Multi-Level Modeling and Distributed Agent-Based Inference: the Role of System Structure

Gregory Provan*
University College Cork
Computer Science Department
Cork, Ireland
g.provan@cs.ucc.ie

Abstract

We describe a framework for modeling and performing inference on complex systems, based on an Attributed Programmed Graph Grammar (APGG) meta-model. We outline this APGG framework, and how it can be used. In particular, show how we can use the underlying system structure as an organising principle for model specification and analysis. In addition, we examine the attribution we assign to the higher-level systems models, namely a constraint-based language, and how a distributed agent-based approach can regulate inference performed asynchronously on several multi-level models.

1 Introduction

A complex system consists of a collection of interacting, autonomous and/or semi-autonomous sub-systems that must interact to accomplish a task. A complex system must be treated holistically, as reducing it to simple sub-systems or components loses a great deal of its identity. Systems of this type are difficult to characterise and model mathematically. Moreover, it is quite difficult to develop a model that achieves a reasonable degree of mathematical precision and computational tractability, but is still faithful enough to the system being modeled.

We address the modeling of complex physical systems in which the structure can be clearly defined. Examples of such systems include the WWW, mechanical and electrical systems, and biological systems. For such systems, we specify the functionality using a variety of modeling formalisms at multiple levels of specification detail; however, we will show how we can use the underlying system structure as an organising principle for model specification and analysis.

* Supported by SFI grant 04/IN3/I524.

It is very important to address the two critical aspects of a complex system, the system's structure (or topology) and the system's functional interactions. For example, in assessing the criticality of a puncture wound in a mammal, it is important to know both the severity of the wound and its location (e.g., proximity to major blood vessels or organs). Many researchers have described how the multiple levels of description of a complex system can be specified, e.g. [6, 23]; however, the literature stresses system functionality and multi-level functional specification and transformation, and does not focus enough on the role that structure and structural invariance across multiple levels can play in such multi-level functional specification and transformation.

The study of complex systems now indicates that all complex systems have structures that can be defined in terms of power-law graphs [17]. Moreover, certain structural properties appear to be invariant across multiple scales [5, 11, 12]. In this article we will show how a simple structural invariant can be used for modeling and inference across multiple scales.

This article adopts the Attributed Programmed Graph Grammar (APGG) framework [3]. The APGG framework can represent both structure and function within a single framework: the hierarchical graphical structure can capture the system topology, and attributes assigned to the graph's nodes and edges can capture arbitrary functional interactions. In addition, the APGG framework can also be used computationally for a variety of purposes, such as a basis for verification and validation of system properties, or as a method for defining transformations that occur within a system.

We summarise the attributes that can be applied to a complex system by the *level* of the system. The high level description of a complex system describes the system in an abstract sense, where we outline properties such as the global objectives of the system, and can perform constraint optimisation. We use a constraint language [22] for this

level. At the other extreme, the low (or detailed) level, we specify the complex system dynamics of the system at a fine level of granularity, and can simulate the continuously-varying behaviours of the systems. We use dynamical systems, stochastic dynamical systems or hybrid systems models for describing such attributes.

Given the system structure, we focus on the meta-level model representation and the higher-level inference models. Our use of graph grammars for meta-modeling is similar to that in [9, 21, 24]; our contributions reside not in the notion or use of meta-modeling, but in our model transformations and agent-based framework for multi-level model inference. Our novel contributions include (1) the use of a fixed-topology meta-model to generate a topologically- and functionally-consistent set of inference models at multiple levels, and (2) a distributed, agent-based method for integrating the results of the simultaneous, asynchronous computation of the various multi-level models.

In the following we will describe a meta-model that consists of two main parts: (1) a structural component describing the system topology, and (2) a semantic component describing the attributes, such as functional transformations. We first describe a motivating example, and then outline the APGG representation that provides a framework for the entire model. We then we outline high-level models that represent particular attributes for a complex system, and a method for performing inference on these models in a distributed fashion.

2 Approach

Our proposed methodology (CN+) develops models for complex systems using three main steps: (1) we use a GUI to develop a graphical representation of the system, $\mathcal{M}_{GUI}$; (2) we transform $\mathcal{M}_{GUI}$ into an APGG meta-model representation for the complex system $\mathcal{M}_A$, using transformation τ_1; and (3) we generate a set of multi-level inference models via appropriate model transformations, τ_2. Figure 1 depicts the steps of model generation and analysis using CN+.

The meta-model captures the topology that is consistent across several levels of the model. It also captures the overall functional transformations, which may be described using different languages at the different levels, but in an abstract sense are equivalent specifications. For example, a set of differential equations and their associated qualitative differential equations [14] are equivalent in an abstract sense.

The meta-model is used to generate three hierarchically-ordered classes of model, which we call mission-, process- and machine-level models (in order of decreasing model abstraction). Given this hierarchy of models, we group the models based on function and topology, and use an agent-based framework to accomplish mission-level optimisation tasks in a distributed and autonomous fashion.

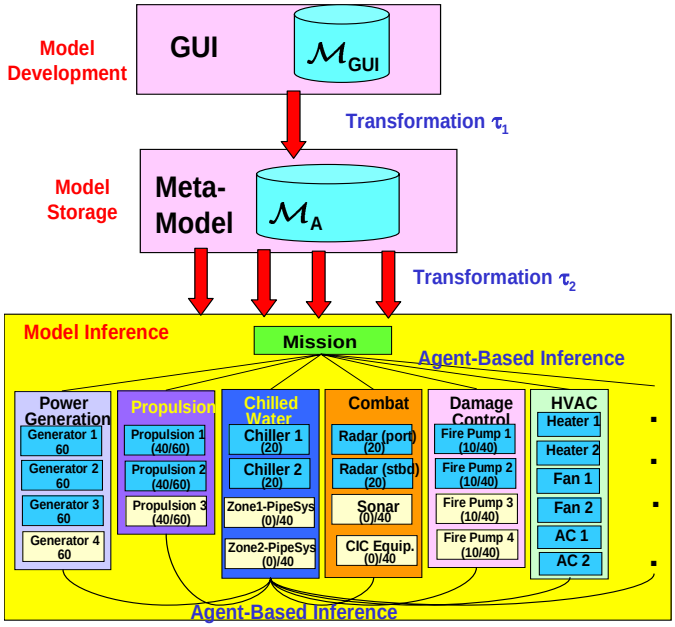

Figure 1. Steps of model generation and analysis using CN+. Top-level hypergraph decomposition of ship, showing the hyperedges between the the Chilled Water subsystem and other sub-systems.

The meta-model serves as a generator for all models that perform inference (using the notion of generator from [21, 24]), and as a representation that will re-generate new models given changes to the system. This capability is very useful during the model design phase, during which system each re-design must be mapped to all sub-systems. In addition, this can be useful when performing inference on the models. For example, if significant damage is done to a part of the system, this damage will be recorded in the meta-model, and new inference models that reflect the changes necessitated by the damage will be re-generated. This ensures that inference using consistent models is enforced across all levels.[1]

The meta-model's structure is based on the systems's underlying invariant structure. We will define this invariant structure in Section 4.2, and provide examples in Section 3. Intuitively, the invariant structure is the structure of a system that is invariant across multiple levels of abstraction. For example, the road network between San Francisco and Denver has many side-roads at the state and local level, but the invariant is the interstate link between San Francisco and Denver, which is preserved at interstate, state and local levels.

[1]Less substantial changes may entail just a modification of inference, and this is facilitated by the meta-model, or may be covered just be the distributed agent framework.

3 A Simple Shipboard System

This section provides an example of a shipboard system, which we use to explain our notions of complex hierarchical systems, and of multi-level inference upon such systems.

3.1 Overall Ship System

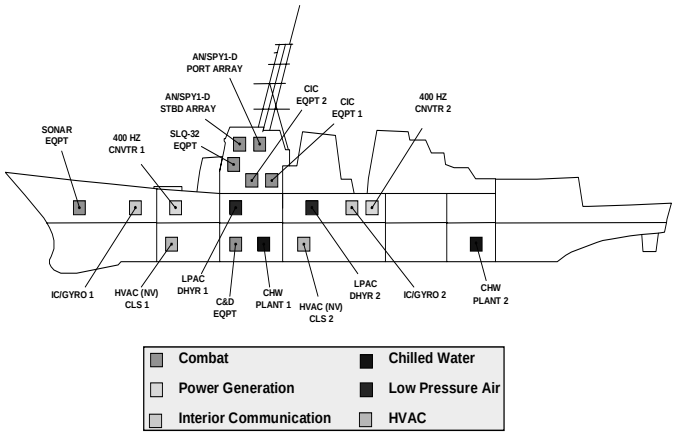

Figure 2. Location and functional classification of several major subsystems of a ship.

Figure 2 presents a functional view of a simple shipboard engineering system. The primary ship functions include propulsion, power generation, heating/ ventilation/ air-conditioning (HVAC), combat (radar and sonar systems), etc.

The functional representation of a shipboard engineering system can be decomposed into three primary abstraction levels: mission control, process and machine.

Mission control: The highest level in the control architecture is represented by a mission control block, which sets priorities and performance objectives for all shipboard engineering systems based on the overall mission goals. For example, in a damage control situation, resources would be shifted from less critical services, such as drinking water production, to fire-fighting systems. In functional terms, this level serves as a global inter-process coordinator and user level interface.

At this high level of abstraction, the typical computational tasks, which can be generalised as optimisation tasks, would include resource allocation and scheduling. High-level system models would be defined to solve this class of task; we use constraint-based models for this purpose.

Process: Each process-level function block provides a general ship service (*e.g.*, power generation, propulsion, damage control, chilled water, material handling, HVAC, etc.), and will reconfigure its system operation based on the performance objectives set by the higher level mission control. The process blocks also cooperate with other process blocks to satisfy global goals.

Machine: At the lowest level, the operation of a machine is represented by a machine block/model (*e.g.*, Generator 1 block in the power generator process). Controllers at this level monitor and control individual machine units (*e.g.*, a propulsion unit) to maintain machine availability and achieve the machine setpoints requested by the process-level.

At this low level of system description, tasks would include system estimation and low-level control. Dynamical systems models are typically used to describe systems at this level. For example, we could specify a dynamical model of fluid flow in the chilled water sub-system.

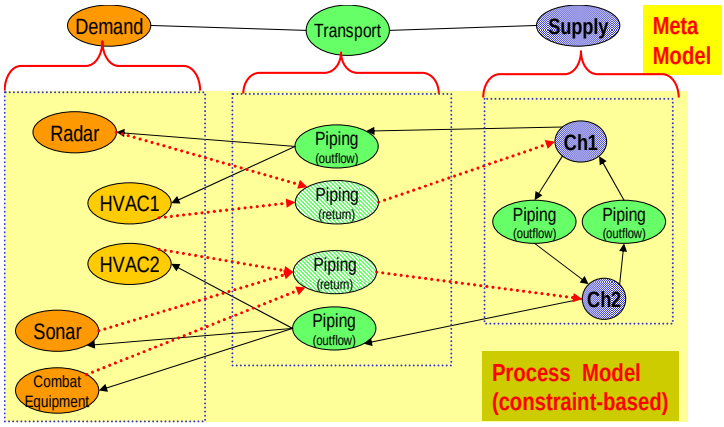

Figure 3. Meta- and constraint-graph models of chilled water sub-system. We show the meta-model at the top, with three main components, supply, transport and demand. The main part of the figure shows an abstract view of the constraint-graph process-model, which can be grouped into three sub-graphs representing demand, transport (piping/valve) and supply sub-systems.

3.2 Chilled Water Sub-System

We now focus on the Chilled Water sub-system (ChW), in order to be able to describe the models that can be gen-

erated for various tasks that must be performed with this sub-system.

Figure 4 shows a simplified diagram of a ship's Chilled Water sub-system, and how it is connected via piping systems to several of the ship's other sub-systems, such as power generation, combat (radar and sonar systems), HVAC, etc.

The ChW system can be decomposed into three main classes: the *supply* components (chillers), the *fluid transport* components (pipes and valves), and the *demand* components (loads on the system, such as combat sub-system, propulsion sub-system, etc.). The structural invariant for the ChW system consists of a simple graph shown at the top of Figure 3. The graph consists of nodes for *supply*, *transport*, and *demand*, since the ChW system provides chilled water (the supply) to a set of ship sub-systems (the demand), through a complex set of pipes distributed throughout the ship (the transport mechanism).

The ChW *supply components* consist of two chillers (Ch1 and Ch2), shown at the right of Figure 4. The two chillers are the two sources of chilled water, and typically serve as redundant units; they can work together to satisfy high loads. Figure 4 depicts the *piping elements*, showing the supply pipes as dotted blue lines, and the return pipes as solid red lines. We have decomposed the piping system into a collection of contiguous piping sections, noted P_1 through P_8 in the figure. In addition, we have assigned the valves into five valve groups, noted V_1 through V_5. The grouping is done for computational purposes, i.e., to guarantee fast inference on each individual group, and to enable an agent to be assigned to each group. We denote the *demand components* (loads on the system, such as combat sub-system, propulsion sub-system, etc.), using L_1 through L_9.

3.3 Shipboard System Model

We build a formal model of the shipboard system by focusing on two aspects of the system: the system topology, and the system functionality.

In terms of the topology, we can build a hypergraph model $\mathcal{G}$ of the interconnections of the components of the shipboard system. The set of inference models in Figure 1 depict a top-level view of such a hypergraph, which is defined around a functional decomposition of the system. In the figure we focus on the hyper-edges between the the Chilled Water sub-system and other sub-systems, and omit the other hyper-edges for the purposes of exposition.

This structure captured by the hypergraph model $\mathcal{G}$ provides the basis for building various models of aspects of the ship. For example, at the top-level this model focuses on resource allocation and scheduling, and $\mathcal{G}$ captures the relations between supply and demand entities. At the bottom-level, where we focus on building a dynamical systems

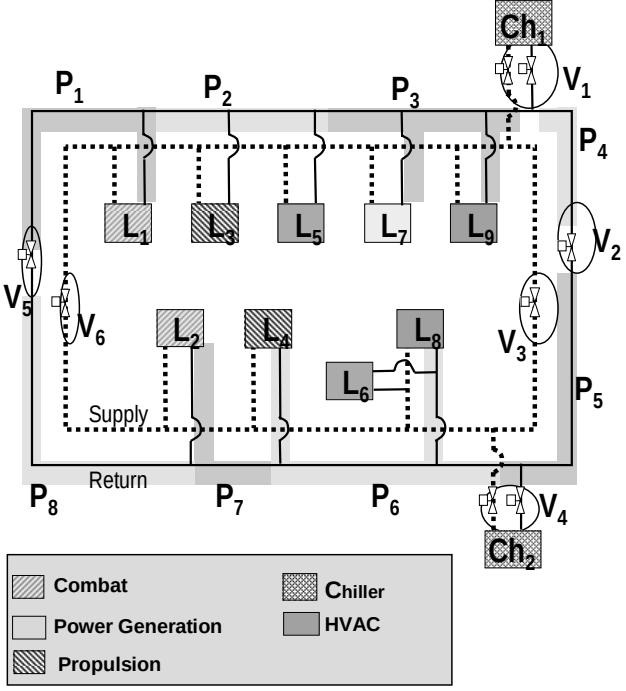

Figure 4. Engineering schematic of the Chilled Water sub-system. The lines in the figure represent pipes, whose flow (from the chillers to the loads) is mediated by valves. The various classes of sub-system are tabulated underneath the schematic.

model to simulate fluid flow within the chilled water system, the identical hypergraph model $\mathcal{G}$ provides the basis for building a dynamical systems model, in that it specifies the fluid flow relations, e.g., which pumps in the chillers force chilled water through pipes to various loads.

4 Attributed Programmed Graph Grammars

Graph grammars are used for systems modeling because of their ability to naturally depict complex situations in an intuitive, perceptually-clear manner. Graph grammars originally were derived to generalise formal language theory, as based on strings and the theory of term rewriting [19]. Graph grammars introduce a precise mathematical methodology for locally transforming structures represented as graphs.

An attributed graph consists of two components, (1) a structural component and (2) a semantic component describing the attributes. We adopt the notation of Bunke [3] for describing attributed graph grammars, and extend it by defining a hierarchical graph structure.

Definition 1 *Let Z and W be two alphabets for labeling the nodes and the edges. An unattributed structure graph (s-graph) is a 3-tuple $G = (V, E, \lambda)$ where:*

- *V is the finite set of vertices;*
- *$E = \{E_w\}_{w \in W}$ is a tuple of relations such that $E \subseteq V \times V$ for each $w \in W$;*
- *$\lambda : V \rightarrow Z$ is the node labeling function*

A pair $(V, V') \subseteq E_w$, is interpreted as a directed edge from node V to node V' having label w.

4.1 Hierarchical Structure

We adopt the hierarchical hypergraph representation of [8]. This representation defines hyperedges, called *frames*, which contain hypergraphs that can themselves be hierarchical, with an arbitrary depth of nesting.

A hierarchical graph consists of a graph, the root of the hierarchy (which we call the primal graph), a designated subset of its edges, called the frames, and a mapping assigning to each frame its contents, which are either a hierarchical graph or a variable. We formalize this as follows:

Definition 2 *Let V be a set of variables represented by nodes in a graph. The class $\mathcal{G}(V)$ of hierarchical graphs with variables in V consists of triples $\mathcal{G} = (G, \Gamma, \chi)$, where G is a graph, Γ is the set of frames, and $\chi : \Gamma \rightarrow \mathcal{G}(X) \cup V$ assigns to each frame $\gamma \in \Gamma$ its contents $\chi(\gamma) \in \mathcal{G}(V) \cup V$. Each hierarchical graph $\mathcal{G}_i(V) \in \mathcal{G}(V)$ is defined inductively as follows:*

- *A triple $\mathcal{G} = (G, \Gamma, \chi)$ as above is in $\mathcal{G}_0(V)$ if $\Gamma = 0$.*
- *For $i > 0$, $\mathcal{G} \in \mathcal{G}_i(V)$ if $\chi(\gamma) \in \mathcal{G}_{i-1}(V)$ for every frame $\gamma \in \Gamma$.*

4.2 Structural Invariance

The graph grammar specification of Section 4.1 provides only the syntax for a system whose hierarchical structure is known. In this section we summarise our method for defining the *semantics* of the hierarchical structure and the structural invariants that can generate the hierarchy.

Little work exists in the literature on the semantics of hierarchical system structures. One area in which the semantics of system structure and its impact on functionality has been addressed is in reverse engineering. For example, Cremer [4] has developed a set of tools for reverse engineering systems by using the system structure to extract functionality. This methodology has been applied to telecommuncations systems in [15]. This work takes a similar approach to ours, in that it addresses systems with structural invariants, and makes use of those invariants. However, rather than discovering the system's structure and functionality, we assume that we know these, and use them for multi-level system specification and analysis.

A second area has been the theoretical analysis of graph invariants across multiple scales, e.g., [10]. We adopt the notion of structure-preserving graphical hierarchical decompositions of [10]. These notions of structure-preserving decompositions provide a clear semantics for the transformations across different representational levels, and of the properties that are preserved by the transformations.

We denote a collection V_ξ of nodes of $\mathcal{G}(V)$ as a cluster ξ. The nodes in a cluster ξ are said to be *covered* by ξ; we denote $v \in V$ being covered by ξ using $v \prec \xi$. This notion of covering provides a semantic counterpart for the syntactic notion of a frame γ and its contents $\chi(\gamma)$.

If we restrict the base hierarchical graph $\mathcal{G}_0$ to be a tree, then we can define a set of structure-preserving hierarchical decompositions [10] with respect to a property $\mathcal{P}$. Moreover, these decompositions can map precisely onto the set Γ of frames of $\mathcal{G}(V)$.

For example, consider the meta- and constraint-graph models of Figure 3. The meta-graph is said to be a *contraction* of the constraint-graph in which the *demand* node consists of the cluster {*radar, HVAC1, HVAC2, sonar, combat-equipment*}. Similarly, the meta-graph nodes for *transport* and *supply* consist of clusters of corresponding constraint-graph nodes. In performing this clustering process, the property $\mathcal{P}$ that we preserve is the type attribute of a node, i.e., whether the type is *demand*, *supply* or *transport*. Note that this contraction process preserves several properties, for example graph connectivity, across the levels: if nodes v_i and v_j are connected (by a path) in the constraint-graph, and $v_i \prec \xi_i$ and $v_j \prec \xi_j$, then ξ_i and ξ_j will be connected (by a path) in the meta-graph. This is true for the nodes of Figure 3.

We can formalise this as follows. Ξ is a covering of graph $\mathcal{G}$ iff for each $v \in V$ $\exists$ unique $\xi \in \Xi$ such that $v \prec \xi$. We can use this notion of covering to define semantic conditions under which $\mathcal{G}_j(V)$ is a contraction graph of $\mathcal{G}_i(V)$. Hence a graph $\mathcal{G}_j(V)$ is a *contraction* of $\mathcal{G}_i(V)$ if $i < j$ and the nodes of $\mathcal{G}_i(V)$ are covered by those of $\mathcal{G}_j(V)$, $i < j$. In this shipboard application, by using coverings based on the type attributes, we can preserve the supply-transport-demand properties of the hypergraph model for all levels $\mathcal{G}_i(V)$, for $igeq0$.

4.3 Attributed Graphs

This section describes how we assigned attributes to the hierarchical structure of $\mathcal{G}$. Let A and B be two sets of attributes, where an attribute is a function associating attribute values with nodes or edges.

Definition 3 *An attributed graph (a-graph) is a tuple $\mathcal{G} = (V, E, \lambda, \Gamma, \chi, \alpha, \beta)$ where:*

- *$V, E, \lambda, \Gamma, \chi$ are the same as in Definitions 1 and 2;*
- *$\alpha : N \rightarrow 2^A$ is a function that assigns to each node a set of node attributes;*

- $\beta = \{\beta_w\}_{w \in W}$ *is a tuple of functions* $\beta_w : E_w \rightarrow 2^B$ *associating a set of edge attributes with each w-edge.*

According to Definitions 2 and 3, an attributed graph consists of two components:

- a syntactic or structural part described by $(V, E, \lambda, \Gamma, \chi)$, denoting the underlying hierarchical graph (s-graph), and
- a semantic part given by the node and edge attributes (α, β).

A graph (s-graph/a-graph) $\mathcal{G}'$ is a subgraph of $\mathcal{G}$, i.e., $\mathcal{G}' \subseteq \mathcal{G}$, if all nodes and edges belonging to $\mathcal{G}'$ also belong to $\mathcal{G}$. Additionally, corresponding nodes and edges must have identical labels and attributes. If $\mathcal{G}' \subseteq \mathcal{G}$ let $\mathcal{G} \setminus \mathcal{G}'$ denote the graph that remains after removing $\mathcal{G}'$ from $\mathcal{G}$.

Definition 4 *The edges between* $\mathcal{G}'$ *and* $\mathcal{G} \setminus \mathcal{G}'$ *are denoted as the embedding of* $\mathcal{G}'$ *in* $\mathcal{G}'$, *EMB*$(\mathcal{G}', \mathcal{G})$.

Definition 5 *A production is a 5-tuple* $p = (\mathcal{G}_l, \mathcal{G}_r, T, \pi, F)$ *where.:*

- $\mathcal{G}_l$ *and* $\mathcal{G}_r$ *are s-graphs (left - and right-hand side of the production)*
- $T = \{L_w, R_w | w \in W\}$ *is the embedding transformation with* $L_w, R_w \subseteq N_l \times V_r$*;*
- $\pi : T \rightarrow$ *[TRUE,FALSE) is the applicability predicate*
- F *is a finite set of partial functions* $f_a : V_r \rightarrow D_a$ *and* $f_b : E_r \cup EMB(\mathcal{G}_r, g) \rightarrow D_b$ *with* $a \in A$ *and* $b \in B$.

f_a *denote the node attribute and* f_b *the edge attribute transfer functions.*

Function F describes the embedding of $\mathcal{G}_r$ into $\mathcal{G} \setminus \mathcal{G}_l$ The attribute transfer functions f_a and f_b change the attributes of $\mathcal{G}_r$. π summarises the conditions that have to be fulfilled by the left-hand side in order to apply the production p.

Definition 6 *The direct derivation of a graph* $\mathcal{G}'$ *from a graph* $\mathcal{G}$ *by means of a production* p, *denoted* $\mathcal{G} \stackrel{p}{\rightarrow} \mathcal{G}'$, *is defined as follows:*

1. *Check a possible occurrence of* $\mathcal{G}_l$ *as a subgraph of* $\mathcal{G}$ *and the applicability predicate* π*; if both TRUE goto 2.*
2. *Replace* $\mathcal{G}_l$ *by* $\mathcal{G}_r$ *in* g.
3. *Embed* $\mathcal{G}_r$ *in* $\mathcal{G} \setminus \mathcal{G}_l$, *using T.*
4. *Change the attributes of* $\mathcal{G}_r$, *according to* f_a *and* f_b.

Example 1 Figure 5 depicts the results of applying a production p to a sub-graph of the shipboard domain. This production entails adding a mode for a failure mode of the piping system to a graph. In this resultant graph, the constraint over the variable for the load now includes the load, the piping variable and the failure-mode variable.

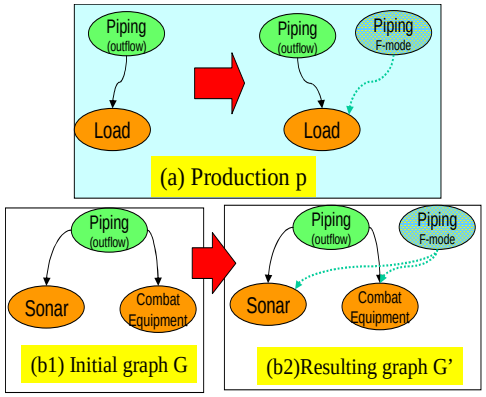

Figure 5. (a) shows a production p, and (b) shows the application of p to an initial graph, together with the resulting graph.

Definitions 1 to 6 describe an attributed graph grammar (AGG). Nothing has been said about the order in which the productions should be applied. We can extend this representation to include an ordering for applying productions, using a control diagram [20]. This control knowledge is used to expand an AGG to a programmed graph grammar, thereby improving the efficiency of the model transformation [13].

5 GUI- and Meta-Model Specification

This section describes the representations and transformations that we adopt for the GUI- and Meta-Model. We summarise the model representations in Table 1. The GUI- and meta-model are quite similar: the GUI model has a similar hierarchical framework as that of the meta-model; however, the GUI model attributes are GUI-specific, and consist of GUI data $\wp$, that describe the size, colour, position, etc. of GUI objects.

Model	Representation
$\mathcal{M}_{GUI}$	$\langle \mathcal{B}, \mathcal{E}(\mathcal{B}), \wp \rangle$
$\mathcal{M}_A$	$\langle \mathcal{B}, \mathcal{E}(\mathcal{B}), \mathcal{F} \rangle$
$\mathcal{M}_{CSP}$	$\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$

Table 1. Different model representations.

Model construction and analysis consists of two main stages, as shown in Figure 1: (1) model construction, using the GUI; and (2) model inference, using the multi-level models. This process involves two main classes of model transformation:

- transform the GUI model to the meta-model, using transformation $\tau_1 : \mathcal{M}_{GUI} \rightarrow \mathcal{M}_A$, and

- transform the meta-model to the multi-level model(s), using transformation $\tau_2 : \mathcal{M}_A \rightarrow \mathcal{M}_{CSP}$.

The *meta-model* represents the system components, their connectivity, and the functional transformations performed by each component. We are interested in systems in which the hierarchical and topological structure plays an important role, and the meta-model must capture that. Our meta-model differs from other meta-models, such as that of [24], which uses an entity-relation model for the meta-model representation. In addition, we assume that the primary model structure remains fixed, in contrast to the dynamic model structure addressed in [1].

For ease of exposition, we present our representation as an abstracted version of the DEVS framework of [26].[2] We represent this information using the tuple $\langle \mathcal{B}, \mathcal{E}(\mathcal{B}), \mathcal{F} \rangle$, where $\mathcal{B}$ represents the system components, $\mathcal{E}(\mathcal{B})$ is the set of interconnections, and $\mathcal{F}$ is the set of component functional transformations, which are explicitly defined in a formal language. We assume that each system entity transforms a set of inputs into a set of outputs using a functional transformation.

This model also incorporates the hierarchical information present in the model. We do not assume that this model is in a form for directly performing inference on it, but that it contains the information necessary to transform it into a version on which inference can be performed.

The entities of our meta-model, or blocks $\mathcal{B}$, are denoted using the tuple $(I, O, f, \mu, \mathcal{E})$, representing inputs I, outputs O, function f, mode μ, and connections $\mathcal{E}$. We denote a function f by $O = f(I, \mu)$. f may be represented as a (flat) set of constraints, or it may be represented as a nested set of sub-blocks and interconnections among them

The primary thing that we need to specify for a block (and its transformation f) is whether it contains any sub-blocks as part of its representation. We define a block (function f) as *primitive* if it contains no sub-blocks (sub-functions).

A block B is given by $(I, O, f, \mu, \mathcal{E})$. For a composite block, the inputs I and outputs O remain the same, but we have to specify the function f and modes μ differently, as described below.

Primitive function: We define a composite function for block B in terms of the sub-functions defined in B for each output port. If we assume that B has k output ports, then for output port i, we have a set of inputs $\mathcal{I}_i$, interconnections E_i, and function f_i, such that, for $i = 1, ..., k$:

$$O_i = f_i(\mathcal{I}_i, \mu),$$

$$E_i = \{(\iota, O_i) \mid \iota \in \mathcal{I}_i\}.$$

[2] Our framework is fully compatible with the DEVS representation.

Composite function: A composite function has an internal set of sub-blocks, so we have to define the function definitions for output ports in a different manner. We define a composite function for block B_i in terms of the sub-functions and their order of composition. This in turn is computed from B_i's sub-blocks, $\mathcal{B}_i$, and their interconnectivity $\mathcal{E}_{\mathcal{B}_i}$.

6 Multi-Level Inference Models

6.1 Chilled Water System Mission-Model

This section summarises the attributes we adopt for the higher levels of a complex system model, constraints. We embed a constraint attribute model within the graph grammar framework. To do this, we use the constraint graph representation [7], since this specifies constraints within a graphical framework.

6.1.1 Constraint Problem Formulation

We first define a constraint problem [22], and then show the constraint-graph formulation of this representation.

Definition 7 *A Constraint Satisfaction Problem (CSP)* $\Pi = \langle \mathcal{V}, \mathcal{D}, \mathcal{C} \rangle$ *consists of:*

- *a set of variables* $\mathcal{V} = \{v_1, ..., v_n\}$*;*
- *for each variable* v_i*, a finite set* $\mathcal{D}_i$ *of possible values (its domain);*
- *and a set* $\mathcal{C}$ *of constraints restricting the values that the variables can simultaneously take. A constraint* c_i *is a relation defined on a subset* V' *of the variables, that is,* $c_i \supseteq \times_j \{v_j : v_j \in V'\}$.

A *solution* to a constraint problem is an assignment that satisfies all the constraints. An *assignment* is a mapping θ from the set of domain variables $\mathcal{V}' \in \mathcal{V}$ to their corresponding domains, i.e, $\theta(v_i) \in D_i$ for $v_i \in \mathcal{V}'$. The set of all assignments on $\mathcal{V}'$ is denoted by $\Theta_{\mathcal{V}'}$. In the following we need the notion of parents of a node in a graph. Given a directed graph $G(V, E)$, we define the parents $\nu(v)$ of vertex $v \in V$ as those $v' \in V \setminus v$ such that $(v', v) \in E$.

We now define a constraint system over a directed graph G [7]. Note that that this is a non-hierarchical model, since no hierarchical constraint representation, nor algorithms that could make use of hierarchies, exist.

Definition 8 *An attributed constraint graph* $G(V, E)$ *is a graph over vertices* V *and edges* E *in which the attributes are assigned to* V *such that:*

- *if* $v \in V$ *has no parents in* G*, then the constraint is unary;*

- *if $v \in V$ has parents in G given by $\sigma(v)$, then the constraints over v, c_v, will involve variables $v \cup \sigma(v)$ and no others.*

6.1.2 Valuation and Optimisation

In order to rank the solutions, and thus perform constraint optimisation, we introduce a valuation over the constraints in terms of c-semiring operations [2]. A valuation denotes the importance of a constraint. We represent a valuation of a constraint c using $\vartheta(c)$. In the probabilistic case, for example, our valuation is taken from the interval $[0, 1]$ with total order $\leq$, and ϑ associates a probability with each constraint.

Given this framework, we can define an optimisation task given a valuation over a set of constraints. We solve this optimisation task in a distributed fashion, using an agent-based approach [25].

6.1.3 Mapping Constraints as Attributes

Given a constraint graph, or a constraint hypergraph, we can assign a constraint-based attribute to every variable at the higher-level representations. Given the constraint specification just described, we define a constraint-based attribute as follows:

Definition 9 *A constraint-based attribute for variable v consists of the tuple $\langle \rho_v, \mathcal{D}_v, \mathcal{C}_v, \vartheta_v \rangle$, where:*

- *ρ_v is the name for v;*
- *$\mathcal{D}_v$ is the domain for v;*
- *$\mathcal{C}_v$ is the set of constraints for v;*
- *ϑ_v is the valuation over the constraints for v.*

Given a model with constraint-based attributes, it is simple to prove that we can map it into a constraint-optimisation task, and solve a variety of optimisation tasks, such as scheduling, diagnosis, function optimisation, etc., given an appropriate objective function [25].

6.1.4 Resource Allocation Optimisation

The mission-model of the chilled water system focuses on resource allocation and scheduling: given a fixed set of chilled-water resources and loads, the objective is to minimise energy usage (based on an energy function ξ), i.e., to schedule turning on one or both chillers to best satisfy the (prioritised) demands from the various loads. If at time t we supply $\hat{L}_i(t)$ units of chilled water to load i, when the actual demand is $L_i(t)$, then our objective function is

$$min \left\{ \xi(t) + \sum_i \omega_i |\hat{L}_i(t) - L_i(t)| \right\}, \qquad (1)$$

where ω_i is a penalty function for supplying too much or too little chilled water to load i.

This model thus consists of: (a) variables for supplies (Ch1 and Ch2) and loads (L_1 through L_9), each with domain values {0, nominal, max}, together with (b) constraints that record the topology of the interconnections of loads and supplies, which note which loads can acquire chilled water from which chilling units.

6.2 Chilled Water System Process-Model

The process-model of Chilled Water System focuses on describing how the chilled water system can actually supply all the loads. In other words, it consists of a discrete-event control and diagnostics model that specifies the chiller and valve settings necessary to optimally supply a give set of loads.

This section describes a simplified constraint process-model of the chilled water system, described as an attributed constraint graph.

Graph Nodes: The nodes of the graph consist of variables for supply (Ch1 and Ch2), loads (L_1 through L_9), piping segments (P_1 through P_8), and valves (V_1 through V_6). For each component that can fail, we have an associated failure-mode variable, and for the controllable elements (supply and valve nodes), we have an associated control variable.

Graph Topology: we can build a hypergraph model $\mathcal{G}$ of the interconnections of the components of the shipboard system. Figure 3 depicts both a top-level view of the chilled-water system hypergraph, as well as a more detailed representation. In this figure we focus on the hyperedges between the Chilled Water sub-system and other subsystems. This constraint model captures much of the ChW structure shown in Figure 4. For example, it shows the two chiller units (Ch1 and Ch2), the piping connecting the chillers to the loads (both piping for flows from a chiller to a load and the return flows), as well as a subset of the loads.

Graph Attributes: Table 2 shows some attributes that can be assigned to the constraint graph: subsystem ρ; domain $\mathcal{D}$; constraint $\mathcal{C}$; and transport-system capacity ϑ. For each subsystem, we list a triplet of (C_1, C_2, C_3) of constraints in $\mathcal{C}$, in terms of acceptable tuples. The elements of each triplet denote the (mode, capacity) values. These correspond to: (1) for supply (ChW1, ChW2), each tuple denotes the pair {mode, supply}; (2) for load (radar, sonar), each tuple denotes the pair {mode, demand}; (3) for transport-system, each tuple denotes the pair {mode, capacity}. For example, for the ChW1 subsystem, the possible values of this chiller are {high, nominal, off}; in its operating mode of *high*, it can put out 10 units of chilled water, in its its operating mode of *nominal*, it can put out 5 units of chilled water, etc.

We use this constraint model to compute an optimal control setting, given the loads, chiller capacities and fault con-

subsystem ρ	domain $\mathcal{D}$	constraint $\mathcal{C}$
ChW1	{high, nominal, off}	{high, 10} {nominal, 5} {low, 2}
ChW2	{high, nominal, off}	{high, 10} {nominal, 5} {low, 2}
radar	{high, nominal, off}	{high, 15} {nominal, 5} {low, 2}
pipe	{nominal, leak}	{high, 7} {nominal, 5} {low, 1}

Table 2. Attributes for ChW constraint graph. The constraints denote allowable tuples for the model.

ditions at time t. We use a distributed inference approach based on the frameworks described in [16] and in [18].

6.3 Machine-Level Model of Chilled-Water System

In this ChW application, at the machine level we solve fluid transport equations for the system, in order to: (1) Determine if we can fulfill the required demands for chilled water; (2) Compute the time to deliver ChW to various Demands; and (3) Incorporate system modes in order to be able to reason about system mode configurations, e.g., battle mode specifies port/starboard partitioning, and various failure modes specify faults that can cause problems with delivery of chilled water, e.g., Chiller1-failure specifies that only Chiller2 can supply chilled water.

We have used a qualitative abstraction of dynamical systems, qualitative differential equations [14], to model our system at the machine level. We use qualitative differential equations to model only the qualitatively significant differences. We represent such significant differences using a qualitative range of values, such as {low, normal, high}.

For example, for a pipe, we can approximate the pressure drop along a length l of pipe with diameter d, flow F and friction coefficient μ, using the difference between the inlet and outlet pressures, $P_{in} - P_{out}$:

$$\Delta P = P_{in} - P_{out} = \frac{128 * \mu * l}{\pi * d^4 * F}$$

Using our qualitative approach, we can define this pressure difference ΔP as being either *nominal*, *low* or *high*, and then relate these values to the presence of leaks and/or faults in upstream pumps. For example, if we have possible modes of a pipe M_P as {*OK, blocked, leak*}, then we might have the equations:

$$[M_P = OK] \wedge [\Delta P = nominal] \Rightarrow [P_{out} = nominal]$$
$$[M_P = blocked] \wedge [\Delta P = nominal] \Rightarrow [P_{out} = low]$$
$$[M_P = leak] \wedge [\Delta P = nominal] \Rightarrow [P_{out} = low]$$

Each local component computes its status (both mode and operating parameters), and reports it to its neighbours. Using the distributed agent framework to be summarised in the following section, all of these component status values are synthesized and coordinated, in order to solve the global system optimisation task (Equation 1).

7 Distributed Agent Framework

We use an agent-based approach for multi-model inference, where each agent represents a physical process or component and coordinates its operation with other agents. Control actions are the result of coordinated decisions among all agents.

The agent structure consists of a collection of agents at each level: mission-level, process-level, and machine-level. Each agent at each level receives performance goals from a higher-level agent, and uses its internal model (whether it be a mission-, process-, or machine-level model) to achieve the desired performance goal, and/or diagnose problems.

We assign a process-level agent to each of a set of elements of the ChW system by decomposing the ChW system based on function and topology. Figure 4 shows this decomposition, where the entities in the decomposition consist of: supply elements (Ch1 and Ch2), loads (L_1 through L_9), piping segments (P_1 through P_8), and valves (V_1 through V_6). In addition, we have an agent governing the entire ChW "mission", as well as multiple machine-level agents.

When a requested performance goal cannot be achieved, agents initiate distributed problem solving actions to find alternative solutions that best meet the system's goals. Based on the physical relationships among machine agents (as specified by the system topology), the agents develop feasible plans that satisfy both local and shared constraints.

Agent-based inference is asynchronous, but is coordinated via an event-based mechanism: each agent acts independently unless an event occurs that requires inter-agent communication. For example, if an agent assigned to a valve detects that the valve has failed, it sends that failure event to its neighbours at the same level, and to its higher-level (parent) agent(s).

Another key feature of this agent architecture is that it integrates inference performed simultaneously on several multi-level models. For example, in the Chilled Water system, agents deal with constraint-based models for discrete-event control and steady-state diagnostics, as well as dynamical systems models for transient analysis/FDI.

This agent-based approach has proven itself successful at deriving an energy-minimal control regime for the ChW system (Equation 1), under a variety of operating conditions. In particular, it quickly introduces appropriate control reconfigurations given faults injected into the system, such as simulated pipe leaks and chiller failures.

Cooperative software agents permit the creation of highly flexible control systems based on both autonomy and cooperation. The agent framework is well suited for solving resource allocation problems on distributed hardware [16]. Autonomous capabilities are directly designed into each agent so that it can continue to operate in the event of failures in the other agents.

8 Conclusions

We have described a framework for modeling complex systems, based on an Attributed Programmed Graph Grammar (APGG) meta-model, and on performing inference on a set of multi-level models generated from the meta-model via model transformations. In particular, we examined higher-level constraint-based models as applied to a shipboard application, and how a distributed agent-based framework can coordinate inference performed asynchronously on these multi-level models.

References

[1] F. J. Barros. Dynamic Structure, Multiparadigm Modeling, and Simulation. *ACM Trans. Model. Comput. Simul.*, 13(3):259–275, 2003.

[2] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint logic programming: syntax and semantics. *ACM TOPLAS*, 23(1), 2002.

[3] H. Bunke. Programmed graph grammars. In *Proceedings of the International Workshop on Graph-Grammars and Their Application to Computer Science and Biology*, pages 155–166, London, UK, 1979. Springer-Verlag.

[4] K. Cremer. Graph-based reverse engineering and reengineering tools. In *AGTIVE '99: Proceedings of the International Workshop on Applications of Graph Transformations with Industrial Relevance*, pages 95–109, London, UK, 2000. Springer-Verlag.

[5] L. da Fontoura Costa. The hierarchical backbone of complex networks. *Physical Review Letters*, 93(9):098702, 2004.

[6] J. de Lara, H. Vangheluwe, and M. Alfonseca. Meta-modelling and graph grammars for multi-paradigm modelling in AToM^3. *Software and System Modeling*, 3(3):194–209, 2004.

[7] R. Dechter. Constraint Networks. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1. Addison-Wesley Publishing Company, 1992.

[8] F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *J. Comput. Syst. Sci.*, 64(2):249–283, 2002.

[9] C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: language and environment. pages 551–603, 1999.

[10] I. Finocchi and R. Petreschi. Structure-preserving hierarchical decompositions. *Theor. Comp. Sys.*, 38(6):687–700, 2005.

[11] M. T. Gastner and M. E. Newman. The spatial structure of networks. *The European Physical Journal B - Condensed Matter*, 49(2):247 – 252, 2006.

[12] V. Kalapala, V. Sanwalani, A. Clauset, and C. Moore. Scale invariance in road networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 73(2):026130, 2006.

[13] H.-J. Kreowski and S. Kuske. Graph transformation units with interleaving semantics. *Formal Aspects of Computing*, 11(6):690–723, 1999.

[14] B. Kuipers. *Qualitative Reasoning*. MIT Press Cambridge, Mass, 1994.

[15] A. Marburger and B. Westfechtel. Tools for understanding the behavior of telecommunication systems. In *ICSE '03: Proc. of the 25th Intl. Conf. on Software Engineering*, pages 430–441, Washington, DC, USA, 2003.

[16] P. J. Modi, W.-M. Shen, M. Tambe, and M. Yokoo. An asynchronous complete method for distributed constraint optimization. In *AAMAS '03: Proc. Itnl. Conf. on Autonomous Agents and Multiagent Systems*, pages 161–168, New York, NY, USA, 2003.

[17] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45(2):167– 256, 2003.

[18] G. Provan. A Novel Framework for Integrating Discrete-Event Control and Diagnosis. In *Proceedings of IJCAI*, Cancun, Mexico, August 2003.

[19] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*. World Scientific, 1997.

[20] A. Schurr. Programmed Graph Replacement Systems. pages 479– 546. World Scientific, 1996.

[21] J. Sztipanovits and G. Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.

[22] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[23] A. Uhrmacher, D. Degenring, and B. P. Zeigler. Discrete event multi-level models for systems biology. *T. Comp. Sys. Biology*, 1:66–89, 2005.

[24] H. Vangheluwe and J. de Lara. Foundations of Multi-paradigm Modeling and Simulation: Computer Automated Multi-Paradigm Modelling: Meta-Modelling And Graph Transformation. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 595–603. Winter Simulation Conference, 2003.

[25] M. Yokoo. *Distributed Constraint Satisfaction: Foundations of Cooperation in Multi-Agent Systems*. Springer-Verlag, Berlin, 2001.

[26] B. P. Zeigler, H. S. Song, T. G. Kim, and H. Praehofer. DEVS framework for modeling, simulation, analysis, and design of hybrid systems. *Lecture Notes in Computer Science*, 999:529–549, 1995.