Models, Metamodels, and Model Transformation for Cyber-Physical Systems

Nathan Jarus*, Sahra Sedigh Sarvestani†, and Ali R. Hurson*
*Department of Computer Science
†Department of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO 65409, USA
Email: {nmjxv3,sedighs,hurson}@mst.edu

***Abstract*—One approach to increasing the sustainability of critical systems is to fortify them with cyber infrastructure that monitors the system, enables early diagnosis of faults, and provides decision support that facilitates greater efficacy. Modeling and analysis of the resulting cyber-physical systems is a significant challenge, as the physical and cyber infrastructures can be very different in time scales, complexity, and architecture. Model composition is a potential solution. Metamodeling seeks to facilitate model composition by providing methods for composing models of different types, including performance and dependability models, as well as methods for transforming one type of model to another. This paper describes the application of metamodeling to cyber-physical systems. Our proposed approach to model transformation is based on abstract interpretation, a program analysis technique. Models exist for disparate attributes of cyber-physical systems, but these models are typically domain-specific. We seek to map models from one physical domain to another, or to extract a model of one system attribute from a model of another attribute. This ability would considerably increase the utility of the existing body of knowledge on modeling of cyber-physicals systems.**

I. INTRODUCTION

For infrastructure to be sustainable, it must be both efficient and dependable. Supplementing physical infrastructure with cyber control and communication is a means of achieving both objectives. Identifying suitable cyber elements and determining where they should be placed in the system requires understanding of the effects of their addition on functional and nonfunctional attributes of the system. Models that provide a unified representation of the physical and cyber operation are critical to this understanding, and to justifiable reliance on the resulting *cyber-physical system (CPS)*.

The scale, complexity, and heterogeneity typical of critical infrastructure complicates modeling of these (CPSs). No single model can capture all aspects of a complex system. If any changes are made to a system as part of the design process, the modeling process must start over in order to identify all assumptions that have changed. The field of *metamodeling* has been introduced to increase the efficacy of modeling by facilitating automatic or semi-automatic model generation and composition [23].

Many existing metamodeling tools take a top-down approach, where the system-level model is composed from component-level submodels, which may be of different types. This *model composition* offers detailed and accurate representation, but is complex and requires a) the construction of a large number of models and b) the ability to unify these models in a semantically correct fashion.

Model transformation, in contrast, aims to transforming one system model into a different type of model of the same system, providing a means for model checking to ensure each model holds the same assumptions about the system. The most common model transformation approach is to view models as labeled graphs and to define *graph rewriting* rules that transform one model's graph into another. A significant constraint on this approach is that it only applies to models that can be viewed as graphs. An alternative is the use of *object-oriented class inheritance* to transform models, whose application is constrained to models that can be expressed in terms of an object hierarchy. Creating a technique that can facilitate diverse transformations between diverse model types is an open challenge.

While model transformation may reduce model fidelity by simplifying the task of modeling, the transformation framework itself is prone to error. Care must be taken to ensure that any transformation approach is both correct and specific. *Correctness* requires that the result of transforming one model to another be at most a set of several models containing the desired system model [20]. *Specificity* requires that the set of models produced from a transformation be as small as possible, and the user be provided with a means of selecting the most appropriate model from the generated set.

In this paper, we present a model transformation approach based on concepts from programming language theory. There exist many syntactically different but semantically equivalent ways to write any program. For instance, we could write a program using `while` loops instead of `for` loops, or we could write the program in an entirely different language. Programming language theory studies how to transform one syntactic representation of a program into a syntactically different but semantically equivalent program. A common example of this type of transformation is the process of compiling source code into machine language. We illustrate the application of this approach to system modeling and model transformation.

978-1-5090-5117-5/16/$31.00

TABLE I
RELATED MODEL COMPOSITION AND TRANSFORMATION TOOLS

Project	Target Field	Applicable To	Transformation Technique	Examples
Ptolemy	Cyber-physical systems	Anything with a computation model	Hierarchical	[1], [2], [3], [4]
Möbius	Complex network systems	Performance and dependability	Hierarchical	[5], [6], [7]
AToM^3	General modeling	Anything that can be metamodeled	Graph rewriting	[8], [9]
CONCERTO	Model-driven engineering	Functional models	Graph rewriting	[10], [11], [12], [13], [14]
SIMTHESys	General modeling	Anything that can be metamodeled	Inheritance-based	[15], [16], [17], [18]
Rosetta	General modeling	Discrete event systems	Coalgebraic functors	[19], [20], [21], [22]

Every system model can be deterministically described; therefore, we can view syntactic descriptions of models as a language for semantically describing systems. Applying concepts from programming language theory will allow us to transform these models in a process analogous to compiling a program or performing static program analysis. Many techniques common to program transformation can be proven to be correct. Consequently, our application of these techniques will yield models transformations that are provably correct.

In the remainder of this paper, we briefly introduce model composition and transformation frameworks in Section II. The mathematical theory inspiring our work is presented in Section III. Our proposed approach to model transformation is described in Section IV and illustrated through an example in Section V. Finally, we discuss our conclusions from and future extensions to the work in Section VI.

II. RELATED WORK

Most model transformation techniques are developed as part of a metamodeling tool. In this section we present several metamodeling projects and their approach to model composition and transformation. Table I provides a comparison and summary of these approaches.

The Ptolemy modeling software [1] performs hierarchical modeling and model composition [2], [3]. As such, Ptolemy makes it easy to build and link small models. Hierarchical models can consist of heterogeneous submodels, allowing different parts of the system to be expressed using different types of models. Ptolemy allows for heterogeneous computation: it provides choices for both the modeling language and the solution or simulation technique used to evaluate the model [4]. However, Ptolemy does not offer methods for transforming one system-level model to another.

Möbius [5] is another modeling tool that supports hierarchical modeling. It supports several modeling formalisms, including block diagrams and Petri nets, and can be interfaced to external modeling tools for additional formalisms [6], [7]. While this feature offers considerable flexibility in modeling, Möbius is constructed around a modeling workflow that builds and evaluates hierarchical models and has little support for model transformation.

AToM^3 [8] is capable of both model transformation and model composition. It uses metamodels to describe specific modeling languages, then defines transformations between metamodels to transform models [9]. Models are graph-based and transformations take the form of graph rewriting rules. However, there is no hierarchy of models, so introducing a new model requires writing transformation rules from the new model to each model that AToM^3 implements.

CHESS [10] provides a modeling language for describing systems and includes several model transformation methods specific to creating dependability models. CHESS is based on the Unified Modeling Language (UML); transformations are based on graph rewriting rules. CONCERTO [11] extends CHESS by introducing modeling techniques for nonfunctional system attributes such as dependability [12]. However, CONCERTO is focused on multicore processing systems [13], [14] and lacks the features necessary for modeling of CPSs.

OsMoSys [15] and SIMTHESys [17] are multimodeling systems motivated by model-driven engineering. Their approach to model transformation is based on techniques from software engineering, particularly object-oriented programming and its class inheritance paradigm. OsMoSys features compositional models and interfaces with external tools to solve them [16]. SIMTHESys adds the ability to generate formalism modeling and solution tools based on a user-specified description of a formalism. The solution tools are used to interpret and solve a model specified in the formalism. OsMoSys and SIMTHESys are capable of modeling both functional and non-functional aspects of a system [18]. They perform model transformation by viewing models as classes - in the object-oriented programming sense - and using inheritance principles to convert between models.

Rosetta [19] is focused on functional multiformalism modeling [20]. It takes an algebraic approach to relating models: each formalism is described as a *coalgebra* - a mathematical system useful for describing arbitrary transitions among arbitrary states [21], [22]. The coalgebras corresponding to each formalism are placed in a lattice, which provides a straightforward structure for determining how to transform one model into another (see Section III). Model transformations can be used to relate different models of the same system; for example, it is possible to combine a functional system model with a model of that system's power consumption. However, Rosetta lacks many features required for CPS modeling, especially support for hybrid discrete-continuous formalisms. Furthermore, while the authors propose a method of model transformation that can be proven correct, they do not address specificity; thus, it may be difficult to get meaningful results from certain transformations.

The approach we propose in this paper focuses on model-to-model transformation and is not limited to composition of submodels into a system model. Our approach is general - it allows a variety of transformations among a variety of models. Its application is not constrained to models that can be

represented as a graph or transformations that can be described through an inheritance paradigm. Our focus is on computing for sustainability, and as such, our approach operates on both performance and dependability models in order to fully capture the sustainable behavior of a system. A detailed mathematical foundation enables us to prove the correctness of our approach. Finally, we ensure specificity of our transformation results by allowing users to customize them.

III. Mathematical Foundations

In this section, we present the mathematical theory that underpins our model transformation approach. We begin with a discussion on ordered sets and lattices, which form the basis for our definition of 'correctness.' Next, we elaborate upon specific mappings between lattices, leading up to Galois connections, which we use as model transformation operations. Finally, we reformulate abstract interpretation, a program transformation and analysis technique, in terms of system model transformation.

A. Lattices

A *lattice* is a structure defined on and imparting a specific order upon those objects. Lattices can guarantee the existence of certain properties for objects and functions. The order properties of lattices are commonly used to reduce the search space of complex algorithms. The discussion here is intended to provide a brief overview sufficient for description of our model transformation approach; readers are referred to [24] for greater detail.

Integral to the concept of a lattice is the concept of *ordering* on a set:

Definition III.1. A ***poset***, short for ***partially ordered set***, $(L, \sqsubseteq)$ is a set L and an order relation $\sqsubseteq$ $(read 'less than or equal')$: $L \times L \to \{\mathbf{true}, \mathbf{false}\}$ that is

1) *Reflexive*: $l \sqsubseteq l, \forall l \in L$
2) *Transitive*: If $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_3$, then $l_1 \sqsubseteq l_3, \forall l_1, l_2, l_3 \in L$
3) *Anti-symmetric*: If $l_1 \sqsubseteq l_2$ and $l_2 \sqsubseteq l_1$, then $l_1 = l_2, \forall l_1, l_2 \in L$

Partial ordering does not require that $\forall l_1, l_2 \in L$, $l_1 \sqsubseteq l_2$ or $l_2 \sqsubseteq l_1$. Such a requirement would yield a *total* rather than partial ordering.

For example, the order relation $\sqsubseteq$ can be defined on a Cartesian coordinate system ($\mathbb{R}^2$) as:

$$(x_1, y_1) \sqsubseteq (x_2, y_2) \iff x_1 \leq x_2 \text{ and } y_1 \leq y_2$$

With this definition, $(0,0) \sqsubseteq (1,2)$ and $(0,0) \sqsubseteq (2,1)$, but $(1,2)$ and $(2,1)$ are not comparable using $\sqsubseteq$, $1 \leq 2$ but $2 \not\leq 1$.

Definition III.2. A set $Y \subseteq L$ has $l \in L$ as an ***upper bound*** if $y \sqsubseteq l, \forall y \in Y$. l is a ***least upper bound*** if for any upper bound l', $l \sqsubseteq l'$. The least upper bound is denoted as $\bigsqcup Y$, i.e., the *meet* of Y.
$\bigsqcup\{l_1, l_2\}$ can also be written as $l_1 \sqcup l_2$.

Definition III.3. l is a ***lower bound*** of Y if $l \sqsubseteq y, \forall y \in Y$. The ***greatest lower bound*** is defined analogously to the least upper bound. It is denoted as $\bigsqcap Y$, i.e., the *join* of Y.

Continuing our example in $\mathbb{R}^2$, $(0,0)$ can be identified as a lower bound of $\{(1,2),(2,1)\}$. In addition, $\bigsqcup\{(1,2),(2,1)\} = (2,2)$ and $\bigsqcup\{(0,0),(0,1),(1,0)\} = (1,1)$.

Functions relating two posets P_1 and P_2 can have the following properties:

Definition III.4. A function $f : P_1 \to P_2$ between posets $(P_1, \sqsubseteq_1)$ and $(P_2, \sqsubseteq_2)$ is ***monotone*** (or ***order-preserving***) if

$$p_1 \sqsubseteq_1 p_2 \Rightarrow f(p_1) \sqsubseteq_2 f(p_2), \forall p_1, p_2 \in P_1$$

As an example, let $f : \mathbb{R}^2 \to \mathbb{R}^2$ be defined as $f(x,y) = (x+1, y)$. f is monotone, as:
$(x_1, y_1) \sqsubseteq (x_2, y_2) \iff x_1 \leq x_2 \iff x_1 + 1 \leq x_2 + 1 \iff f(x_1, y_1) \sqsubseteq f(x_2, y_2)$.

On the other hand, $g(x,y) = (x, y^2)$ is not monotone, as:
$(0,-3) \sqsubseteq (0,-2)$ but $g(0,-3) = (0,9) \not\sqsubseteq g(0,-2) = (0,4)$.

Lattices are posets with specific properties. The definitions below articulate these properties.

Definition III.5. A ***complete lattice***, L, is a partially ordered set where the least upper bound and greatest lower bound, $\bigsqcup Y$ and $\bigsqcap Y$, respectively, can be defined for all $Y \subseteq L$.

A consequence of III.5 is that every complete lattice has a ***least element***, $\bot = \bigsqcap L$, and a ***greatest element***, $\top = \bigsqcup L$.

As an example, $(\mathbb{R}^2, \sqsubseteq)$ forms a complete lattice with $\bot = (-\infty, -\infty)$ and $\top = (\infty, \infty)$.

A complete lattice of note is the set of all subsets of a set S, denoted as the ***powerset***, $\mathcal{P}(S)$. $(\mathcal{P}(S), \subseteq)$ forms a complete lattice, with $\bigsqcup = \bigcup$ and $\bigsqcap = \bigcap$. $\top = S$ and $\bot = \emptyset$, as every element of $\mathcal{P}(S)$ is a subset of S and $\emptyset$ is a subset of every set.

For example, if $S = \{1,2,3\}$,

$$\mathcal{P}(S) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

The resulting lattice can be drawn as in Fig. 1.

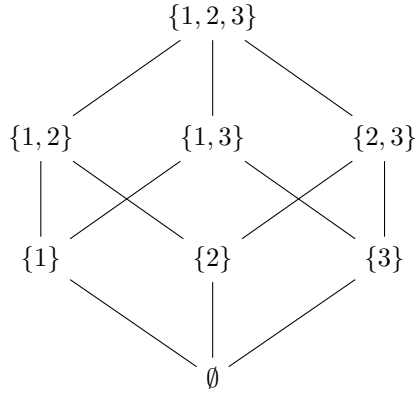

Fig. 1. The lattice for $(\mathcal{P}(S), \subseteq)$.

It is worth noting that any set $M \subseteq \mathcal{P}(S)$ also forms a poset $(M, \subseteq)$. Posets that are ordered by inclusion ($\subseteq$) are denoted as ***inclusion posets***.

Theorem III.1. *Every poset is order-isomorphic to an inclusion poset [25].*

B. Galois Connections

A *Galois connection* between two posets is a somewhat weaker relationship than an order-preserving isomorphism [25]. They are commonly used to abstract information from one poset to another poset that has a better-understood structure. In our approach to model transformation, we abstract information from a model of one aspect of a system, then concretize the abstracted information into models of other aspects of that system.

Definition III.6. A ***Galois connection*** (P, α, γ, Q) between two posets $(P, \sqsubseteq_P)$ and $(Q, \sqsubseteq_Q)$, is a pair of monotone functions $\alpha : P \to Q$ and $\gamma : Q \to P$ such that

$$p \sqsubseteq_P (\gamma \circ \alpha)(p), \forall p \in P$$

$$(\alpha \circ \gamma)(q) \sqsubseteq_Q q, \forall q \in Q$$

α is denoted as the *abstraction operator* and γ as the *concretization operator*.

An alternative representation of a Galois connection can be constructed using $\lambda x.x$ to denote a function that takes a parameter x and returns x.

$$\lambda p.p \sqsubseteq_P \gamma \circ \alpha$$

$$\alpha \circ \gamma \sqsubseteq_Q \lambda q.q$$

Theorem III.2. *α uniquely determines γ by*

$$\gamma(q) = \bigsqcup \{p : \alpha(p) \sqsubseteq_Q q\}$$

and likewise, γ uniquely determines α by

$$\alpha(p) = \bigsqcap \{q : p \sqsubseteq_P \gamma(q)\}$$

Proof.

$$\begin{aligned} \gamma(q) &= \bigsqcup \{p : p \sqsubseteq_P \gamma(q)\} && \text{definition of } \gamma \\ &= \bigsqcup \{p : \alpha(p) \sqsubseteq_Q (\alpha \circ \gamma)(q)\} && \alpha \text{ is monotone} \\ &= \bigsqcup \{p : \alpha(p) \sqsubseteq_Q q\} && (\alpha \circ \gamma)(q) \sqsubseteq_Q q \end{aligned}$$

If both γ_1 and γ_2 form Galois connections with α, then both γ_1 and γ_2 are defined to be $\bigsqcup \{p : \alpha(p) \sqsubseteq_Q q\}$ and thus $\gamma_1 = \gamma_2$. We can similarly prove that α is uniquely determined. □

This theorem does not state that there exists a unique Galois connection between two lattices; there may be two connections, $(P, \alpha_1, \gamma_1, Q)$ and $(P, \alpha_2, \gamma_2, Q)$, that vary in the values of both the abstraction and concretizer operator. However, it is not possible to have two distinct connections (P, α, γ_1, Q) and (P, α, γ_2, Q) with a common value for the abstraction operator, but differ in the value of the concretization operator.

Finally, we can repeatedly abstract and then concretize without incurring a loss of detail:

Theorem III.3. *$\alpha \circ \gamma \circ \alpha = \alpha$ and $\gamma \circ \alpha \circ \gamma = \alpha$.*

Proof. Using the alternate notation for Definition III.6:

$$\begin{aligned} & \lambda p.p \sqsubseteq_P \gamma \circ \alpha && \text{Defn. III.6} \\ \Rightarrow \quad & \alpha \sqsubseteq_Q \alpha \circ \gamma \circ \alpha \\ & \alpha \circ \gamma \sqsubseteq_Q \lambda q.q && \text{Defn. III.6} \\ \Rightarrow \quad & \alpha \circ \gamma \circ \alpha \sqsubseteq_Q \alpha \\ \Rightarrow \quad & \alpha \circ \gamma \circ \alpha = \alpha && \text{Anti-symmetry (Defn. III.1)} \end{aligned}$$

We can similarly prove that $\gamma \circ \alpha \circ \gamma = \gamma$. □

Galois connections are commonly used in program analysis, because they can encode the concept of correctness. In the context of transformation, correctness implies that the result of transforming an object from one lattice to another and back to the original lattice will be at worst more general than the original object. This attribute is valuable in identifying approximations to complex computations. Let (P, α, γ, Q) be a Galois connection and $f_P : P \to P$ a computationally complex operation. Assume there exists an analogous operation, $f_Q : Q \to Q$, that is easier to compute. Assume that $f_P(p_1) = p_2$. The correctness property of Galois connections ensures that $p_2 \sqsubseteq_P (\gamma \circ f_Q \circ \alpha)(p_1)$. f_P can hence be approximated by abstracting the operation to Q, then concretizing the result of f_Q.

C. Abstract Interpretation

Abstract interpretation was originally developed to unify disparate program analysis techniques [26] . In this section, we discuss the application of abstract interpretation to system and model transformation.

In general, a complete system $\mathbf{S}$ is impossible to fully describe. Models are therefore created to describe and represent different aspects of complete systems. If a model, $m \in M$, describes a system, $\mathbf{S}$, this relationship is denoted as $\mathbf{S} \vdash m$ (and read 'as $\mathbf{S}$ entails m'). If it is possible to transform model m_1 into model m_2, we write $\mathbf{S} \vdash m_1 \rightsquigarrow m_2$. $\rightsquigarrow$ may or may not be a function in the mathematical sense. Furthermore, it is difficult to reason about $\rightsquigarrow$ in a general fashion, as it will vary from model to model and system to system.

To facilitate analysis, we relate models to the system properties described by those models. We select properties that can be deterministically transformed under the given system. If a property, $p \in P$, holds for a system, we write $\mathbf{S} \vdash p$. If a set of properties, $p_1 \in P$, can be transformed into a different set of properties, $p_2 \in P$, we write $\mathbf{S} \vdash p_1 \rhd p_2$. Since $\rhd$ is deterministic, i.e., a given p_1 can only be transformed into one specific p_2, we can define a *transformation function* $f_{\mathbf{S}}(p_1) = p_2$.

The final element of our abstraction is a *correctness relation*, $R : M \times P \to \{\mathbf{true}, \mathbf{false}\}$, where M is the set of models and P is the set of sets of properties. If a model m is described by property p, then $m\,R\,p$. In order for $\rhd$ to be well-defined, R has to be preserved under transformation: if $\mathbf{S} \vdash m_1 \rightsquigarrow m_2$, $\mathbf{S} \vdash p_1 \rhd p_2$, and $m_1\,R\,p_1$, then $m_2\,R\,p_2$ as well.

Given $\rhd$ and R, we can reason about model transformation as follows. Assume a system $\mathbf{S}$ and a model of that system, $\mathbf{S} \vdash m_1$, and we wish to compute m_2 such that $\mathbf{S} \vdash m_1 \rightsquigarrow m_2$. Given some set of properties p_1 such that $m_1\,R\,p_1$, we can

deterministically compute $\mathbf{S} \vdash p_1 \rhd p_2$. Then, we find a model m_2 such that $m_2 \, R \, p_2$. This approach then guarantees that $\mathbf{S} \vdash m_1 \rightsquigarrow m_2$, allowing us to reason about $\rightsquigarrow$, albeit in an indirect fashion.

This relationship between model transformation, property transformation, and correctness is illustrated in Fig. 2.

$$\begin{array}{ccccc} \mathbf{S} & \vdash & m_1 & \rightsquigarrow & m_2 \\ & & \| & & \| \\ & & R & \Rightarrow & R \\ & & \| & & \| \\ \mathbf{S} & \vdash & p_1 & \rhd & p_2 \end{array}$$

Fig. 2. Relationship between model transformation and a correctness relation.

We now demonstrate now to leverage the correctness offered by a Galois connection. Assume that $(P, \sqsubseteq)$ is a complete lattice. If necessary, we can 'lift' an underlying set of properties, D, into a lattice, P, by taking $P = \mathcal{P}(D)$.) The correctness relation R can be constrained based on two properties of P:

(i) If $m \, R \, p_1$ and $p_1 \sqsubseteq p_2$, then $m \, R \, p_2$.
(ii) If $m \, R \, p, \forall p \in P' \subseteq P$, then $m \, R \, \bigsqcap P'$.

Property (i) implies that the more specific a set of properties, the more precisely (and thus effectively) it describes a model. Property (ii) implies that given several sets of properties describing a model, there exists a unique set of properties that provides the best description.

We now define a *representation function*, $\beta : M \to P$, that maps a models to the properties that best describe them. If $\beta(m_1) \sqsubseteq p_1$, $\mathbf{S} \vdash m_1 \rightsquigarrow m_2$, and $\mathbf{S} \vdash p_1 \rhd p_2$, then $\beta(m_2) \sqsubseteq p_2$. β can be defined in terms of R, or vice-versa. This relationship is described by the diagram in Fig. 3.

$$\begin{array}{ccccc} \mathbf{S} & \vdash & m_1 & \rightsquigarrow & m_2 \\ & & \beta \downarrow & \Rightarrow & \beta \downarrow \\ & & \mathbin{\rotatebox{90}{$\vDash$}}\sqcap & & \mathbin{\rotatebox{90}{$\vDash$}}\sqcap \\ \mathbf{S} & \vdash & p_1 & \rhd & p_2 \end{array}$$

Fig. 3. Relationship between model transformation and a representation function.

The definition of β may be very difficult for certain property formulations. The combination of (i) and (ii) allows us to approximate β by finding properties in the property set lattice that describe the desired destination model, and then applying fixpoint theory to narrow that property down to the best (smallest) property we can find.

To show the correctness of β, we can define a Galois connection between P and $\mathcal{P}(M)$:

$$\gamma(p) = \{m \in M : \beta(m) \sqsubseteq p\} \tag{1}$$

$$\alpha(M') = \bigsqcup \{\beta(m) : m \in M'\} \tag{2}$$

Concretizing a set of properties yields the models described by those properties. Abstracting a set of models yields the smallest set of properties describing all models in the set.

If a set of properties, D, has been 'lifted' into a powerset lattice, the construction is slightly different. Instead of directly defining β, we define a function, $\eta : M \to D$, that maps models to properties. Let $\beta : M \to \mathcal{P}(D)$ be defined by $\beta(m) = \{\eta(m)\}$, yielding a function that relates models and elements of P.

$$\gamma(P') = \{m \in M : \beta(m) \subseteq P'\} = \{m \in M : \eta(m) \in P'\} \tag{3}$$

$$\alpha(M') = \bigsqcup \{\beta(m) : m \in M'\} = \{\eta(m) : m \in M'\} \tag{4}$$

All of these functions are related, as depicted in Fig. 4. Note that $\mathbf{lift} : X \to \mathcal{P}(X)$ is defined by $\mathbf{lift}(x) = \{x\}$.

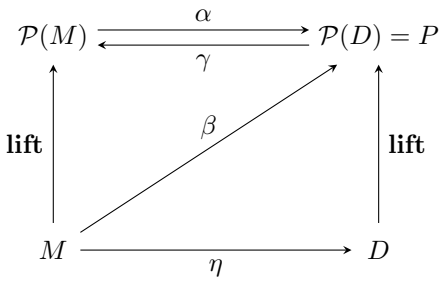

Fig. 4. Galois connection between values and powerset of properties.

IV. Proposed Approach to Model Transformation

Our goal is to be able to perform the following operations:

1) Convert system properties to a model describing those properties.
2) Convert a model to a set of properties that it describes.
3) Incorporate additional information into a set of properties to allow the derivation of additional models from the set.

We must be able to guarantee the correctness of these operations, i.e., that the result they yield is at most a more general result.

Central to this research is the idea of abstracting system properties from models. This allows the selection of ideal representations for both types of objects. We construct one lattice of system properties and a lattice for each type of model. From this, we define Galois connections between the system properties lattice and the lattice of each model. Finally, we demonstrate how to specialize generated models that are too general, by allowing the user to introduce additional information. This last situation arises in cases where the known system properties are not sufficient to define all parameters of the generated model.

We begin by considering the set $\mathbf{properties}$ of sets of system properties and the set $\mathbf{models}$ of objects that are models of a system. For now, we will not identify the elements of these sets; Section V provides one approach to defining these elements.

To view model transformation through the lens of abstract interpretation, sets of properties and models should be considered. A model may describe more than one system, as few models can determine all of a system's parameters, and a set of properties may not describe a sufficient number of parameters

to determine a unique model. To account for these possibilities, we 'lift' our modeling approach from **properties** and **models** to the powersets $\mathcal{P}(\mathbf{properties})$ and $\mathcal{P}(\mathbf{models})$. We can then describe the interactions between these sets via the diagram in Fig. 5.

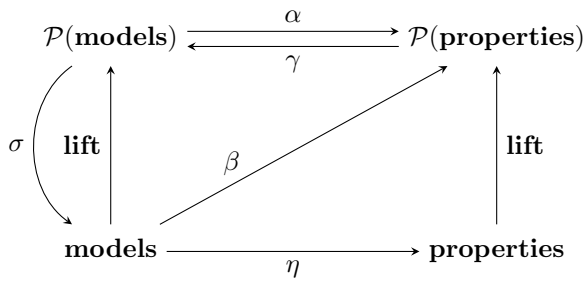

Fig. 5. System and model interaction diagram.

β converts a model to the properties it describes. σ selects one model from a set of models, effectively providing information about assumptions that were not available in the set of properties from which the set of models was created.

If we have $p \in \mathcal{P}(\mathbf{properties})$ and $m = \sigma(\gamma(p))$, the additional information provided by σ can be incorporated into $p' \in \mathcal{P}(\mathbf{properties})$ by letting $p' = p \cup \beta(m)$.

At this point, we have defined all of the operations we require for model transformation, but have not discussed correctness. Since every powerset is a complete lattice, we can define a Galois connection $(\mathcal{P}(\mathbf{properties}), \alpha, \gamma, \mathcal{P}(\mathbf{models}))$ between $\mathcal{P}(\mathbf{properties})$ and $\mathcal{P}(\mathbf{models})$. If α and γ form a Galois connection, it is guaranteed that they will correctly transform elements of the sets on which they operate. Since the operation of lifting is trivially correct, this connection effectively provides guarantees over the entire diagram - provided that we define operations in terms of α and γ.

Transforming from properties to a model (γ), yields a model that describes those properties, but may also describe other sets of properties equally well. For instance, since $\alpha(\gamma(p)) \sqsubseteq p$ for any $p \in \mathcal{P}(\mathbf{properties})$, the model given by $\gamma(p)$ may describe any set of properties that is a subset of p. Transforming from a model to a set of properties (α), yields the set of properties described by that model. In either case, it is impossible for the result to be incorrect; effectively, we may trade specificity for correctness. σ then becomes a means to restore specificity.

This approach shifts the objective of model transformation from finding transformations directly between types of models to a process comprised of abstraction and concretization. Suppose we have two types of models, $\mathbf{model_1}$ and $\mathbf{model_2}$, and that we wish to transform $m_1 \in \mathbf{model_1}$ to an $m_2 \in \mathbf{model_2}$. We will have the following two Galois connections:

$$(\mathcal{P}(\mathbf{model_1}), \alpha_1, \gamma_1, \mathcal{P}(\mathbf{properties}))$$

$$(\mathcal{P}(\mathbf{model_2}), \alpha_2, \gamma_2, \mathcal{P}(\mathbf{properties})),$$

and the associated representation functions:

$$\beta_1 : \mathbf{model_1} \to \mathcal{P}(\mathbf{properties})$$

$$\beta_2 : \mathbf{model_2} \to \mathcal{P}(\mathbf{properties})$$

.

We can abstract information from m_1 to **properties** by letting $p_1 = \beta_1(m_1)$. The correctness guarantee of the Galois connection implies that $m_1 \in \gamma_1(p_1)$. Since we have a second Galois connection to $\mathcal{P}(\mathbf{model_2})$, we can let $M_2 = \gamma_2(p_1)$. At this point, we can define $\sigma_2 : \mathcal{P}(\mathbf{model_2}) \to \mathbf{model_2}$ to introduce any additional information about the system that is necessary to uniquely identify m_2 in M_2. Therefore, we are able to correctly transform models and to explicitly identify system modeling assumptions.

We can capture the additional modeling assumptions provided by $\sigma_2(M_2) = m_2$ in $\mathcal{P}(\mathbf{properties})$ by abstracting properties from m_2: $p_2 = \beta_2(m_2)$. This information can be unified with the information from m_1 by taking $p = p_1 \sqcup p_2$. Thus, the effort expended on identifying modeling assumptions can be leveraged when performing further model transformations.

V. Example

We illustrate our proposed approach to model transformation with an example inspired by the case study in [27], where a Markovian reliability model is developed for a smart grid. For clarity and brevity, we consider a power grid with one generator, one load, and two power lines connected in parallel, as shown in Fig. 6. This figure represents the *topological model* of the system.

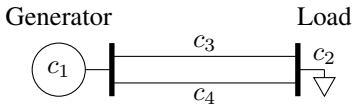

Fig. 6. Sample power grid topology.

The MIS model is a reliability model; it describes system lifetime–the probability the system has not failed before a given time - and determines system-level reliability based on the reliability of individual components. We assume for each line a reliability of p_L (and thus unreliability of $q_L = 1 - p_L$); the generator and load are assumed to not fail and have been omitted from the model. We consider the system to be functional as long as it is capable of transmitting power, i.e., the system is operational if at least one line is up.

The MIS model for the reliability of this system, under our assumptions, is given by four basic equations. The possible states are enumerated in Table II; each state represents one combination of failed and functional components and can be represented by the set of functional components. As an example, $S_1 \simeq \{c_3, c_4\}$, $S_2 \simeq \{c_3\}$, etc. The system is typically assumed to begin operation from an initial state where all components are functional - for this example, S_1. The resulting probability distribution of initial system states is given in Equation 5; this is generally of the form $[1, 0, \ldots, 0]$ in MIS reliability modeling. Equation 6 identifies the states that are considered functional. The matrices in Equations 7 and 8 describe how the failure of c_3 and c_4, respectively, causes the system state to change, i.e., these matrices represent

the state transition probabilities. Equation 9 composes all of these elements into an expression for system-level reliability as a function of transmission line reliability. For very simple systems, the expression can be written by inspection.

TABLE II
STATE DEFINITION MATRIX FOR m.

	Components	
States	c_3	c_4
S_1	1	1
S_2	1	0
S_3	0	1
S_4	0	0

$$\Pi_0 = [1, 0, 0, 0] \tag{5}$$

$$u = [1, 1, 1, 0] \tag{6}$$

$$P_{c_3} = \begin{bmatrix} p_L & 0 & q_L & 0 \\ 0 & p_L & 0 & q_L \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$P_{c_4} = \begin{bmatrix} p_L & q_L & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & p_L & q_L \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$R = \Pi_0^T * P_{c_3} * P_{c_4} * u = p_L^2 + 2p_L q_L \tag{9}$$

Thus far we have two models: a topological model and a reliability model. Let the topological model be denoted as $t \in \mathbf{topology}$ and assume that it can be represented as a graph with labeled edges and nodes. The function $\beta_{\mathbf{topology}} : \mathbf{topology} \rightarrow \mathcal{P}(\mathbf{properties})$ abstracts a representation of system properties described by this topology. Let $\beta_{\mathbf{topology}}(t) = T$. Members of $\mathbf{properties}$ are labeled tuples; T contains elements of the following types:

$$\mathsf{components} \subseteq \mathbf{component} \tag{10}$$

$$\mathsf{attributes} \subseteq \mathbf{component} \times \{\mathtt{generator}, \mathtt{load}, \mathtt{line}\} \tag{11}$$

$$\mathsf{links} \subseteq \mathbf{component} \times \mathbf{component} \times \mathbf{component} \tag{12}$$

$$\mathsf{neighbors} \subseteq \mathbf{component} \times \mathcal{P}(\mathbf{component}) \tag{13}$$

This formalism enables identification of the components of a system, the attributes of each component, the connections (edges) among components, and the edges incident on each node. The set $\mathbf{component}$ is an '*index set*'; the elements themselves are uninteresting, but they provide a means to attain useful information from other functions, such as $\mathsf{attributes}$.

For the example of Figure 6:

$$\begin{aligned} \mathsf{components}(T) &= \{c_1, c_2, c_3, c_4\} \\ \mathsf{attributes}(T) &= \{(c_1, \mathtt{generator}), (c_2, \mathtt{load}), \\ &\qquad (c_3, \mathtt{line}), (c_4, \mathtt{line})\} \\ \mathsf{links}(T) &= \{(c_1, c_3, c_2), (c_1, c_4, c_2)\} \\ \mathsf{neighbors}(T) &= \{(c_1, \{c_3, c_4\}), (c_2, \{c_3, c_4\})\} \end{aligned}$$

In representing the MIS model, $m \in \mathbf{MIS}$, we define $\beta_{\mathbf{MIS}} : \mathbf{MIS} \rightarrow \mathcal{P}(\mathbf{properties})$ and let $\beta_{\mathbf{MIS}}(m) = M$. The elements of M will differ from those of T:

$$\mathsf{components} \subseteq \mathbf{component} \tag{14}$$

$$\mathsf{attributes} \subseteq \mathbf{component} \times [0, 1] \tag{15}$$

$$\mathsf{functional\_states} \subseteq \mathcal{P}(\mathbf{component}) \tag{16}$$

$$\mathsf{initial\_probability} \subseteq \mathcal{P}(\mathbf{component}) \times [0, 1] \tag{17}$$

This yields another index set of components. Note that this set will not include information about all components of $\mathsf{components}(T)$, as m only considers line reliabilities. In addition, we have another function that yields a set of all of sets of components that compose a functional system state and a function describing the probability of a given state being the initial state.

$$\begin{aligned} \mathsf{components}(M) &= \{c_3, c_4\} \\ \mathsf{attributes}(M) &= \{(c_3, p_L), (c_4, p_L)\} \\ \mathsf{functional\_states} &= \{\{c_3, c_4\}, \{c_3\}, \{c_4\}\} \\ \mathsf{initial\_probability} &= \{(\{c_3, c_4\}, 1)\} \end{aligned}$$

Thus far, we have demonstrated how to determine the system properties that are described by certain models. We now demonstrate how to transform one model to another with the concretization operator, γ_{MIS}.

Let $\gamma_{MIS}(T) = m_T$. We identify unbound variables - those that have not been assigned a value - with the Fraktur typeface, e.g., $\mathfrak{p}$. m_T represents the set of MIS models, where all possible values have been assigned to each unbound variables. Equations 18 through 22 complete representation of the MIS model. The matrices corresponding to P_{c_2}, P_{c_3}, and P_{c_4} have been omitted for brevity.

TABLE III
STATE DEFINITION MATRIX FOR m_T

	Components			
States	c_1	c_2	c_3	c_4
S_1	1	1	1	1
S_2	1	1	1	0
S_3	1	1	0	1
S_4	1	1	0	0
S_5	1	0	1	1
S_6	1	0	1	0
S_7	1	0	0	1
S_8	1	0	0	0
S_9	0	1	1	1
S_{10}	0	1	1	0
S_{11}	0	1	0	1
S_{12}	0	1	0	0
S_{13}	0	0	1	1
S_{14}	0	0	1	0
S_{15}	0	0	0	1
S_{16}	0	0	0	0

$$\Pi_0 = [\mathfrak{s}_1, \cdots, \mathfrak{s}_{15}] \tag{18}$$

$$u = [\mathfrak{u}_1, \cdots, \mathfrak{u}_{15}] \tag{19}$$

$$P_{c_1} = \tag{20}$$

$$\begin{bmatrix} \mathfrak{p}_{\mathfrak{c}_1} & 0 & \cdots & \mathfrak{q}_{\mathfrak{c}_1} & 0 & \cdots \\ \vdots & \ddots & & \vdots & \ddots & \\ 0 & \cdots & \mathfrak{p}_{\mathfrak{c}_1} & 0 & \cdots & \mathfrak{q}_{\mathfrak{c}_1} \\ 0 & \cdots & 0 & 1 & 0 & \cdots \\ \vdots & & \vdots & \vdots & \ddots & \\ 0 & \cdots & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{21}$$

$$R = \Pi_0^T * P_{c_1} * P_{c_2} * P_{c_3} * P_{c_4} * u \tag{22}$$

The selection operator, σ_{MIS}, binds values to the unbound variables in a model-aware fashion. If we use σ_{MIS} to bind $\mathfrak{p}_{\mathfrak{c}_1} = \mathfrak{p}_{\mathfrak{c}_2} = 1$, this forces $\mathfrak{u}_5 = \cdots = \mathfrak{u}_{16} = 0$ and $\mathfrak{s}_5 = \cdots = \mathfrak{s}_{16} = 0$, since the failed states for these components are unreachable. Binding $\mathfrak{p}_{\mathfrak{c}_3} = \mathfrak{p}_{\mathfrak{c}_4} = p_L$, $\mathfrak{s}_1 = 1$, $\mathfrak{u}_1 = \mathfrak{u}_2 = \mathfrak{u}_3 = 1$, and $\mathfrak{u}_4 = 0$ suffice to generate an MIS model equivalent to m.

VI. CONCLUSION

In this paper, we propose a systematic approach to model transformation, based on concepts from abstract interpretation. We demonstrate a) how this approach can be used to abstract information about a system from a model of the system and b) how the information abstracted from one model can be concretized into a different model. Our approach is both correct–it always produces a set containing the desired model–and specific–the set of models produced is as small as possible.

This technique can be applied to a variety of application domains and can be used to generate cross-domain models, such as a model of the power consumption of a water distribution network. Dependability models can be generated from performance models, or vice versa, considerably facilitating the modeling and analysis of sustainable infrastructure.

While this approach is promising, our future work will aim to include more rigorous descriptions of the various operators involved. Extending and applying the method to diverse modeling formalisms is another near-term objective. Creation of a software tool for management of system modeling workflows is another extension planned to this work.

REFERENCES

[1] C. Ptolemaeus, ed., *System design, modeling, and simulation: using Ptolemy II*. Berkeley, Calif: UC Berkeley EECS Dept, 1. ed., version 1.02 ed., 2014.

[2] Y. Xiong, E. Lee, X. Liu, Y. Zhao, and L. Zhong, "The design and application of structured types in Ptolemy II," in *2005 IEEE International Conference on Granular Computing*, vol. 2, pp. 683–688 Vol. 2, July 2005.

[3] B. Lickly, C. Shelton, E. Latronico, and E. A. Lee, "A practical ontology framework for static model analysis," in *Proceedings of the Ninth ACM International Conference on Embedded Software*, EMSOFT '11, (New York, NY, USA), pp. 23–32, ACM, 2011.

[4] A. Goderis, C. Brooks, I. Altintas, E. A. Lee, and C. Goble, "Heterogeneous composition of models of computation," *Future Generation Computer Systems*, vol. 25, pp. 552–560, May 2009.

[5] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. Doyle, W. Sanders, and P. Webster, "The Möbius modeling tool," in *9th International Workshop on Petri Nets and Performance Models*, pp. 241–250, 2001.

[6] S. Gaonkar, K. Keefe, R. Lamprecht, E. Rozier, P. Kemper, and W. H. Sanders, "Performance and dependability modeling with Möbius," *SIGMETRICS Performance Evaluation Review*, vol. 36, pp. 16–21, Mar. 2009.

[7] C. Buchanan and K. Keefe, "Simulation debugging and visualization in the Möbius Modeling Framework," in *Quantitative Evaluation of Systems* (G. Norman and W. Sanders, eds.), no. 8657 in Lecture Notes in Computer Science, pp. 226–240, Springer International Publishing, Sept. 2014.

[8] J. De Lara and H. Vangheluwe, "AToM^3: A tool for multi-formalism and meta-modelling," in *FASE*, vol. 2, pp. 174–188, Springer, 2002.

[9] J. De Lara, H. Vangheluwe, and M. Moreno, "Using meta-modelling and graph grammars to create modelling environments," *Electronic Notes in Theoretical Computer Science*, vol. 72, no. 3, 2002.

[10] "CHESS Project Website - CHESS." http://www.chess-project.org/.

[11] "CONCERTO Project." http://www.concerto-project.org/.

[12] L. Montecchi, P. Lollini, and A. Bondavalli, "A reusable modular toolchain for automated dependability evaluation," in *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, pp. 298–303, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[13] V. Bonfiglio, L. Montecchi, F. Rossi, P. Lollini, A. Pataricza, and A. Bondavalli, "Executable models to Support Automated Software FMEA," pp. 189–196, IEEE, Jan. 2015.

[14] A. de Matos Pedro, D. Pereira, L. M. Pinho, and J. S. Pinto, "Towards a runtime verification framework for the ada programming language," in *Reliable Software Technologies–Ada-Europe 2014*, pp. 58–73, Springer, 2014.

[15] V. Vittorini, M. Iacono, N. Mazzocca, and G. Franceschinis, "The OsMoSys approach to multi-formalism modeling of systems," *Software and Systems Modeling*, vol. 3, pp. 68–81, Nov. 2003.

[16] G. Franceschinis, M. Gribaudo, M. Iacono, N. Mazzocca, and V. Vittorini, "Towards an object based multi-formalism multi-solution modeling approach," *Proceedings of the Second Workshop on Modelling of Objects, Components and Agents Aarhus (MOCA02), Denmark*, vol. 26, no. 27, pp. 47–65, 2002.

[17] E. Barbierato, M. Gribaudo, and M. Iacono, "Simthesyser: a tool generator for the performance evaluation of multiformalism models," tech. rep., Universití degli Studi di Napoli, Belvedere Reale di San Leucio 81100 Caserta, Italy, 2012.

[18] M. Iacono, M. Gribaudo, and E. Barbierato, "Exploiting multiformalism models for testing and performance evaluation in SIMTHESys," ACM, 2011.

[19] C. Kong and P. Alexander, "The Rosetta meta-model framework," in *10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, pp. 133–140, Apr. 2003.

[20] J. Streb and P. Alexander, "Using a lattice of coalgebras for heterogeneous model composition," in *Proceedings of the MoDELS Workshop on Multi-Paradigm Modeling*, pp. 27–38, 2006.

[21] N. Frisby, M. Peck, M. Snyder, and P. Alexander, "Model Composition in Rosetta," in *18th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, pp. 140–148, Apr. 2011.

[22] P. Alexander, "Rosetta: Standardization at the System Level," *Computer*, vol. 42, pp. 108–110, Jan. 2009.

[23] H. Vangheluwe, J. De Lara, and P. J. Mosterman, "An introduction to multi-paradigm modelling and simulation," in *Proceedings of the AIS'2002 conference (AI, Simulation and Planning in High Autonomy Systems), Lisboa, Portugal*, pp. 9–20, 2002.

[24] B. A. Davey and H. A. Priestley, *Introduction to lattices and order*. Cambridge university press, 2002.

[25] P. Smith, "The Galois connection between syntax and semantics," *University of Cambridge*, 2010.

[26] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999.

[27] M. N. Albasrawi, N. Jarus, K. A. Joshi, and S. Sedigh Sarvestani, "Analysis of reliability and resilience for smart grids," in *IEEE 38th Annual Computer Software and Applications Conference (COMPSAC)*, pp. 529–534, IEEE, 2014.