduce the *Core Gnosis* modelling language via a series of examples to illustrate the disciplined approach to modelling discussed above. Core Gnosis is a strict subset of Gnosis, a larger, more expressive language possessing more extensive, but conventional, data representation capabilities.[3] Core Gnosis (and Gnosis) are developments building on experience from Birtwistle's Demos [4], on Demos2k [14], and Located Demos2k [6, 7, 8], and the rigorous mathematical framework described herein.

Core Gnosis includes constructs for describing processes, resources, and locations that capture many (though, at this stage, not quite all) aspects of the mathematical structures described in Section 2. Core Gnosis also provides a good degree of stochastic capability. Events (actions) may be drawn from the following probability distributions: uniform, normal, negative exponential, and Weibull; for example, these allow us to model stochastic queuing and Markov chain process phenomena. Experience suggests (see, for example, [3, 2, 1, 41]) that this choice provides a range of expressivity that is sufficient for a very wide range of examples whilst keeping the tool conceptually and pragmatically compact.[4]

Our starting point is a classic example from the work of Birtwistle [4]: the docking of boats in a harbour with various jetties and tugs, which we extend to include secure docking of boats. Here is the first version in Gnosis:

[3]Gnosis is at an advanced stage of development at HP Labs [17].

[4]The full Gnosis tool [17] will, in common with Demos and Demos2k, for example, include a very wide range of distributions. See, for example, [15, 22] for explanations of how this is done.

```
-- title : Boats example : time units = hours
-- seed = 426724262

param runTime = (24 * 7)                // 7 days

param numJetty = 2; param numTug = 3

param dockTime    = negexp(2.0)
param undockTime  = negexp(1.5)
param unloadTime  = uniform(1.0, 4.0)

param boatMeanArrival = 10.3
param boatDelay       = negexp(boatMeanArrival)

share jetty numJetty
share tug   numTug

process boat = {
  claim 1 jetty
    claim 2 tug; hold(dockTime); release 2 tug
      hold(unloadTime)
    claim 1 tug; hold(undockTime); release 1 tug
  release 1 jetty
  hold (boatDelay); launch boat
}

launch boat

hold (runTime)
close
```

The model first defines a series of constants and distributions, followed by two shared resource elements, jetties and tugs. A single process corresponding to a relevant abstraction of the boat's activities is then defined. This process defines a number of sequential activities performed by the boat: getting a jetty, getting some tugs, docking, unloading, and so on. Note that there are many activities associated with the boat that are not modelled (e.g., navigation). Finally, a boat instance is (immediately) launched and the simulation then runs for 168 (= 24×7) (simulated) hours before closing.

Notice how the boat process launches a further boat process after a randomized time delay, `boatDelay`, ensuring a random sequence of boat arrival instances. Core Gnosis checks at runtime whether process instances terminate whilst owning some shared resource, and fails if any are found.

3.1 Adding located resource: secure boats

We illustrate location by adding security properties to the simple boats example. The idea is that there are two kinds of dock, Basic and Secure, plus a Guard area to check tugs entering the Secure docking area. There are some number of basic and secure jetties (which remain in place) to which boats can be docked. There is also a pool of tugs that can be moved to and from Basic and Guard, and also from Guard to Secure and and back to Basic. We present the extended model in sections. The first section gives the parameters:

```
param runTime = (24 * 7)                // 7 days

param numJetty       = 2; param numTug       = 3
param numSecureJetty = 1; param numSecureTug = 3

param dockTime    =  weibull(2.0, 1.5)
param undockTime  =  weibull(1.5, 1.5)
param unloadTime  =  uniform(1.0, 4.0)

param checkTime   =  weibull (2.0, 3.0)

param passCheck   =  normal (1, 0.5)
param passLevel   =  0.5

param boatMeanArrival = 10.3
param boatDelay = negexp (boatMeanArrival)
param secureBoatMeanArrival = 18.9
param secureBoatDelay = negexp (secureBoatMeanArrival)

param checkInterval = 3.5
param checkDelay = negexp (checkInterval)
```

We now have parameters not only for the standard boats but also, for instance, those defining the stream of secure boats as well (e.g., `secureBoatMeanArrival`).

Using these parameters, we now introduce *locations* and *links* (connections) between locations. The links are generally used to constrain moving resources from one location to another. We then define the shared, located resources we need:

```
location  Basic, Guard, Secure
link      Basic ↔ Guard → Secure → Basic

share jetty@Basic    numJetty
share jetty@Secure   numSecureJetty

share tug@Basic      numTug
share tug@Secure     numSecureTug
```

More generally, although we only use simple locations in this paper, the locations can also be hierarchically addressed (cf. URLs):

$$\text{Resource@Loc}_1/\text{Loc}_2/\ldots/\text{Loc}_n$$

This hierarchical structure also extends to the way that linkage is defined.

There are now two kinds of boat, a standard boat and a secure boat. Standard (i.e, low-security) boats can only use the Basic jetties whereas secure (i.e., high-security) boats can only use the Secure jetties. Each tug can be used to dock/undock the boats in either docking area. However, tugs may need to change their rôle/location and move from one to the other as circumstance demands.

Here is the standard boat process:

```
process boat = {
  claim 1 jetty@Basic

    select [claim 2 tug@Basic] {
      hold(dockTime)
      release 2 tug@Basic
    }

    or [claim 2 tug@Guard] {
      move share (2) tug@Guard → tug@Basic
      hold(dockTime)
      release 2 tug@Basic
    }

    or [claim 2 tug@Secure] {
      move share (2) tug@Secure → tug@Basic
      hold(dockTime)
      release 2 tug@Basic
    }

    hold(unloadTime)

    select [claim 1 tug@Basic]  {
      hold(dockTime)
      release 1 tug@Basic
    }

    or [claim 1 tug@Guard] {
      move share (1) tug@Guard → tug@Basic
      hold(dockTime)
      release 1 tug@Basic
    }

    or [claim 1 tug@Secure] {
      move share (1) tug@Secure → tug@Basic
      hold(dockTime)
      release 1 tug@Basic
    }
  release 1 jetty@Basic

  hold (boatDelay); launch boat
}
```

Note how tugs are initially claimed from either the Basic, Guard, or Secure pools and, if necessary, moved into the Basic pool. Our version of **move** can only move resources *already* owned by the process (i.e., claimed) from one location to another. To do the move, there must be a valid link between the two locations. Once a resource is moved to a destination, it must also be released back to that location and not from where it was claimed.

Here is the secure boat process:

```
process secureBoat = {
  claim 1 jetty@Secure
    select [claim 2 tug@Secure] {
      hold(dockTime)
      release 2 tug@Secure
    }
    or [claim 2 tug@Guard] {
      move share (2) tug@Guard → tug@Secure
      hold(dockTime)
      release 2 tug@Secure
    }

    hold(unloadTime)

    select [claim 1 tug@Secure] {
      hold(undockTime)
      release 1 tug@Secure
    }
    or [claim 1 tug@Guard] {
      move share (1) tug@Guard → tug@Secure
      hold (undockTime)
      release 1 tug@Secure
    }
  release 1 jetty@Secure

  hold(secureBoatDelay); launch secureBoat
}
```

Again, note how the tug resources can be claimed and used from only the Guard and Secure locations, and then moved as necessary. This ensures that only potentially vetted tugs can be used within the secure docking area.

The next process performs the 'randomized inspection' of tugs — the check process takes either one or two tugs in Basic and 'decides' (via a distribution) whether or not to inspect. The tugs always end up in the Guard area:

```
process check = {
  select [claim 1 tug@Basic] {
    move share (1) tug@Basic → tug@Guard

    if [passCheck > passLevel] {hold(checkTime)} or else {}
    release 1 tug@Guard
  }

  or [claim 2 tug@Basic] {
    move share (2) tug@Basic → tug@Guard

    if [passCheck > passLevel] {hold(checkTime)} or else {}
    release 1 tug@Guard

    if [passCheck > passLevel] {hold(checkTime)} or else {}
    release 1 tug@Guard
  }

  hold(checkDelay); launch check
}
```

Finally, we launch all three processes, *boat*, *secureBoat*, and *check*, to perform the overall simulation:

```
launch boat
launch secureBoat
hold(checkDelay); launch check

hold (runTime)
close
```

The evolution of the Core Gnosis abstract machine determines the observable change of state recorded by the trace (history).

The language also allows for statements of the form **forget**(l, l') and **recall**(l, l'), where l and l' are simple locations. These statements make the topology of the system dynamic, in that processes may not be able to use the declared links at all points in time. The **forget**(l, l') statement changes the system state by dropping the link from l to l'. Note that **move** statements taking resources from l to l' will block when the link is thus broken. A **recall**(l, l') statement re-connects the link from l to l'. A process which is blocked on a **move** from l to l' will be un-blocked when this link is recalled. For example, one may wish to consider enriching the Secure Boats example, so that the tugs kept at the `Guard` location are, periodically, distrusted. This may be represented by having the link from `Guard` to `Secure` forgotten and recalled periodically.

As we have already pointed out at the semantic level, examples such as secure boats, and other far more complex examples, can evidently be coded in modelling languages, such as Demos [4], which lack a concept of location. Two possible approaches are the following: use the imperative structure of the code to capture the distinctions between implicit locations; and encode a data type of locations and employ it whenever a distinction between locations is required. The first choice is clearly very limited in the extent to which it can be employed. The second, whilst quite flexible, leads to models within which there is a great deal of code whose only purpose is to capture the necessary data type, and is not otherwise related to the meaning of the model, as captured by the mathematical framework. Experience suggests that such models are difficult to conceptualize, prone to error, and computationally inefficient.

4. THE SEMANTICS OF CORE GNOSIS

4.1 The Approach

The semantics we give is a translation of Core Gnosis into the more foundational setting of the process calculus LSCRP. The translation is partial, in the sense that we restrict our attention to models that have been written with a particular discipline — we refer to such models as restricted Core Gnosis models. Furthermore, the translation is forgetful, in the sense that some of the structure of the original model is lost in translation. A similar approach was taken for Demos2k [14] in [5].

Critical features of the original model are, however, preserved under the translation. Specifically, certain critical aspects of control flow and the orderings of certain kinds of important event. From this we can argue that (most of) the aspects of control flow that are written by a modeller using declarations of concurrent processes in Core Gnosis are meaningful in the simpler setting of LSCRP, where behaviour is easier to reason about.

The reader may reasonably ask why we do not give a complete semantics to the entire Core Gnosis language that preserves all aspects of the original model. Such a semantics has been the traditional goal of the semantics community. Furthermore, any property that one wanted to prove about a Core Gnosis model could be proved instead for the LSCRP translation, where analysis is easier, and formal methods for proofs are available. There are two reasons why we do not attempt this. The first is that such a translation is a huge task — the complexity of even a relatively small language like Core Gnosis should not be underestimated. The second is that such a translation would help the applications modeller very little. In particular, one would be forced to encode all aspects of the Core Gnosis language and abstract machine into the process calculus.

In common with many other simulation languages, the abstract machine used for the execution of Core Gnosis models is based around the notion of scheduler. The essential idea behind this is, that at any (simulation) time there are a finite number of live process instances, but that there will always be one of these which performs its action first. This is linked to the discrete nature of evolution captured by such tools. Concurrency is represented by the fact that actions may (and often do) occur at the same simulation time, even though they are given an ordering. The scheduling is, in part, dependent upon draws from probability distributions relating to **hold**-statements. Apart from such (pseudorandom) draws, the scheduling is entirely deterministic.

The scheduling algorithm makes use of two principal data structures, the *process-queue* and the *blocked-list*, as well as some other global parameters including the current *(simulation) time*, the current model-topology, and the currently available model-resources. The process-queue is a list of *process-actions*, each of which consists of an identifier (for a process instance) and a representation of the control stage in its declaration that it has so far reached (the continuation corresponding to that instance). The element at the head of the queue is the next process-instance that may proceed with its next action, unless some elements from the blocked-list may unblock. The blocked-list consists of a list of processes that are currently suspended (blocked) because their leading action has been issued, but cannot yet be satisfied (e.g., a **claim** on insufficient resource. At any stage, each identifier occurs at most once in the union of the process-queue with the blocked-list. Morally, this union represents the current synchronous product of live processes that we would have in LSCRP, but ordered using the additional information of time (and the scheduler's notion of priority).

Describing the evolution of the abstract machine in LSCRP essentially involves giving the evolution of an interpreter for the automatic execution of processes in LSCRP itself. The particular semantics given here elides timing constraints (which are mostly stochastic). Although a detailed treatment of timing constraints can be given within our framework, we argue that these are better treated as model properties than as control constructs to be relied upon. The result of all this would be a formalization very much less comprehensible than the original completely functional (and hence referentially transparent) implementation. The experience of the semantics community is that such interpreters are best written in purely functional languages, as we have done for (Core) Gnosis.

A framework for providing a logic programming-style semantics for Core Gnosis would be the Event Calculus (see, for example, [37]). Such an approach would fix the semantics at a slightly lower level of abstraction, with a consequent loss of natural representational choices, corresponding less directly and conceptually to the intended modelling paradigm, as described in Section 1.

4.2 Concrete LSCRP

The process calculus of Section 2 is deliberately abstract. It is intended that it may be instantiated in many different ways, depending on the application domain, just as different simulation languages are well-suited to different modelling problems. In this section, we present a concrete realization of LSCRP, that is appropriate for providing a simple but instructive translation of Core Gnosis models. The key steps in this realization are: to ground the notion of location; to ground the notion of resource with respect to location; to ground the set of actions with respect to location and resource; to define the modification and hiding functions, used to govern the evolution of LSCRP-states.

Suppose that some directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertices $\mathcal{V}$ and (unlabelled) edges $\mathcal{E}$ has been given, as has a set RNm, called the set of *resource names*.

A *location* is a subgraph of $\mathcal{G}$. Let $\mathbf{L}$ be the set of such locations. Let the distinguished element ℓ of $\mathbf{L}$ be the empty subgraph. A simple choice of location composition, which is adequate for our purposes here, takes for all locations L and L': $L \circ \ell = L = \ell \circ L = L \circ L$ are defined; for $L \neq \ell \neq L'$, the composite $L \circ L'$ is defined if and only if $L = L'$. For any L, let V_L be the set of vertices and E_L be the set of edges.

A *resource* is a function $R : \mathcal{V} \times RNm \longrightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers (including zero). Let $\mathbf{R}$ be the set of such resources. Let the distinguished element $\mathbf{e}$ of $\mathbf{R}$ be the map which takes every pair $(v, r) \in \mathcal{V} \times RNm$ to 0. Composition of locations is pointwise sum: that is, $(R \circ R')(v, r) = R(v, r) + R'(v, r)$ for all R, R', v and r. Note that resources can also be written as sets of triples of the form (v, r, n), such that $n \neq 0$.

The set Act of actions is generated freely from a set of atomic actions. Thus a general action a satisfies $a = \alpha_1 \cdots \alpha_n$, where the sequence of atoms $\alpha_1, \ldots, \alpha_n$ is determined uniquely, up to re-ordering. The identity action 1 is given by the case $n = 0$.

The atoms take the forms $\mathsf{get}(v, r, n)$, $\mathsf{put}(v, r, n)$, $\mathsf{forget}(v, v')$, $\mathsf{move}(v, v', r, n)$, and $\mathsf{recall}(v, v')$, the parameters v, v' ranging over $\mathcal{V}$, r ranges over RNm, and n ranges over $\mathbb{N}$. Thus our atoms are simple operations on a graph populated with resources, and are sharply divided into two kinds: those that manipulate resource, and those that change the underlying topology.

Define a corresponding family of parametrized partial functions. Each such function takes an argument consisting of a location L and a resource R. Fix the parameters v, v', r, and n. For get-actions, $\mu_{\mathsf{get}(v,r,n)}(L, R) \simeq L, R - \{(v, r, n)\}$ ($-$ here is the partially defined cancellation operation for this monoid). For put-actions, the modification $\mu_{\mathsf{put}(v,r,n)}(L, R)$ is defined iff $v \in V_L$, and $\mu_{\mathsf{put}(v,r,n)}(L, R) = L, R \circ \{(v, r, n)\}$, if defined. For move-actions, we have $\mu_{\mathsf{move}(v,v',r,n)}(L, R)$ is defined iff $(v, v') \in E_L$ and $R(v, r) \geq n$, and $\mu_{\mathsf{move}(v,v',r,n)}(L, R) = L, (R - \{(v, r, n)\}) \circ \{(v', r, n)\}$, if defined. For forget-actions, $\mu_{\mathsf{forget}(v,v')}(L, R) \simeq (V_L, E_L - \{(v, v')\}), R$. For recall-actions, $\mu_{\mathsf{recall}(v,v')}(L, R)$ is defined iff $v, v' \in V_L$, and $\mu_{\mathsf{recall}(v,v')}(L, R) = (V_L, E_L \cup \{e\}), R$, if defined.

Let a be an action, which may be given using atoms as $\alpha_1 \cdots \alpha_n$. For each atom α_i, there is a unique least resource R_i such that there is some L with $\mu(\alpha_i, L, R_i)$ defined. A well-defined modification is generated by taking $\mu(\alpha_1 \cdots \alpha_n, L, R)$ to be given by

$$\mu(\alpha_1, L, R_1) \circ \cdots \circ \mu(\alpha_n, L, R_n) \circ (\ell, R - (R_1 \circ \cdots \circ R_n))$$

for $n > 0$ and $\mu(1, L, R) = L, R$, for all L and R.

This provides all the information required to generate processes, states, and transitions in LSCRP, as indicated in Section 2, above.

4.3 Restricted Core Gnosis Models

Part of the Core Gnosis syntax is described in Section 3 above. We do not attempt to translate all models that may be written in the language. For the purposes of this section, we restrict the class of models in order to make a meaningful translation. For simplicity, we use an (evident) abbreviated syntax for Core Gnosis's resource-manipulating operations. The first restriction is that the guards in (all branches of) **select** statements contain only sequences of **claim** statements. The second restriction concerns the way in which the **move** action is used. In particular, we impose a tight discipline on the way in which resources are claimed and released, before and after being moved. Any **move** statement in a process declaration is allowed to occur in one of two ways: in a sequence (called a CMR-sequence), or in a sequence (called a SCMR-sequence) at the head of a branch of a **select** statement.

A *CMR-sequence* is a sequence of the form $B_1; B_2;$ **hold** $t; B_3$, where B_1 contains only **claim**s, B_2 contains only **move**s and B_3 contains only **release**s. For every **move**(r, l, l', n) in B_2 there is precisely one **claim**(r, l, n) in B_1, and precisely one **release**(r, l', n) in B_3, and there are no other **claim**s or **release**s in these sequences.

An *SCMR-sequence* is the same as a CMR-sequence, except that all of the **claim** statements occur inside the guard of a branch of a **select** statement. A simple SCMR-sequence may look like:

$$\textbf{select}[\textbf{claim}(r,l,n)]\{\textbf{move}(r,l,l',n); \textbf{release}(r,l,l',n) \ldots .$$

The point of this restriction is that CMR- and SCMR-sequences may be translated directly into concurrent compounds of move actions. However, this imposes a genuine restriction on the class of models. For example, we have lost the ability to write models with move-actions which start simultaneously (in simulation time) but finish at different times.

We further simplify here by assuming that conditionals have only Boolean guards.

4.4 A Translation

We now sketch a translation of programs of restricted Core Gnosis models into LSCRP-states. This is done using a partial function τ defined below. Let $\mathcal{M}$ be a Core Gnosis model, subject to the restrictions above. This is translated into the LSCRP-state $\tau(\mathcal{M}) = \tau_L(\mathcal{M}), \tau_R(\mathcal{M}), \tau_P(\mathcal{M})$.

The location $\tau_L(\mathcal{M})$ is the graph determined by the **location** and **link** declarations in the preamble of $\mathcal{M}$.

The resource $\tau_R(\mathcal{M})$ is the resource determined by the distribution of shares declared in the premable of $\mathcal{M}$.

It remains to describe the translation of the process declaration and launch phase of $\mathcal{M}$ into an LSCRP-process $\tau_P(\mathcal{M})$. This translation breaks down into three phases. First, to each ith process declaration P_i in $\mathcal{M}$ associate a process constant, say E_i, and to the launch phase associate a process constant E_0, and let E be this sequence of processes. Second, make a preliminary translation $\tau'(B_i)$ of the body B_i of all process declarations. Third, identify the images of the CMR- and SCMR-sequences in each $\tau'(B_i)$, and then collapse these into composites of move actions.

Let σ be any simple statement (**claim**, **release**, **move**, **forget**, **recall**, **hold**). Define a mapping from such σ to LSCRP actions:

$$\tau_S(\sigma) = \begin{cases} \mathsf{get}(l,r,n) & \text{if } \sigma = \textbf{claim}(l,r,n) \\ \mathsf{put}(l,r,n) & \text{if } \sigma = \textbf{release}(l,r,n) \\ \mathsf{move}(l,l',r,n) & \text{if } \sigma = \textbf{move}(r,l,l',n) \\ \mathsf{forget}(l,l') & \text{if } \sigma = \textbf{forget}(l,l') \\ \mathsf{recall}(l,l') & \text{if } \sigma = \textbf{recall}(l,l') \\ 1 & \text{if } \sigma = \textbf{hold}\,. \end{cases}$$

For the first part τ' of the translation the simple statements are translated straightforwardly into atomic actions, or else forgotten by translating to the tick action. In particular, timing constraints imposed by holds are not preserved.

If the sequence of statements is empty, then we translate it to the unit process 1 under τ'. Suppose that a sequence B_i takes the form $\sigma; B_i'$. Define, where γ_is are guards and b_is are booleans,

$$\begin{array}{ll} \tau'(\sigma; B_i') = & \\ \quad \tau_S(\sigma).\tau'(B_i') & \text{if } \sigma \text{ is simple} \\ \quad E_j \times \tau'(B_i') & \text{if } \sigma = \textbf{launch } P_j \\ \quad (\tau_S(\gamma_1).\tau'(B_1; B_i')) + & \\ \qquad\quad (\tau_S(\gamma_2).\tau'(B_2; B_i')) & \text{if } \sigma = \textbf{select}[\gamma_1]\{B_1\}\textbf{or}[\gamma_2]\{B_2\} \\ \quad (\tau'(B_1; B_i')) + (\tau'(B_2; B_i')) & \text{if } \sigma = \textbf{if}[b_1]\{B_1\}\textbf{or else}\{B_2\} \end{array}$$

This translation takes both CMR- and SCMR-sequences into sequences of gets, moves and puts (possibly with a tick action before the move). Furthermore, any move action is contained in such a sequence. Let us call such a sequence a GMP-sequence. The GMP-sequences may be automatically identified.

Define a further translation τ'' taking the family of $\tau'(B_i)$ to a new family of LSCRP-terms, by collapsing each GMP-sequence down to a single action consisting of a concurrent composite of all the move actions in that sequence. This gives a new recursive definition for the family of constants E. Define $\tau_P(\mathcal{M}) = E_0$. This completes the determination of the initial state $\tau(\mathcal{M})$ of the model $\mathcal{M}$. An example of this translation in action is given in the following section. The evolution of the model $\tau(\mathcal{M})$ is determined by the SOS-rules for LSCRP, similar to the versions in [8].

It is important to compare the behaviour of the model $\mathcal{M}$ with its translation $\tau(\mathcal{M})$, in order to give credence to the suggestion that this translation is helpful for thinking about $\mathcal{M}$. Let us call the **claim**s, **release**s, **forget**s, **recall**s, CMR- and SCMR-sequences the substantial events of Core Gnosis. Let us call the gets, puts, moves, forgets and recalls the *substantial events* of LSCRP. For the purpose of the following discussion, let a CMR- or SCMR-sequence be referred to as a **move**. The first important point to note about the translation is that substantial events of $\mathcal{M}$ are mapped to substantial events of $\tau(\mathcal{M})$ of the same kind. On the other hand, specific timing constraints have been forgotten. However, note that within any process the ordering of the substantial events is preserved. That is, if σ and σ' are simple events in a process declaration P_i, such that σ occurs before σ', then the associated LSCRP constant E_i will contain an instance of (the translation of) σ before an instance of σ'. Any trace of $\mathcal{M}$ in which an instance of P_i successfully performs both of the instances, will be such that σ occurs before σ', if both occur. Similarly, any path through the transition system generated by $\tau(\mathcal{M})$ will be such that if both instances of (the translations of) σ and σ' are present, then σ occurs before σ', if both occur. Of course, there may be additional instances of similar simple events generated by translations of other instances. This analysis extends to statements which occur in distinct process instances, where one is called by the other. In short, the sequencing aspect of control flow, and the ordering of substantial events is preserved under the translation. There is a caveat in that not all aspects of control-flow are preserved. In particular, the translations of the **select** and **if** statements are not completely faithful. For **select**, this should be rectifiable in a weighted extension [39] of LSCRP, since then the priority between branches may be preserved. For **if**, a faithful translation can be made by taking $\tau_P(\mathcal{M})$ to be a map from resources to processes. Finally, we remark that, for the other imperative constructs of Core Gnosis, not interpreted here, the Hiding operator of LSCRP will be required (see [10] for details).

4.5 Example

We now consider how the example from Section 3 is translated. Let this model be $\mathcal{M}$. The location component of $\tau(\mathcal{M})$ is the graph $\tau_L(\mathcal{M})$. The vertices of $\tau_L(\mathcal{M})$ are `Basic`, `Guard` and `Secure`. The pairs of locations, (`Basic`, `Guard`), (`Guard`, `Secure`), (`Secure`, `Basic`), and (`Guard`, `Basic`), give the edges. The resource $\tau_R(\mathcal{M})$ is given (as a set) by the set consisting of four triples of locations, resources, and quantities, (`Basic`, `tug`, 3), (`Basic`, `jetty`, 2), (`Secure`, `tug`, 3), and (`Secure`, `jetty`, 1).

Four process constants are used to translate the process declarations of $\mathcal{M}$. Let E_0 be the process constant that corresponds to the main process (launch phase), let E_1 correspond to `secureBoat`, E_2 correspond to `boat` and E_3 correspond to `check`. The process E_1 is given (using auxilliary process constants E_1' and E_1'' for the sake of readability) by:

$$\begin{array}{l}\mathsf{get}(\mathtt{Secure},\mathtt{jetty},1).\\ ((\mathsf{get}(\mathtt{Secure},\mathtt{tug},2).1.\mathsf{put}(\mathtt{Secure},\mathtt{tug},2).E_1')\\ +(\mathsf{move}(\mathtt{Guard},\mathtt{Secure},\mathtt{tug},2).E_1'))\end{array}$$

where E_1' is given by

$$\begin{array}{l}1.((\mathsf{get}(\mathtt{Secure},\mathtt{tug},1).1.\mathsf{put}(\mathtt{Secure},\mathtt{tug},1).E_1'')\\ +(\mathsf{move}(\mathtt{Guard},\mathtt{Secure},\mathtt{tug},1).E_1''))\end{array}$$

and where E_1'' is given by $\mathsf{put}(\mathtt{Secure},\mathtt{jetty},1).1.(E_1 \times 1)$.

The boat process is translated in much the same way. The translation of the `check` process is equally straightforward, but note that it contains an **if** statement and so the caveat above applies.

We then take $\tau_P(\mathcal{M}) = E_0 := E_2 \times (E_1 \times (1.E_3))$.

5. REASONING ABOUT MODELS; A LITTLE MORE META-THEORY

Process calculi such as SCCS, CCS, and the pi-calculus come along with associated modal logics [18, 25, 38]. Similarly, the calculus (L)SCRP has an associated modal logic, MBI [10, 31, 30]. The basic idea — deriving from Hennessy-Milner logic [18, 38] — is to work with a logical judgement of the form

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

read as 'relative to the available resources R, the process E has property ϕ'. Building on the ideas of bunched logic [29, 33, 32] and its application to Separation Logic [34, 21], MBI has, in addition to the usual *additive* connectives, $\top, \wedge, \rightarrow, \bot, \vee$, a *multiplicative* conjunction, $*$, and a multiplicative implication. Similarly, in addition to the usual additive quantifiers and modalities, MBI has multiplicative quantifiers and modalities.

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

$$\begin{array}{lll} R, E \models \langle a \rangle \phi & \text{iff} & \text{there exists } E' \text{ such that } R, E \xrightarrow{a} R', E' \\ & & \text{and } R', E' \models \phi \\[1ex] R, E \models [a]\phi & \text{iff} & \text{for all } E' \text{ such that } R, E \xrightarrow{a} R', E', \\ & & R', E' \models \phi. \end{array}$$

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

$$\begin{array}{lll} R, E \models \phi_1 * \phi_2 & \text{iff} & \text{there are } R_1 \text{ and } R_2 \text{ such that} \\ & & R_1 \circ R_2 \sqsubseteq R \text{ and there are } E_1 \text{ and } E_2 \\ & & \text{such that } E_1 \times E_2 \sim E, \text{ and} \\ & & R_1, E_1 \models \phi_1 \text{ and } R_2, E_2 \models \phi_2. \end{array}$$

Here the truth condition for the multiplicative conjunction requires the combination of resources from the truth conditions for its component formulae. Hiding can also be characterized [10].

These characterizations provide tools for reasoning about security properties such as joint-access (control), where two agents must both provide some resource for access to be granted, and authorization by delegation, where Hiding is used to establish a private channel; see [11] for the details of these and other examples.

With locations, a similar logical judgement is available [8]:

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

where the property ϕ of the process E holds relative to resources R at location L; that is, if a is an action guarding (the rest of) E, then $\mu(a, L, R)$ is defined.

We have talked a little about security and access control in the context of the processing of low-security and high-security boats by a system of docks. In such a setting, it is important to consider questions about the control of processes' (e.g., boats') access to resources. For example, eliding many details for the brevity of a short paper, consider a logical judgement

$$L, R, E \models \phi_1 * \phi_2,$$

where L, R, E is a suitable description of state of the docks example and where ϕ_1 is 'there are two free low-security tugs' and ϕ_2 is 'there are two free high-security tugs'. The fact that the model will satisfy such a multiplicative conjunction is one of the reasons why it allows low- and high-security boats to dock simultaneously. There are, however, states L', R', E' that, for example, have no $\mathsf{get}(\mathtt{Secure},\mathtt{tug},2)$-transition, because, say, insufficient tugs are available at the given location. For such a state,

$$L', R', E' \not\models \langle \mathsf{get}(\mathtt{Secure},\mathtt{tug},2) \rangle \top.$$

Returning again briefly to the meta-theory, it is important to have a useful relationship between a process calculus and its associated modal logic. The key idea originates with Hennessy, Plotkin, and Milner [18], and sets up a correspondence between process bisimulation and logical equivalence. In our setting, the strength of the available correspondence depends on the choice of process equivalence, but useful results are available, for significant fragments of MBI, for both the local and global equivalences. Details in [10, 8].

As currently formulated, MBI does not concern itself with stochastic properties of models. In contrast, Probabilistic Computation Tree Logic (see, for example, [36]) is intimately related to Markov chains, but lacks the structural analysis afforded by MBI. Consideration of adding stochastic properties to MBI represents challenging further work. A degree of model checking is available for MBI [9]. In contrast with PRISM [23], for example, stochastic issues are not considered, the focus being structural decompositions of models via the multiplicative connectives, such as $*$. Again, further work.

6. DISCUSSION

We have explained, at least informally, how semantic methods can be used to relate a conceptual modelling framework to a practical modelling language that is used to handle large-scale applications. We believe, with support from a good deal of experience (see, for example, [3, 2, 1, 41]), that the modelling discipline so engendered supports the construction of models having high integrity with respect to the system of interest. In addition to the further work mentioned above, there are two current foci of our work: one is the provision of better I/O, visualization, and experimental tools for the Gnosis engine; the other is exploring the application of our approach to topologically rich examples.

7. REFERENCES

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

[2] Y. Beres, J. Griffin, S. Shiu, M. Heitman, D. Markle, and P. Ventura. Analysing the performance of security solutions to reduce vulnerability exposure window. In *Proc. 2008*

Annual Computer Security Applications Conference, 33–42. IEEE Computer Society, 2008.

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

[4] G. Birtwistle. *Demos — discrete event modelling on Simula*. Macmillan, 1979.

[5] G. Birtwistle and C. Tofts. A denotational semantics for a process-based simulation language. *ACM ToMaCS*, 8(3):281 – 305, 1998.

[6] M. Collinson, B. Monahan, and D. Pym. Located Demos2k — towards a tool for modelling processes and distributed resources. Technical Report HPL-2008-76, Hewlett-Packard Laboratories, 2008.

[7] M. Collinson, B. Monahan, and D. Pym. An update to Located Demos2k. Technical Report HPL-2008-205, Hewlett-Packard Laboratories, 2008.

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

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

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

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

[12] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, 2005.

[13] O.-J. Dahl, E.W. Dijkstra, and C.A.R. Hoare. *Structured Programming*. Academic Press, 1972.

[14] Demos2k. `http://www.demos2k.org`.

[15] L. Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, 1986.

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

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

[18] M. Hennessy and G. Plotkin. On observing nondeterminism and concurrency. In *Proc. 7th ICALP*, LNCS 85, 299–309, 1980. Springer-Verlag.

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

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

[21] 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.

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

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

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

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

[26] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

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

[28] Möbius. `http://www.mobius.illinois.edu`.

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

[30] David Pym and Chris Tofts. A calculus and logic of resources and processes. *Formal Aspects of Computing*, 18(4):495–517, 2006. Erratum (with Collinson, M.) *Formal Aspects of Computing* (2007) 19: 551-554.

[31] 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 (Computation, Meaning, and Logic: Articles dedicated to Gordon Plotkin)*, volume 107, pages 545–587, 2007. Erratum (with Collinson, M.) *Formal Aspects of Computing* (2007) 19: 551-554.

[32] 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`.

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

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

[35] B. Ross. *An Algebraic Semantics of Prolog Control*. PhD thesis, University of Edinburgh, 1992.

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

[37] M.P. Shanahan. The event calculus explained. In *Artificial Intelligence Today*, LNAI 1600, 409–430, 1999. Springer-Verlag.

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

[39] C. Tofts. Processes with probability, priority and time. *Formal Aspects of Computing*, 6(5):536–564, 1994.

[40] G. Winskel. *The Formal Semantics of Programming Languages: An Introduction*. The MIT Press, Cambridge, Mass., and London, England, 1993.

[41] 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/`.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8631
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8631