Engineering Ecosystem Models: Semantics and Pragmatics *

Tristan Caulfield[1], Marius-Constantin Ilau[1], and David Pym[1,2]

[1] University College London, UK
[2] Institute of Philosophy, University of London, UK
{t.caulfield,marius-constantin.ilau.18,d.pym}@ucl.ac.uk

Abstract. In a world of ever-increasing complexity, the smooth functioning of society is critically dependent on our ability to understand and manage both individual systems and complex ecosystems of systems. Models, combined with tools to reason about them, can provide a way to do this. In order for rigorous reasoning about models to be possible, they must have a robust mathematical foundation, which must also support tools for the engineering principles — compositionality, interfaces, and local reasoning — that are required to enable the practical construction of models of ecosystems. In this paper, we present a vision for a system of modelling, based on the concept of distributed systems as a metaphor for ecosystems of systems, that captures these requirements. We describe a mathematical foundation, identify the engineering principles needed, and show how they can be built in a rigorous way that preserves the ability to reason when dealing with complex, large-scale ecosystem models. We illustrate our ideas with examples and briefly explain how they apply in a practical modelling project.

Keywords: distributed systems, ecosystems, modelling, compositionality, interfaces, local reasoning

1 Introduction

The environment we inhabit is constantly increasing in complexity and with this comes a need for new ways of understanding and thinking about the world that can manage this complexity.

There is a famous quote from Grace Hopper:

> 'Life was simple before World War II. After that, we had systems.'

Now, systems are pervasive. They interact with and depend on each other and we, in turn, depend on them. It has become important to be able to think about not just a single system, but also its interactions with other systems — it has become necessary to think of *ecosystems*.

* This work has been partially supported by the UK EPSRC research grant EP/R006865/1.

Models are a way of understanding and reasoning about the world. There is a lot of existing modelling technology and practice that is good for capturing models of individual systems. The necessary shift to the ecosystem view means that we must consider not just the construction of models of individual systems but also how they are constructed in order to interact with one another.

Rigorous mathematical reasoning about models is important. In addition to the usual simulation techniques, it is necessary to be able to reason about models supports tools such as model checking, formal specification, and correctness and verification techniques. In order for a system of modelling to support this type of rigorous reasoning, it must itself have a rigorous mathematical foundation.

Ecosystems are systems of systems or, to put it another way, they are compositions of interacting subsystems. Examples include the global financial infrastructure of banks and exchanges, transportation networks, cyber-physical systems such as or workplace IoT networks and semi-automated manufacturing lines, and distributed databases. Other examples include electricity distribution networks, package delivery (or any supply chain using GPS and tags), and artificial organs (smart heart pumps, etc.).

What properties, built on the rigorous mathematical foundations, should a system of modelling have in order to make it a useful tool for capturing and reasoning about models of ecosystems?

Two of the first things that are required are systematic accounts of composition and interfaces. Composition is an important concept because it captures the structure of systems in the real world, but also because it is a useful tool for modelling — decomposing a large system or ecosystem into smaller subsystems allows us to manage the complexity of models. Interfaces are essential for understanding how systems are composed together. They specify what it is that interacting systems require of each other.

The notions of composition and interface support a range of useful modelling tools. For example, substitution — replacing one component model with another — which in turn supports refinement, abstraction, and extension of models.

Another requirement is local reasoning [26, 38, 39, 42, 51], which enables the specification of the conditions required for models to be composed with one another. The components used in forming compositions are the interfaces between the models. As such, the conceptual and technical complexity of reasoning about composite models is controlled. In the absence of local reasoning, actions such as the substitution of one (possibly small) component model for another might involve reasoning about the entire ecosystem. In general, such a situation would be conceptually and technically intractable.

In order to reason about systems, a rigorous mathematical foundation is required. In order to reason about ecosystems, these concepts of composition, interface, and local reasoning must build upon this rigorous foundation. With these three properties and the additional modelling tools they enable, a system of modelling can support practical and efficient modelling and reasoning about ecosystems. The question then is, of the ways of thinking about models, which ones are good ones that provide the substrate for thinking about ecosystems?

Other modelling approaches have tried to support, at least partially, the properties described above. For example, [13] describes an extension to the UML language [46] that facilitates composition using two different strategies — merging and overwriting, representing variations of the definition of the composition relationship — better to align object-oriented model structures with the structure of requirement specifications, [19] describes an event algebra for the synchronization and composition of labeled transition systems applied to timed automata, and Alloy [27], can be seen as an attempt at constructing a modelling framework based on rigorous mathematical concepts, but which does not focus on the composition of large-scale models. Furthermore, general techniques such as [31] focus on model decompositions that follow a modular approach.

When considering composition for more practical purposes, a particularly interesting class of modelling tools focused on transformation can be found in the model driven development literature. Focusing on the Model Transformation Chains abstraction, [1], practical tools such as MTCFlow or UniTI [50] support the composition of such chain abstractions and allow therefore the production of reusable modelling artefacts not only directly, but also through executable code generation [29].

This shows that composition — both at the level of the languages used to describe the components of the models and at the level of the construction and manipulation of the models themselves — is a useful tool in a variety of different modelling approaches. Here, we seek to present a *generic*, semantically-rigorous modelling approach that encompasses these concepts and can be implemented directly.

Computer science provides the concept of a *distributed system* (e.g., [16]) as a paradigm that encompasses not only single systems, but also ecosystems of systems. We propose that a theory of distributed systems models, based on a rigorous mathematical foundation, can meet these requirements. Thus our contribution is to capture the inherent semantic structure of ecosystems and derive appropriate tools for constructing and reasoning about systems — that is, accounts of interfaces, substitution, and local reasoning — from that structure.

In Section 2, we discuss how we use the distributed systems metaphor as a basis for modelling systems and their (de)composition. In Section 3, we sketch the process-algebraic foundation of models and logical tools for reasoning about them. In Section 4, we explain how we model locations and how this gives rise to an analysis interfaces, substitution, and local reasoning about components and (de)composition through reasoning about interfaces.

In Section 5, we discuss a practical example of the deployment of our modelling approach using tools — implemented in the Julia language [28], available at [8] — that capture our modelling framework (cf. [10, 14]). Specifically, we consider a model of strategies for device recovery in the aftermath of security breaches that require the reinstallation of operating systems.

Finally, in Section 6, we discuss what is required to deliver fully our vision of a modelling framework.

2 Modelling Distributed Systems

The growth of interconnected networked systems led to the development of the concept and theory of distributed systems in computer science. This paradigm views systems as collections of components, in different locations, that work together to perform some task and communicate by sending information or messages over network connections.

This view is obviously very specific to its focus on computer systems, but its concepts can be taken more generally to provide a useful metaphor for understanding all types of systems and, finally, ecosystems. There are three key components upon which we draw.

- *Location* — Distributed systems naturally have a concept of different locations, which are connected to each other. In the CS view, components are present at different locations and connected by a network. In the more general view, locations can be physical (e.g., a room, a container), logical (e.g., an address in computer memory), or abstract (e.g., the location where a semaphore exists).
- *Resource* — Resources exist at locations and can move between them according to the locations' connections. In the general view, they can represent anything: physical objects, people, information.
- *Process* — Processes execute and manipulate resources as they do so.

These concepts can be used to build a representation of a system's structure and operation, but there is one more concept required: the environment in which the system operates.

- *Environment* — Environments capture the world outside of the system of interest and how the two interact.

This generalization provides concepts that can be used to model essentially any type of system, from physical to logical, or systems that incorporate both. We note also that these concepts are scale-free — they can be used at any level of abstraction or representation. However, we have not actually defined what it means to *build* a model using this distributed systems approach. This, too, is very flexible. Models can be largely conceptual, and use the ideas of location, resource, and process as a means to help think about the structure and behaviour of a system. Or distributed systems models can be mathematical, as we will show in the next section. Finally, this metaphor can be used to build executable models, in the spirit of Birtwistle's Demos [7], where a programmatic description of the system (in terms of locations, resources, and processes) is run to simulate the behaviour of the system [8–10, 14]. An early implementation of these ideas, Gnosis [14], has been used in significant commercial applications [4–6] derived from an industry-based research project [22].

The ability to compose models is important for modelling larger systems and ecosystems. During the modelling process, these systems can be decomposed into smaller parts, which can be modelled separately and then recombined, and which

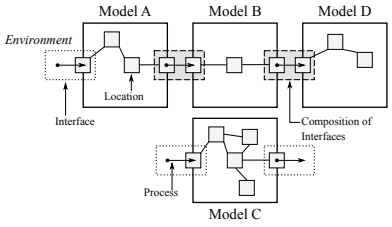

Fig. 1. Interfaces, Composition, and Substitution

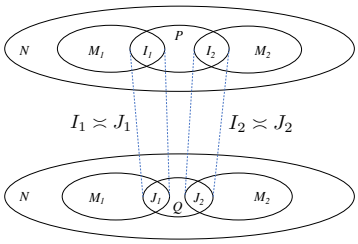

Fig. 2. Interfaces and Substitution

helps manage complexity. What does composition look like in the distributed systems approach?

We start with interfaces. These define, for a model, the locations, resources, and processes involved in a composition. For a composition of two models to be valid, the interfaces in both models must match. Figure 1 depicts three models which compose together. When models with interfaces are not composed, the environment generates the events expected by the interface; when composed, the environment is replaced by a model. Also shown is an example of substitution: `Model C` can be substituted for `Model B` as the interfaces of the two models match; this allows a modeller to refine or increase detail in parts of a larger model. We give more detail about interfaces and composition in Section 4.

Interfaces and composition seem to support the concept of local naturally. Obtaining such an account of reasoning requires a mathematical conception of the distributed systems metaphor on top of which interfaces and composition can be defined. Milner [36, 37] considers the concept of interface from the point of view of a quite abstract graphical theory of processes. Our notion is more directly grounded in the concept of a distributed system, but we conjecture that the approaches can be understood comparatively. Our approach is more directly concerned with the logical concept of local reasoning.

3 A Sketch of the Mathematical Foundations

We begin by giving a formal framework for capturing the distributed systems metaphor that we are proposing as a basis for a semantically and logically well founded framework for modelling ecosystems of systems in the absence of locations. The basic theory of processes and their associated logics is technically essentially determined by the interaction between processes and resources, with locations playing a significant conceptual rôle only when the model-engineering concepts of interface, substitution, and local reasoning are considered, which we do in Section 4. The results presented in this section for states R, E extend to states L, R, E [14].

3.1 Processes and Resources

Our starting points are Milner's synchronous calculus of communicating systems, SCCS [33] — perhaps the most basic of process calculi, the collection of which includes also CCS [32], CSP [23], Meije [17], and their derivatives, as well as the π-calculus [35], bigraphs [36] and their derivatives — and the resource semantics of bunched logic [20, 40, 42, 43]. 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:

$$E ::= 0 \mid a \mid a : E \mid \sum_{i \in I} E_i \mid E \times E \mid \dots$$

Most of the cases here, such as 0, action, action prefix, sum, concurrent product, and recursion, will seem quite familiar.

Mathematically, this notion of resource — which covers examples such as space, memory, and money — is based on (ordered, partial, commutative) monoids (e.g., the non-negative integers with zero, addition, and less-than-or-equals):

- each type of resource is based on a basic set of resource elements,
- resource elements can be combined, and
- resource elements can be compared.

Formally, we consider pre-ordered, partial commutative monoids of resources, $(\mathbf{R}, \circ, e, \sqsubseteq)$, where $\mathbf{R}$ is the carrier set of resource elements, $\circ$ is a partial monoid composition, with unit 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, presented in Plotkin's SOS style [41], is given by action prefix and concurrent composition, $\times$, exploits the monoid composition, $\circ$, on resources:

$$\frac{}{R, a : E \xrightarrow{a} R', E'} \quad \mu(a, R) = R' \qquad \frac{R, E \xrightarrow{a} R', E' \quad S, F \xrightarrow{b} S', F'}{R \circ S, E \times F \xrightarrow{ab} R' \circ S', E' \times F'}.$$

This (rather general [17,33]) notion of composition at the level of process does not explain the engineering concept of the composition of models, with its requisite notions of interface and substitution, that we discuss in the sequel.

Sums, which represented choices, recursion, and other combinators are defined in similar ways.

A modification function is required to satisfy some basic coherence conditions (in certain circumstances, additional structure may be required [2]): for all actions a and b and all resources R and S, and where $\simeq$ is Kleene equality,

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

This function specifies the *signature* of the model.

3.2 Logic

Process calculi such as SCCS, CCS, and others come along with associated modal logics [21, 34, 47, 48]. Similarly, the calculus sketched here has associated modal logic, MBI [2, 14, 15]. The basic logical judgement is 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 the bunched logic BI (e.g., [20, 40, 42, 43]) and its application to Separation Logic [26,44], MBI has, the usual *additive* connectives, $\top$, $\wedge$, $\rightarrow$, $\bot$, $\vee$.

These are all defined by semantic clauses of a satisfaction relation, where $\mathcal{V}$ is an interpretation of propositional letters in the usual way — see, for example, [49] — beginning as follows:

$$R, E \models \mathrm{p} \text{ iff } (R, E) \in \mathcal{V}(\mathrm{p})$$

In addition, MBI also has a *multiplicative* conjunction, $*$,

$$\begin{aligned} R, E \models \phi * \psi \;\text{ iff }\; & \text{there are } S, T \text{ and } F, G \text{ s.t. } S \circ T \sqsubseteq R,\ F \times G \sim E, \\ & \text{and } S, F \models \phi \text{ and } T, E \models \psi \end{aligned} \tag{1}$$

where $\sim$ is bisimulation (see, e.g., [33, 47, 48]) of processes, together with a multiplicative implication, $-\!\!*$. Note that the truth condition for $*$ — sometimes called a 'separating conjunction', since its conjuncts use separate resources — combines the resources from the truth conditions for its component formulae.

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 $R' = \mu(a, R)$):

$$\begin{aligned} R, E \models \langle a \rangle \phi \text{ iff there exists } E' \text{ s.t. } R, E \xrightarrow{a} R', E' \text{ and } R', E' \models \phi \\ R, E \models [a] \phi \text{ iff for all } E' \text{ s.t. } R, E \xrightarrow{a} R', E',\ R', E' \models \phi \end{aligned}$$

Similarly, in addition to the usual additive quantifiers and modalities, MBI has multiplicative quantifiers and multiplicative modalities [14, 15] (we elide the details of MBI's predication).

The basic connection between the process calculus and the logic is given by a form of van Benthem-Hennessy-Milner theorem that relates process equivalence, as given by bisimulation, and logical equivalence (e.g., [2, 14, 15, 21, 34, 47, 48]), for MBI, defined by

$$R, E \equiv_{\mathrm{MBI}} R, F \quad \text{iff} \quad \text{for all } \phi,\ R, E \models \phi \text{ iff } R, F \models \phi$$

For image-finite processes E and F and any R, [2, 14, 15],

$$R, E \sim R, F \quad \text{iff} \quad R, E \equiv_{\text{MBI}} R, F \tag{2}$$

Under stronger assumptions about the nature of resources [2], or with restrictions to the logic [14], this equivalence can be extended to pairs R, E and S, F of states with distinct resources.

Logics based on the language of MBI have proved valuable in program analysis — see the Infer tool [18] — partly by virtue of their deployment of local reasoning, based on the connective $*$.

4 Ecosystem Modelling and Local Reasoning

As we have discussed, a key component of the distributed systems metaphor that we propose as a basis for a semantically and logically well-founded framework for modelling ecosystems of systems is location, logical or physical.

In general, we can identify a few requirements for a useful notion of location in systems modelling. Specifically,

- a collection of basic locations,
- directed connections between locations,
- a notion of substitution, which respects connections, and
- (optionally) a (monoidal) product of locations (a technical requirement).

In the presence of locations, the judgements for the transition relation for model states and the associated logical truth, respectively, take the forms

$$L, R, E \xrightarrow{a} L', R', E' \quad \text{and} \quad L, R, E \models \phi,$$

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, but are otherwise defined similarly as above [2, 14].

4.1 Interfaces

The mathematical structure of models as described above provides the basis for the class of models that have been implemented in a systems modelling package [9, 10] for the Julia language. Closely following [9, 10], we describe interfaces more formally using well-motivated simplifications (that are, in fact, convenient to implement [10]) of the general semantic set-up [2, 14, 15].

Models in this methodology are designed to be composed with other models (Figure 1). Composition allows two or more models to be combined and the resulting behaviour explored. When models are composed there are interactions at the location, process, and resource levels, and the role of their intended environments is critical. Processes evolve (transition) and resources are moved between models at locations shared between the models. To enable composition, models need interfaces, which define the locations at which models fit together and

which actions, defined at appropriate locations within the interface, are party to the composition. Actions in the interface will nevertheless be able to execute only if the resources they require are available.

The locations and resources of a model are represented using a location graph, $\mathcal{G}(\mathcal{V}[\mathcal{R}], \mathcal{E})$, with a set of vertices, $\mathcal{V}$, representing the locations of the model, and a set of directed edges, $\mathcal{E}$, giving the connections between the locations. Vertices are labelled with resources $\mathcal{R}$. Rather than thinking of actions evolving processes, it is convenient to think of a process as a trace of actions — the history of actions that have evolved a process during the execution of the model. All of the actions in a model are contained in a set, $\mathcal{A}$, and process traces are comprised of these.

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

Writing I for the set of interfaces on a model, then an interface $I \in I$ on a model is a tuple (In, Out, L) of sets of input and output vertices, where $In \subseteq \mathcal{V}$ and $Out \subseteq \mathcal{V}$, and a set of located actions $L \subseteq \mathcal{L}$. The sets of input vertices and output vertices in interfaces must be disjoint; that is,

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

Given this set-up, we can define a model as follows:

Definition 1. *A model $M = (\mathcal{G}(\mathcal{V}[\mathcal{R}], \mathcal{E}), \mathcal{A}, \mathcal{P}, \mathcal{L}, \mathcal{I})$ consists of a location graph $\mathcal{G}$, a set of actions $\mathcal{A}$, a set of processes $\mathcal{P}$, a set of located actions $\mathcal{L}$, and a set of interfaces $\mathcal{I}$. (Note that we can still consider the evolution of model states to be described as above.)* □

Our notion of interface is related to Lynch and Tuttle's input/output automata [30].

Two models, M_1 and M_2 are composed using specific interfaces $I_{1,1}, \ldots, I_{1,j}, \ldots, I_{1,n} \in \mathcal{I}_1$ and $I_{2,1}, \ldots, I_{2,k}, \ldots, I_{2,m} \in \mathcal{I}_2$ using the composition operator, to give $M_1{}_{I_{1,j}}|_{I_{2,k}} M_2$, which is defined using an operation $\oplus$ on each of the elements of a model. First, we define the $\oplus$ operator for vertices and edges, $\mathcal{V}_1 \oplus \mathcal{V}_2 = \mathcal{V}_1 \cup \mathcal{V}_2$ and, for each $v \in \mathcal{V}_1 \oplus \mathcal{V}_2$, and then

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

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

To define the $\oplus$ operator for locations and interfaces, we first need to introduce some notation. The interfaces on a model are a set of tuples; for example, the interfaces of M_1: $\mathcal{I}_1 = \{(In_1, Out_1, L_1)_i\}$. A particular interface from $\mathcal{I}_1$ is

referred to as $I_{1,i}$, and the input locations from that interface are referred to as $In_{1,i}$, the outputs as $Out_{1,i}$, and the located actions as $L_{1,i}$.

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

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

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

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

with the constraint that $\mathcal{V}_1 \cap \mathcal{V}_2 = In_{1,j} \cup In_{2,k}$. (This constraint above represents a significant design choice in the definition of interfaces.) □

Proposition 1 ([10]). *$M_1 {}_{I_{1,j}}|_{I_{2,k}} M_2$ is a model.* □

Proposition 2 ([10]). *For any models M_1 and M_2, let $I_{1,2}$ be the subset of interfaces in $\mathcal{I}_1$ that compose with M_2. Composition of models is commutative and associative: $M_1 {}_{I_{1,j}}|_{I_{2,k}} M_2 = M_2 {}_{I_{2,k}}|_{I_{1,j}} M_1$ and $(M_1 {}_{I_{1,2}}|_{I_{2,1}} M_2)_{I_{1,3} \cup I_{2,3}}|_{I_{3,1} \cup I_{3,2}} M_3 = M_1 {}_{I_{1,2} \cup I_{1,3}}|_{I_{2,1} \cup I_{3,1}} (M_2 {}_{I_{2,3}}|_{I_{3,2}} M_3)$* □

So far, this definition of interface says little about how a model becomes animated. How this actually works is that a model is animated when events occur at its boundaries. As we have seen, models exist within environments and, as we have remarked, environments are captured within our framework stochastically. In fact, our treatment of environment — that is, that part of a model that is not captured in detail, using the distributed systems structure of locations, resources, and processes — is rather simple.

These issues will be clear by a simple example: the conveyor belt, represented using the language of our distributed systems modelling metaphor, and explain how it can decomposed into two component subsystems using an appropriate choice of interface. Figure 3 depicts a conveyor belt in which resources r are moved along from right to left, with *in* and *out* locations at either end.

The signature for this model, as described by its modification function, can be specified as follows:

$$\begin{array}{cc} \mu(move(r, in, l_1), in, r) = (l_1, r) & \mu(move(r, l_5, out), l_5, r) = (out, r) \\ \mu(move(r, l_k, l_{k+1}), l_k, r) = (l_{k+1}, r) & otherwise \quad \uparrow \end{array}$$

where, as usual, $\uparrow$ denotes 'undefined'.

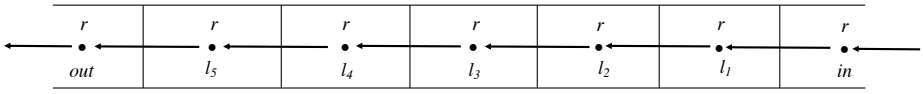

Fig. 3. A conveyor belt

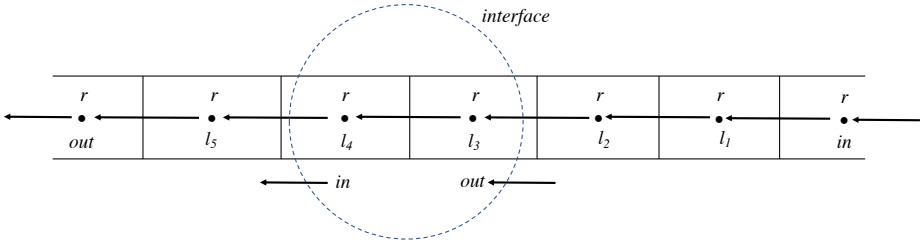

Fig. 4. A composite conveyor belt

The process-component of the model is then defined, recursively, as follows:

$$ConBelt ::= (move(r, in, l_1) : ConBelt \times move(r, l_1, l_2) : ConBelt \times \ldots \times move(r, l_5, out) : ConBelt) + 0$$

Then the system Ls , Rs , $ConBelt$, where we right Ls and Rs for the evident lists of locations and resources, describes the basic operation of a conveyor belt, as depicted in Figure 3. Either the belt moves with each section in lockstep or it stops (0 denotes termination).

Consider now a conveyor belt that consists of one belt passing on its items to another, perhaps because different machines are used to process the items on different belts. We can use our idea of interfaces to describe how to think of our $ConBelt$ as the composition of two, component, ConBelts.

The set-up is depicted in Figure 4. Here we can see that the conveyor belt can be understood as the composition of two such belts, the right-hand one of which has l_3 as it's *out* location, which then leads to the *in* location, l_4, of the right-hand one. The interface consists of the two locations, l_3 and l_4, together with their associated data.

4.2 Substitution

As we have briefly discussed, the construction of models of complex systems may require the substitution of one component model for another; for example, perhaps, to either increase or reduce the level of detail; or, perhaps, to explore a quite different design for part of a model; or, perhaps, to replace part of the environment with a specific model. The typical situation is, more or less, as depicted in Figure 2: a model N has components M_1 and M_2 connected by a model Q. We seek to replace Q with the model P.

For simplicity, denote the interfaces between M_1 and Q and Q and M_2 — formally defined as composites, as above — by J_1 and J_2, respectively. Similarly, suppose that P, which replaces Q, has interfaces I_1 and I_2 to M_1 and M_2.

For substitution to behave as required, what must we require of P, I_1, and I_2? We identify the following requirements: (i) the pairs of substituting and substituted interfaces should be able to simulate one another; (ii) the distributions of the events that are incident upon the corresponding boundaries of the interfaces should be the same, up to choices of parameters. These two conditions together give us what we need: let $\mathcal{D}_M(L)$ denote a set of pairs of probability distributions and locations in a model M. We write $d_1 \asymp d_2$ if d_1 and d_2 are distributions that are the same up to choices of parameters and extend this notation to sets $\mathcal{D}_M(L)$. Then, we require:

- $\mathcal{D}_N(In_{I_1}) = \mathcal{D}_N(In_{J_1})$ and $\mathcal{D}_N(Out_{I_1}) = \mathcal{D}_N(Out_{J_1})$
- $\mathcal{D}_N(In_{I_2}) = \mathcal{D}_N(In_{J_2})$ and $\mathcal{D}_N(Out_{I_2}) = \mathcal{D}_N(Out_{J_2})$
- for $i = 1, 2$, abusing notation a little, $I_i \asymp J_i$.

Consider the example of a substitution depicted in Figure 5, in which a small-scale road map of the roads in and out of a city is replaced by a larger scale map, which has more detail of the topography of the city. The relevant interfaces here are simply the points of contact between the roads within the city and their connections in the environment, together with their associated probability distributions.

The logic MBI allows us to assert some useful properties. For example, if $\mathcal{S}$ (i.e., some L, R, E) and $\mathcal{S}'$ (i.e., some L', R', E') denote states (we elide details) of the smaller and larger scale models, then we can write

$$\mathcal{S} \models \phi \quad \text{and} \quad \mathcal{S}' \models \phi'$$

where — writing c for a car, g_1, g_2, g_3 for the three city gates, and t and u for time periods, all as parameters for actions in the evident way — we can assert $\phi = [enter_{c,g_1}]\top \to (\langle exit_{c,g_2}\rangle\top \vee \langle exit_{c,g_3}\rangle\top)$ and

$$\phi' = [enter_{c,g_1}](\langle park_{c,t}\rangle\top \vee \langle gas_{c,u}\rangle\top) \to (\langle exit_{c,g_2}\top\rangle \vee \langle exit_{c,g_3}\rangle\top)$$

Here, just as in the transition from $\mathcal{S}$ to $\mathcal{S}'$, we give greater detail of the properties that may hold of a city location.

Note that the exit possibilities are not the only such possibilities (e.g., a car may remain in the town and never leave).

4.3 Local Reasoning

In this section, we introduce the concept of local reasoning, first introduced in the context of Separation Logic [26, 45, 52]. This conceptual design facilitates the ability to reason locally about the underlying components of an system or ecosystem.

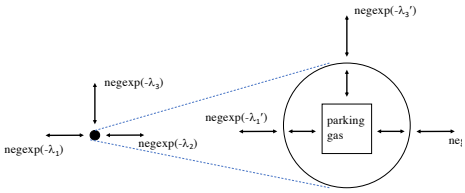

Fig. 5. Substitution

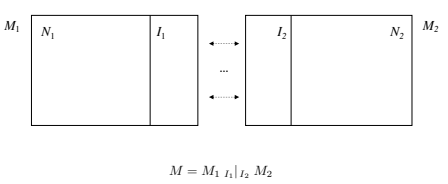

Fig. 6. Interfaces and Local Reasoning

The primary advantage of this is that the properties of an specific component in a decomposition of a model can be reasoned about without the need to reason about other components other than in respect of the interfaces to the specific component. Consequently, modularity (and substitution) are supported, with the conceptual and computational complexity of reasoning constrained.

With respect to local reasoning, we argue that the combination of the mathematical foundations sketched in Section 3 and the conceptual separation of components, as described above, offers the ability to focus analyses on specific components and simply state the relevant aspects for intercommunication at the level of interfaces.

We can identify here a local reasoning principle, or *frame rule* [26, 38, 39, 42, 51]. We begin by setting up some notation for the states of the various component models depicted in Figure 6:

- let the model $M = M_1{}_{I_1}|_{I_2} M_2$ have state $\mathcal{S}$;
- let the component models (of the composition of interest) M_i have states $\mathcal{S}_i$, respectively;
- let the submodels N_i have states $\mathcal{U}_i$, respectively; and
- let the interfaces I_i have states $\mathcal{I}_i$, respectively.

Now, using $\circ$ for composition of states, we assume the following, for $i = 1, 2$:

- $\mathcal{S}_i \sim \mathcal{U}_i \circ \mathcal{I}_i$, $\mathcal{S} \xrightarrow{a} \mathcal{T}$, and $\mathcal{I}_i \xrightarrow{a} \mathcal{J}_i$
- $a \# N_i \backslash I_i$; that is, that the action a is 'separated from' that part of the model N_i that is not coincident with the interface I_i in that the execution of a does not affect N_i.

Now, suppose that $\mathcal{U}_i \models \phi_i$, for $i = 1, 2$. Then we have the following frame rule:

$$\frac{\mathcal{J}_1 \models \psi_1 \quad \mathcal{J}_2 \models \psi_2}{\mathcal{T} \models (\phi_1 * \psi_1) * (\psi_2 * \phi_2)} \quad \begin{array}{l} \mathcal{S} \xrightarrow{a} \mathcal{T} \text{ and } \mathcal{I}_i \xrightarrow{a} \mathcal{J}_i \\ a \# N_i \backslash I_i \end{array}$$

This rule is sound with respect to bisimulation equivalence:

Proposition 3 (Soundness of the frame rule). *Suppose, for $i = 1, 2$, $\mathcal{J}_i \sim \mathcal{J}_i'$, $\mathcal{I}_1 \sim \mathcal{I}_i'$, and $\mathcal{S} \sim \mathcal{S}'$ and $\mathcal{T} \sim \mathcal{T}'$. Then $\mathcal{T}' \models (\phi_1 * \psi_1) * (\psi_2 * \phi_2)$.*

Proof sketch. By (2), we have that that, for $i = 1, 2$, $\mathcal{U}_i \models \phi_i$ and $\mathcal{J}_i \models \psi_i$. Then, note that separation condition, $a\#N_i \backslash I_i$, and the definition (1) of satisfaction for $*$ are respected by bisimulation. Finally, further applications of (2) then give the required conclusion.

To understand how all this works, consider again Figure 5 and suppose we have a model M for the part of the city that includes the parking and the gas station. That model is connected by interfaces — here again they are just point-to-point, respecting stochastic flows — to the rest of the more detailed model of the city. The facilities of the gas station and their operating capacities, which can be expressed logically, are properties of M that are independent of the model of the surrounding city. In this example, these properties correspond to the ϕ_is in the frame rule: separated by the multiplicative conjunction, $*$, they are invariant under changes to the surrounding model and the interfaces to it when the overall model evolves. The primary advantage of such a setting is that the modeller can confidently focus its analysis on a singular model component without the need to reason about its relationships with other components — the relevant aspects of intercommunication remain located at the interface level, acting as contracts that submodels have to fulfill in order for the composition to be possible.

Returning to our example of the conveyor belt, as depicted in Figures 3 and 4, suppose the two component belts are there to support two different sequences of operations:

- The right-hand belt performs actions op_1 and op_2 on the resources at locations l_1 and l_2, respectively;
- At l_3, in the interface, the correct completion of the operations op_1 and op_2 is verified;
- At l_4, in the interface, the readiness of the resources for the operations of the left-hand belt is verified;
- The left-hand belt performs the operation op_5 on the resource at l_5.

What can a frame rule say about this situation? First, we give the conveyor belt a bit more to do. Suppose that at locations l_1, l_2, and l_5, the operations op_1, op_2, and op_5, respectively, may — provided the machines servicing the belts are functioning correctly — be performed. Then, using MBI's modalities, as defined in Section 3.2,

$$in\,,\, r\,,\, move(r, in, l_1) : ConBelt \models [move(r, in, l_1)]\ \langle op_1 \rangle \top$$

since $move(r, in, l_1)$ takes our focus to location l_1, at which point op_1 may be performed, and nothing else happens to the resource r until it moves to l_2. Similarly,

$$l_1\,,\, r\,,\, move(r, l_1, l_2) : ConBelt \models [move(r, l_1, l_2)]\ \langle op_2 \rangle \top$$

These properties hold of that part of the right-hand belt that lies outwith its interface to the left-hand belt.

A similar logical judgement holds for the left-hand belt:

$$in\ ,\ r\ ,\ move(r, l_4, l_5) : ConBelt \models [move(r, l_4, l_5)]\ \langle op_5 \rangle \top$$

Again, this property holds independently of properties of the right-hand belt.

Here we are assuming, for simplicity, that the belt(s) cannot stall or otherwise prevent the passing of resources from one location to the next — such a possibility would break our separation condition. This assumption, however, provides a clue to the use of the frame rule.

So far, our discussion of interfaces has been purely at the operational level: locations, actions, and so on. But the composition of models through interfaces might also be subject to some requirements that certain properties of the component models hold. That is, the composition

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

might be made subject to conditions, following the notational convention set out above, as follows: $\mathcal{I}_{1,j} \models \phi_{1,j}$, for each j and $\mathcal{I}_{2,k} \models \phi_{2,k}$, for each k, specifying the required properties of the output from one model and input to the other.

Within our conveyor belt(s) example, we can set up an example of such a situation. Let $Op_1(r)$ and $Op_2(r)$ be propositions that denote that the resource has received the operations op_1 and op_2, respectively. Then we may impose the conditions

- On the output of right-hand belt:

$$l_3\ ,\ r\ ,\ move(r, l_3, l_4) : ConBelt \models Op_1(r) \wedge Op_2(r)$$

- On the input to the left-hand belt:

$$l_4\ ,\ r\ ,\ move(r, l_4, l_5) : ConBelt \models Op_1(r) \wedge Op_2(r)$$

In order to check that the two conveyor belts can be composed, we need only check that the resources arriving at l_3 have received the operations op_1 and op_2. Of course, the left-hand belt may require that the resources it receives also carry a certification that these operations have been performed. Such a certification might be delivered as part of a check at l_3 and a verification at l_4:

- Check: $l_3\ ,\ r\ ,\ move(r, l_3, l_4) : ConBelt \models Check(op_1, op_2)$
- Validate: $l_4\ ,\ r\ ,\ move(r, l_4, l_5) : ConBelt \models Validate(op_1, op_2)$

Again, checking these properties would be independent of those parts of the belts outwith their interfaces.

5 Applying the Framework

Modelling tools based on the framework we have described above have been deployed in a range of applications that aim to support information security decision-making such as [10], [9], [11], [12], [24] or [25].

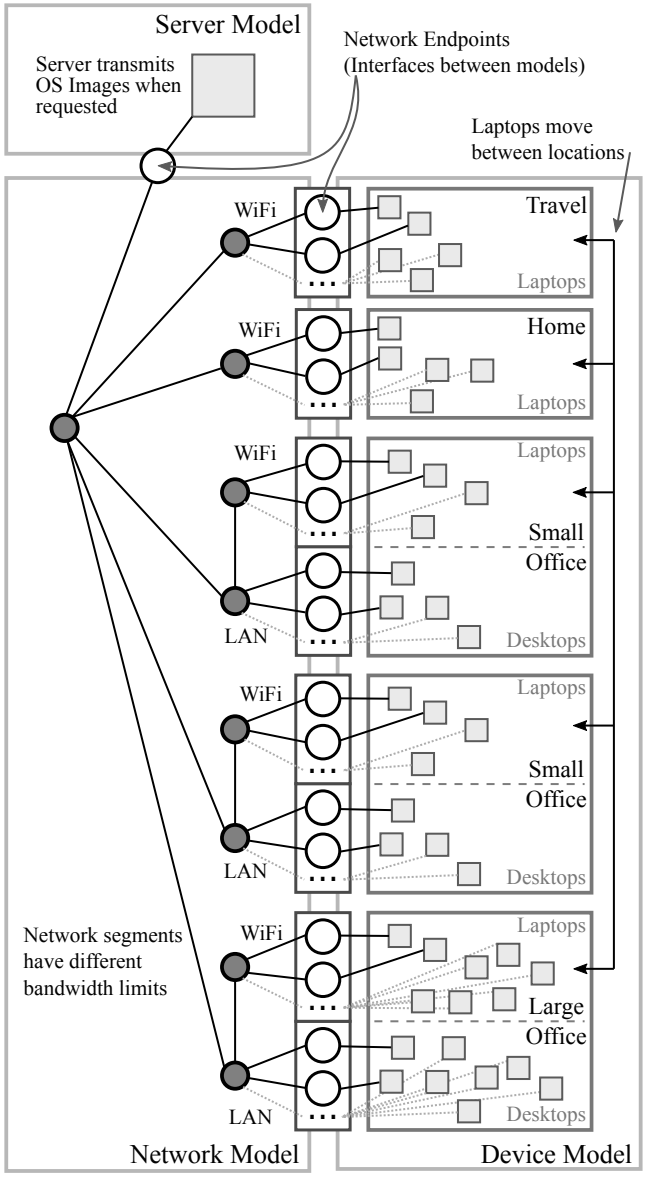

Fig. 7. The Architecture of the Recovery Model

Recently — using tools, implemented in the Julia language [28] and available at [8], that capture the framework we have described — we have considered the problem of organizational recovery under ransomware attacks and constructed a model for exploring different attack scenarios for the purpose of increasing organizational resilience. This work, has been carried out in collaboration with industry colleagues, is reported in [3].

Here, we use this problem of organizational recovery to discuss how our analysis of compositionality and local reasoning can be applied to the models generated in this work. To discuss the method of application of these ideas in generality and depth would be a substantial piece of work. For brevity, here we must restrict our discussion to a short conceptual account of how the above described notions of interfaces, composition, substitution and local reasoning are utilized in an implemented model. We begin by presenting an overview of the model.

The architecture of the organizational recovery model is presented in Figure 7. The model represents the process of a device becoming compromised by ransomware and being reinstalled to a fresh state to recover. Devices can use different methods of recovery: reinstalling from a USB stick, using a recovery image embedded in the device, or fetching an operating system image over the network and using that to recover. The model is used to explore different recovery strategies under different types of attack. For example, a quickly-spreading attack that compromises a large number of devices that use network recovery might cause a serious delay in recovery because of the bandwidth limitations of the network; in such a situation, having some embedded recovery devices might be beneficial as this can reduce the demand on the network.

In Figure 7, it can be seen that the overall model is actually built out of three separate sub-models: a network model, a server model, and a device model. The device model represents an organization's physical architecture and the devices that may be present in those locations. For example, an organization may be spread over a number of different physical locations, such as offices, with devices such as laptops and desktops present in each of those. The physical locations also may include places such as coffee shops, hotels, and homes, where work devices may travel to. The movement of devices is included in the model. The model also includes devices' connections to the network or internet — *network endpoints* are locations in the model that represent a connection to the network, over a LAN or WiFi connection — and the different methods for recovery.

The server model represents a remote server that responds to devices' requests for operating system images. When the server receives a request, it transmits an operating system image back to the device that requested it. The server has a network endpoint that represents its connection to the network.

The network model sits between the device model and the server model. It defines the network and internet architecture that the devices and server connect to. The network model also specifies network endpoints, and these correspond to the network endpoints that are present in the device and server models. These network endpoints are *interfaces* between the models. The network model ac-

cepts network traffic resources placed into the endpoints by transmitting devices, and delivers them, after a delay, to the receiving network endpoint. The delay is based on the size of the data, the bandwidth of the different network segments, and how much other traffic is also being transmitted at the same time.

These three sub-models compose together at the interfaces — the network endpoints — to become a larger, complete model. In the complete model, when a device is compromised and initiates recovery over the network, it 'transmits' a request for an operating system image to the server over the network. It does this by moving a resource representing the request into the network endpoint. The network model waits for such resources to arrive in the endpoints; when one arrives, it begins the transmission process by calculating the transmission time (which can vary and is updated based on network utilization by other devices). When the transmission time has elapsed, the network model delivers the request resource to the server's endpoint. The server waits for these requests to arrive from the network and responds to them by transmitting an operating system image resource back to the device that requested it over the network using the same process. When the device receives the image, it completes the recovery process by installing it.

When considering substitution, two options are possible: substituting a model with another model or substituting an environment component with a model. For example, one might consider a single network storage point as too simplistic and wish to employ a more complex storage model that includes different servers, load balancing or back-up procedures. In such a case, the modeller can focus on developing the desired architecture as long as the output network packets have the same signature as the ones produced by the model to be substituted. Furthermore, perhaps the modeller considers that additional focus has to be placed on different types of malware that might affect the organization. In that case, the environmental process that stochastically triggered malware events can be developed into a fully structured model and then substitute that part of the environment.

Furthermore, we describe a plausible scenario for the application of local reasoning. Although the original model goal was to illustrate the different recovery times of different mechanisms and strategies, one might imagine that our model setting can be used for simulating the impact of different data loss policies. In that case, the modeller need not worry about having to fully re-implement the models to target data loss or perform a complete re-validation procedure. Different modelling approaches can tend to either take this consistency for granted or perform time consuming re-validation steps, whereas in our case, the frame rule described above ensures that the previously considered invariant components remain invariant as long as they don't interact with any of the new development additions.

To see how our approach to compositionality and local reasoning can be applied to such a setting, let's consider a stripped down, somewhat abstracted, version of the composite model. Here, for simplicity, we assume that composed models — Server–Network and Network–Device — have interfaces that are iden-

tical; that is, in terms of our definition in Section 4.1, this amounts to the interfaces from each of the models that are used in a composition being identical in each model. The simplified composite model is depicted in Figure 8 .

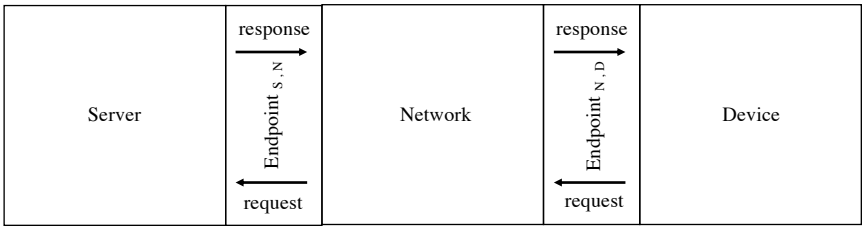

Fig. 8. A Simplified Recovery Model

By way of an example, consider the composition of the device model and the network model. A device may request an image from the server by sending a request from the endpoint interface for transmission over the network to the server. The server's response, including the image, is transmitted over the network and received at the Endpoint interface, which now holds the image for receipt by the device:

$$\text{Endpoint}_{N,D} \overset{response}{\longrightarrow} \text{Endpoint}'_{N,D}$$

The availability of the image that is appropriate for the device can be expressed by a logical assertion such as

$$\text{Endpoint}'_{N,D} \models \text{Image}_X \wedge \text{Device}_X$$

where X denotes the required OS, so that Image_X denotes a proposition asserting that an X image is available and Device_X denotes that the device requires the X image.

Note that the separation condition, as defined above,

$$response \# \text{Device} \backslash \text{Endpoint}_{N,D}$$

holds. Consequently, applying the frame rule, we can substitute a different device model, Device$'$, provided

$$\text{Endpoint}'_{N,D'} \models \text{Image}_X \wedge \text{Device}'_X$$

can be verified.

As the examples above and in Section 3 illustrate, the way in which composition, substitution, and local reasoning have been defined and supported by our modelling approach brings a certain set of advantages. Composition ensures model consistency. Substitution and the use of interfaces offers great flexibility

while at the same time conserving model consistency. Lastly, local reasoning offers the confidence to focus model analysis on singular components and ability to better manage complexity in a timely manner when considering practical implementation.

6 Discussion and Future Work

We have explained the need for a compositional engineering methodology — using the ideas of interfaces and substitution — for constructing models of complex ecosystems. Our methodology is mathematically rigorous, itself building on a rigorous approach to modelling distributed systems, which are seen as a metaphor for the structure of complex ecosystems.

Models based on the distributed systems metaphor have found significant application in commercial contexts. As we have described in Section 2, an early implementation of these ideas, Gnosis [14], formed the basis of the application of an industry-based modelling project [22] to major clients [5,6]. However, Gnosis lacks the compositional engineering and reasoning tools that we have proposed in this paper, with only a very limited model-checker implemented. A compositional approach for executable models was subsequently developed in [8, 10] and used in recent, as yet unpublished, work by some of us that has involved the application of our ideas to resource-allocation problems in hospitals, with a focus on emergency care and its follow-up. Here, the location-based, modular approach appears to be particularly valuable, as it allows different clinical departments, which must interact with one another, to be modelled and reasoned about independently.

While our current tools support the distributed systems modelling framework that we have described in Section 2, including the notion of interface, for a systematic approach to the construction of executable models much work remains to be done to deliver our vision of a fully developed practical methodology:

- tools to support the design and specification of models that support our theory of interfaces and substitution;
- tools to support logical reasoning — in particular, tools based on model checking — about models; and
- tools to support local reasoning about the compositional structure of models.

These additions will enable the construction of large-scale models of ecosystems that can be executed and reasoned about. This will allow us to better represent, understand, and make decisions about the complex systems in the modern world.

References

1. Camilo Alvarez and Rubby Casallas. Mtc flow: A tool to design, develop and deploy model transformation chains. In *Proceedings of the Workshop on ACadeMics Tooling with Eclipse*, ACME '13, New York, NY, USA, 2013. Association for Computing Machinery.
2. G. Anderson and D.Pym. A calculus and logic of bunched resources and processes. *Theoretical Computer Science*, 614:63–96, 2016.
3. Anonymous authors. Modelling Organizational Recovery. Submitted, 2021.
4. Adrian Baldwin, Yolanta Beres, Geoffrey B. Duggan, Marco Casassa Mont, Hilary Johnson, Chris Middup, and Simon Shiu. Economic methods and decision making by security professionals. In *Schneier B. (eds) Economics of Information Security and Privacy III*. Springer, New York, NY, 2012. 978-1-4614-1981-5.
5. Y. Beres, Jonathan Griffin, S. Shiu, Max Heitman, David Markle, and Peter Ventura. Analysing the performance of security solutions to reduce vulnerability exposure window. *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 33–42, 2008.
6. Yolanta Beresnevichiene, D. Pym, and S. Shiu. Decision support for systems security investment. *2010 IEEE/IFIP Network Operations and Management Symposium Workshops*, pages 118–125, 2010.
7. G. Birtwistle. *Demos — discrete event modelling on Simula*. Macmillan, 1979.
8. T. Caulfield. SysModels Julia Package. Available at Available at `https://github.com/tristanc/SysModels`. Accessed 10/05/2021.
9. T. Caulfield and D. Pym. Improving security policy decisions with models. *IEEE Security and Privacy*, 13(5):34–41, 2015.
10. T. Caulfield and D. Pym. Modelling and simulating systems security policy. In *Proc. SimuTools*, 2015.
11. Tristan Caulfield and Andrew Fielder. Optimizing time allocation for network defence. *Journal of Cybersecurity*, 1(1):37–51, 2015.
12. Tristan Caulfield and Simon Parkin. Case study: predicting the impact of a physical access control intervention. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust*, pages 37–46, 2016.
13. Siobhán Clarke. Extending standard uml with model composition semantics. *Science of Computer Programming*, 44(1):71–100, 2002. Special Issue on Unified Modeling Language (UML 2000).
14. M. Collinson, B. Monahan, and D. Pym. *A Discipline of Math.Systems Modelling*. College Publns., 2012.
15. M. Collinson and D. Pym. Algebra and logic for resource-based systems modelling. *Math. Structures in Comput. Sci.*, 19:959–1027, 2009.
16. George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley; 3rd edition, 2000.
17. R. de Simone. Higher-level synchronising devices in Meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985.
18. Facebook. Open-sourcing Facebook Infer. Available at `https://engineering.fb.com/2015/06/11/developer-tools/open-sourcing-facebook-infer-identify-bugs-before-you-ship/`. Accessed 10/05/2021.
19. Elie Fares, Jean-Paul Bodeveix, and Mamoun Filali. Event algebra for transition systems composition application to timed automata. *Acta Informatica*, 55:363–400, August 2018.

20. D. Galmiche, D. Méry, and D. Pym. The Semantics of BI and Resource Tableaux. *Math. Structures in Comput. Sci.*, 15:1033–1088, 2005.
21. M. Hennessy and G. Plotkin. On observing nondeterminism and concurrency. In *Proceedings of the 7th ICALP*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309. Springer-Verlag, 1980.
22. Hewlett-Packard Laboratories. Security Analytics. Available at `https://www.hpl.hp.com/news/2011/oct-dec/security_analytics.html`. Accessed 10/05/2021.
23. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, London, 1985.
24. C. Ioannidis, D. Pym, and J. Williams. Information security trade-offs and optimal patching policies. *European Journal of Operational Research*, 216(2):434–444, 2011. doi:10.1016/j.ejor.2011.05.050.
25. C. Ioannidis, D. Pym, and J. Williams. Fixed costs, investment rigidities, and risk aversion in information security: A utility-theoretic approach. In Bruce Schneier, editor, *Economics of Security and Privacy III*. Springer, 2012. Proceedings of the 2011 Workshop on the Economics of Information Security.
26. S.S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *Proc. POPL*, 2001.
27. Daniel Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, 11(2):256–290, April 2002.
28. julia. `http://julialang.org`.
29. Anneke Kleppe. Mcc: A model transformation environment. In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture – Foundations and Applications*, pages 173–187, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
30. Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989.
31. Qin Ma, Pierre Kelsen, and Christian Glodt. A generic model decomposition technique and its application to the eclipse modeling framework. *Softw. Syst. Model.*, 14(2):921–952, May 2015.
32. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer Verlag, 1980.
33. R. Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25(3):267–310, 1983.
34. R. Milner. *Communication and Concurrency*. Prentice Hall, New York, 1989.
35. R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.
36. Robin Milner. Bigraphs as a model for mobile interaction (invited paper). In *ICGT 2002, First International Conference on Graph Transformation*, volume 2505 of *LNCS*, pages 8–13. Springer, 2002.
37. Robin Milner. *The Space and Motion of Communicating Agents*. Cambridge University Press, 2009.
38. P. O'Hearn. Resources, concurrency, and local reasoning. *Theor. Comput. Sci.*, 375(1–3):271–307, May 2007.
39. Peter W. O'Hearn, John C. Reynolds, and Hongseok Yang. Local reasoning about programs that alter data structures. In *Proceedings of the 15th International Workshop on Computer Science Logic*, CSL '01, page 1–19, Berlin, Heidelberg, 2001. Springer-Verlag.
40. P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

41. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Dept., Aarhus University, Aarhus, Denmark, 1981.
42. David Pym. Resource semantics: Logic as a modelling technology. *ACM SIGLOG News*, 6(2):5–41, April 2019.
43. D.J. Pym, P.W. O'Hearn, and H. Yang. Possible Worlds and Resources: The Semantics of BI. *Theor. Comput. Sci.*, 315(1):257–305, 2004.
44. John Reynolds. Separation logic: A logic for shared mutable data structures. In *Proc. LICS*, 2002.
45. John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, LICS '02, pages 55–74, Washington, DC, USA, 2002. IEEE Computer Society.
46. James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.
47. Colin Stirling. *Modal and Temporal Properties of Processes*. Springer Verlag, 2001.
48. Johan van Benthem. *Logical Dynamics of Information and Interaction*. Cambridge University Press, 2011.
49. D. van Dalen. *Logic and Structure*. Springer, Berlin, third edition, 1997.
50. Bert Vanhooff, Dhouha Ayed, Stefan Van Baelen, Wouter Joosen, and Yolande Berbers. Uniti: A unified transformation infrastructure. In Gregor Engels, Bill Opdyke, Douglas C. Schmidt, and Frank Weil, editors, *Model Driven Engineering Languages and Systems*, pages 31–45, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
51. Hongseok Yang and Peter O'Hearn. A semantic basis for local reasoning. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures*, pages 402–416. Springer Berlin Heidelberg, 2002.
52. Hongseok Yang and Peter O'Hearn. A semantic basis for local reasoning. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures*, pages 402–416, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.