ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/262355819

ConceVE: Conceptual Modeling and Formal Validation for Everyone

Article *in* ACM Transactions on Modeling and Computer Simulation · February 2014

DOI: 10.1145/2567897

CITATIONS
9

READS
47

3 authors:

Ross Gore
Old Dominion University
84 PUBLICATIONS **1,184** CITATIONS
SEE PROFILE

Saikou Y. Diallo
Old Dominion University
162 PUBLICATIONS **2,248** CITATIONS
SEE PROFILE

José julian Padilla
University of Guadalajara
108 PUBLICATIONS **1,281** CITATIONS
SEE PROFILE

All content following this page was uploaded by Ross Gore on 08 May 2024.

The user has requested enhancement of the downloaded file.

ConceVE: Conceptual Modeling and Formal Validation for Everyone

ROSS GORE, SAIKOU DIALLO, and JOSE PADILLA, Old Dominion University

In this article, we present ConceVE, an approach for designing and validating models before they are implemented in a computer simulation. The approach relies on (1) domain-specific languages for model specification, (2) the Alloy Specification Language and its constraint solving analysis capabilities for exploring the state space of the model dynamically, and (3) supporting visualization tools to relay the results of the analysis to the user. We show that our approach is applicable with generic languages such as the Web Ontology Language as well as special XML-based languages such as the Coalition Battle Management Language.

Categories and Subject Descriptors: D.2.5 [**Software Engineering**]: Formal Verification

General Terms: Design, Model Checking

Additional Key Words and Phrases: Verification, M&S formalism, conceptual modeling, validity, interoperability

ACM Reference Format:
Ross Gore, Saikou Diallo, and Jose Padilla. 2014. ConceVE: Conceptual modeling and formal validation for everyone. ACM Trans. Model. Comput. Simul 24, 2, Article 12 (February 2014), 17 pages.
DOI: http://dx.doi.org/10.1145/2567897

1. INTRODUCTION

The process of developing, verifying, and validating models and simulations should be straightforward. A conceptual model is designed by a Subject Matter Expert (SME) from careful consideration of a problem and its domain. Then, it is realized via a source code simulation through the implementation of interfaces, data structures, and algorithms. Finally, the output of the simulation for a set of test cases is validated against historical data or other trusted sources [Pace 2000; Robinson 2004, 2006; Page et al. 1997].

Unfortunately, naively following this approach has pitfalls [Sargent 2005; Birta and Özmizrak 1996]. The design of a model that appeared complete and robust can become incoherent, incomplete, and potentially invalid during simulation implementation. The exactness required by the tools that execute the simulation creates complexity that might lead to the design of the conceptual model being hidden in irrelevant implementation details. Exploring alternative models and alternative modeling questions becomes impossible because SMEs are unable to identify the respective modifications that need to be made to the simulation [Law 2009].

An alternative approach is to attack the needed exactness head-on by employing a precise and unambiguous notation for SMEs to describe their models. This approach,

This work is supported by grants.
Author's address: Ross Gore, Saikou Diallo, and Jose Padilla, Virginia Modeling, Analysis and Simulation Center, Old Dominion University, University Blvd. Suffolk, VA 23435; email: ross.gore@gmail.com, {sdiallo, jpadilla}@odu.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 1049-3301/2014/02-ART12 $15.00
DOI: http://dx.doi.org/10.1145/2567897

known as formal methods, has had a number of major successes but is rarely used in practice due to three obstacles. First, the notation of formal specification languages has a mathematical syntax that makes them unintuitive and potentially intimidating. Second, these tools demand more investment of effort than can be expected from most SMEs. Finally, the tools force attention to mathematical details that do not reflect to SMEs the fundamental properties of their modeling question at hand [Kurshan 1997; Harbola et al. 2012; Gajski et al. 2009; Woodcock et al. 2009; Findler and Mazur 1990].

We present our approach, Conceptual-modeling & Validation for Everyone (ConceVE), which is a partial realization of our Modeling & Simulation System Development Framework (MS-SDF) [Tolk et al. 2013]. ConceVE utilizes Domain-Specific Languages (DSLs) for SMEs to provide a precise and expressive description of a model. Then, it replaces conventional formal analysis with a fully automatic analysis that gives immediate feedback to SMEs in familiar visualizations. ConceVE combines the incrementality and immediacy of small-scale domain-specific model design with the depth and clarity of traditional formal methods (e.g., model checking, theorem proving) to create a new style of modeling and validation accessible to a new, more general, audience.

2. THE PROPOSED APPROACH AND IMPLEMENTATION

The chief concern when developing a conceptual model is whether it and its resulting implementation will be correct. This concern is addressed through verification and validation (V&V) [Sargent 2005]. In previous work, we identified an approach to enable simulation developers to efficiently find and correct errors made while implementing an SME's conceptual model [Gore et al. 2012b]. In this article, our focus is not simulation implementation. Instead, we are concerned with providing an SME an efficient and usable approach to determine if his or her conceptual model is designed correctly.

Historically, a combination of requirements gathering, conceptual modeling, formal specification, and model checking has been employed to achieve this goal [Kurshan 1997]. However, the syntax of the tools that support these efforts is notoriously difficult for SMEs to understand because it is mired in mathematical notation [Jackson 1999]. Efficiently working with these tools requires an expert understanding of predicate-logic and set theory that is not feasible to expect from most SMEs.

To address this issue, we proposed the Modeling and Simulation System Development Framework (MS-SDF), which unifies the Systems Engineering processes of requirements capture, conceptual modeling, and V&V and extends them to Modeling and Simulation (M&S). The hallmark of this approach is its breadth, its formalism, and the ability of SMEs to leverage that formalism as they design, implement, and validate models. Figure 1 provides an overview of the components within the MS-SDF.

MS-SDF users begin by specifying their domain knowledge, conceptual model, and requirements for a system. Then, a simulation is generated from the conceptual model, and both the model and the simulation are checked separately against the requirements. If the model or simulation *is not consistent with* or *does not satisfy* the requirements, then the MS-SDF generates a counterexample depicting the inconsistency in a manner the user can easily understand. Otherwise, the user is notified that the model and simulation meet the specified requirements. Users refine their models incrementally by considering the feedback from the MS-SDF, and the resulting simulation reflects these updates.

ConceVE realizes the portion of the MS-SDF highlighted in blue. It employs (1) DSLs to enable SMEs to specify the structure, operations, and requirements of a conceptual model, (2) a translator to convert the SME's model specification to an formal model specification (DSL-To-Alloy Specification Language translation), (3) analysis to validate the SME's conceptual model (Alloy Analyzer), and (4) visualization

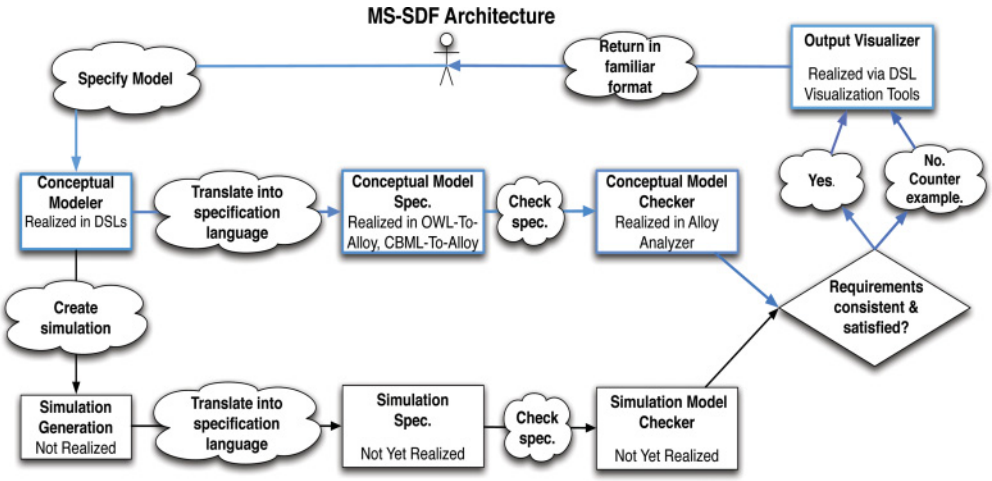

Fig. 1. MS-SDF overview.

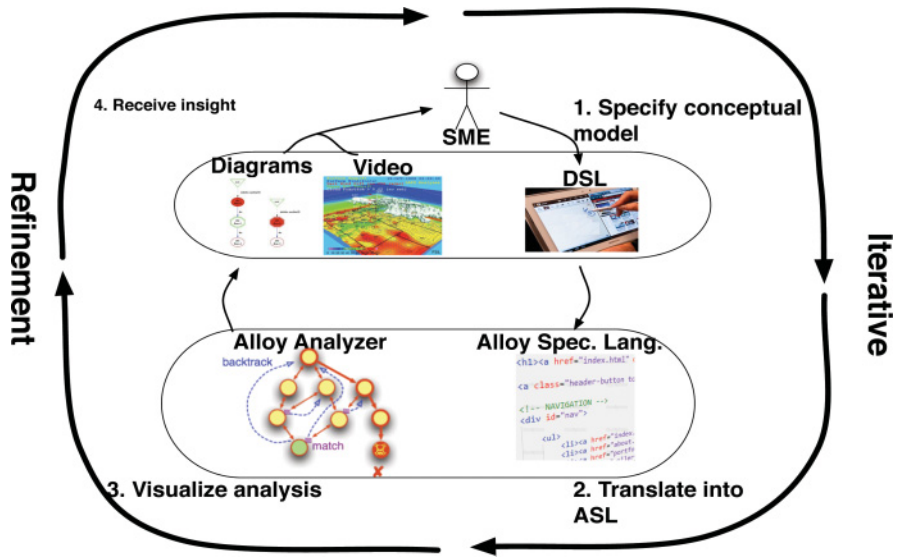

Fig. 2. Implementation of ConveVE overview.

tools to present the results of the validation to the SME in an easy-to-understand and familiar format (DSL visualization toolsets). The tools and the interactions within ConceVE that achieve these capabilities and realize the highlighted components in Figure 1 are shown in Figure 2.

The novelty of ConceVE lies in the use of DSLs, their supporting visualizations, and the formal method toolset, Alloy. DSLs are an ideal user interface. They provide a small, declarative language with expressive power that is shaped for a particular problem domain. By taking such a specific approach to problem solving, they achieve a strict separation of concerns of the domain semantics of the problem at hand from the program-specific semantics of specification [Kurtev et al. 2006]. Furthermore, most DSLs come with visualization support to help SMEs as they develop programs [Van

Deursen et al. 2000; Mernik et al. 2005]. By leveraging these toolsets for visualization, ConceVE has the ability to return the results of formal methods analysis to SMEs in a familiar format.

Alloy also offers a unique advantage that makes it ideal for ConceVE. Unlike traditional formal methods, Alloy analysis is not complete (infinite). For a specified model, the Alloy Analyzer only considers a finite space of cases. This significantly improves the efficiency of the validation analysis and provides users with immediate feedback. Furthermore, the space of cases examined is on the order of billions and thus offers a degree of coverage than is unattainable in testing. Combined in the novel manner depicted in Figure 2, the capabilities of DSLs and Alloy create a more robust means for SMEs to design and validate domain-specific models than existing alternatives. Next, we review background material for the ConceVE components.

3. BACKGROUND

In this section, we provide a summary of the underlying technologies in ConceVE. First we review DSLs and then we discuss several different formal methods, including formal specification, model checking, theorem proving, and the constraint solver Alloy.

3.1. Domain-Specific Languages

DSLs are small languages, targeted toward a particular discipline, that offer a *separation of concerns*. Achieving a separation of concerns allows users to distinguish the semantics of the problem domain at hand from the programming-specific semantics. This enables users tasked with a particular programming problem to encounter less of a learning curve and express more concise solutions with a DSL as opposed to a General-purpose Programming Language (GPL) [Van Deursen et al. 2000; Mernik et al. 2005]. Furthermore, studies show that the communication between domain experts improves and less code maintenance is needed when DSLs are employed in practice [North et al. 2006; Visser 2008].

The strong relationship between DSLs and conceptual models makes a DSL the best syntax for a user to specify his or her conceptual model. In Kurtev et al. [2006], a DSL is defined as "a set of coordinated [conceptual] models." We support this definition. DSLs have a clearly identified, concrete problem domain. In contrast, GPLs cover multiple domains. Programs in a DSL represent concrete states of affairs in this domain; they are models. A conceptualization of the domain is an abstract entity that captures the commonalities among the possible state of affairs. It introduces the basic abstractions of the domain and their mutual relations. Once such an abstract entity is explicitly represented by the user in the syntax of the DSL, the conceptual model is realized.

The eXtensible Markup Language (XML) provides a platform to develop DSLs. As a result, there has been a host of M&S-related XML-based DSLs that have been successfully developed to address problems in numerous disciplines [McGinnis and Ustun 2009; Chandrasekaran et al. 2002; Brutzman et al. 2002]. ConceVE looks to combine the extensibility and flexibility of XML with the consistency and satisfiability results created by employing formal methods.

3.2. Formal Methods

Employing formal methods in the field of M&S validation is not new. In safety-critical software, where billions of dollars and millions of lives depend on correctness, formal methods are routinely applied [Bowen and Stavridou 1993]. This requires specifying a model in a formal language and using the properties of that language to ensure that every statement of the model is either part of the language or can be produced by the language [Tolk et al. 2013]. The term *formal methods* encompasses a family of approaches that sometimes refer to the tools, processes, or languages used in support

of validation [Tofts and Birtwistle 1998; Aldini et al. 2001]. In this section, we review several relevant formal methods that facilitate an understanding of our approach.

3.2.1. Formal Specification. Formal specification is the process of describing a system and its desired properties using a language with a mathematically defined syntax and semantics. Some formal specification languages such as Z [Spivey 1988], VDM [Jones 1986], and Larch [Guttag et al. 1993] focus on specifying the behavior of sequential systems, where states are described in mathematical structures such as sets, relations, and functions. Other methods such as CSP [Hoare 1978], CCS [Milner 1982], Statecharts [Harel 1987], Temporal Logic [Lamport 1994], and I/O automata [Lynch and Tuttle 1987] focus on specifying system behaviors in terms of sequences, trees, or partial orders of events. Common to all of these methods is the use of the mathematical concepts of abstraction and composition. Unfortunately, this syntax makes specification languages intimidating and unintuitive to most SMEs.

3.2.2. Model Checking and Theorem Proving. Model checking is an automated means to check a formal specification for correctness. It requires a user to formulate a property as a predicate over variable values. Then, it automatically checks to see if the property holds in the specification. For example, a property stipulating that a variable x always be positive and that a variable y always be strictly smaller than x can be formulated as $x > 0 \wedge y < x$ [Clarke et al. 2000].

Two distinct approaches using properties are used to assess correctness: (1) pre/post condition and (2) invariant assertion.

(1) Pre/post condition approaches formulate the correctness problem as the relationship between a formula that is assumed to hold at the beginning of program execution, denoted ΦPRE, and a formula that should hold at the end of program execution, denoted ΦPOST. Assessing the correctness of the program involves determining whether the semantics of the program establishes ΦPOST given ΦPRE.
(2) Invariant assertion-based approaches define the correctness of a simulation by verifying that a user-specified Boolean formula, ΦINV, holds throughout simulation execution.

These approaches can be performed within minutes for small- and intermediate-sized model specifications. However, for large specifications, analysis can take hours and even days, significantly limiting the utility of model checkers [Burch et al. 1992]. Theorem proving can be a more effective means to validate a large model specification. Theorem provers enable a system and its desired properties to be specified in mathematical logic that defines a set of axioms and inference rules. Theorem proving is defined as the process of finding a proof of a property from the axioms of the system. The steps in the proof are derived from definitions and intermediate lemmas derived from the logics axioms and rules [Bonacina 2010].

However, theorem provers are interactive, not automatic. Since the user is part of the theorem proving process, the proof construction is slow and often error prone. However, in the process of finding the proof, invaluable user insight into the specification can be gained.

3.2.3. The Alloy Specification Language and Analyzer. The Alloy Specification Language (ASL) is a language based on first-order logic for expressing behavioral constraints. Alloy treats relations as first-class citizens and uses relational composition as a powerful operator to combine entities. The essential constructs of Alloy are as follows [Jackson 2006]:

—*Signatures:* Signatures reflect a collection of relations (called *fields*) and a set of constraints on their values. A signature may inherit fields and constraints from other signatures.
—*Predicates:* Predicates capture behavior constraints through general formulas.
—*Facts:* Facts are formulas that take no arguments and impose global constraints on the relations and the objects.
—*Assertions:* Assertions specify intended properties within a model.

The Alloy Analyzer is a constraint solver. Given a model composed of signatures, predicates, and facts and one or more specified assertions, it performs two types of analyses: (1) model instantiation and (2) counterexample generation [Jackson 2006].

(1) *Model instantiation* attempts to generate a version of a model specified in ASL that satisfies all of the predicates and facts given by the user. In general, model instantiation helps catch errors where the user has overspecified the model, by reporting, contrary to the user intent, when no model instance satisfying all of the specified predicates and facts exists.
(2) *Counterexample generation* catches errors of underconstraint by showing model instances that are acceptable given the specified scope, structure, operation, and preconditions of the model but that violate a user-specified assertion.

Together, the model instantiation and counterexample generation work within ConceVE to enable an incremental verification process. The user starts with a minimal conceptual model and performs a variety of instantiations to detect overconstraint. Intended consequences are formulated, with counterexamples suggesting additional constraints to be added to the specification. This process, especially visualizing it through DSL toolsets, helps produce a conceptual model that has only desired properties.

3.3. Related Research

Effectively employing formal methods in the conceptual modeling process is not a new problem. Here, we review existing work in this research area and identify those characteristics of ConceVE that make it novel. Recall that formal methods are mired in mathematical syntax and thus intimidating and difficult for most users. As a result, several different structures in a variety of domains have been employed as interface to formal methods for users. In the field of genetics, where large datasets make simulation expensive, researchers combined state transition graphs derived from qualitative simulation with model checking to create a more efficient mechanism to derive insight into conceptual models [Batt et al. 2005]. Similarly, in biology and biochemical simulations, query languages have been used to enable users to determine if a conceptual model satisfies a given property [Park et al. 2002]. Queries have also been combined with graph grammars to specify and check properties in control systems [Copstein et al. 2000].

In related work, a structure protocol language was successfully mapped to the High-Level Architecture (HLA) interoperability standard. The structured protocol language enables users to detect anomalies, race conditions, and deadlocks within component integration [Allen et al. 1998]. Researchers working on a similar problem—the composition of Web services,—have debated the benefits and drawbacks of using more established formalisms (i.e., Pi calculus and Petri nets) to express queries and receive feedback about anomalies, race conditions, and deadlocks [Van der Aalst 2005; Ter Beek et al. 2007].

ConceVE differs from each of these previous efforts in terms of (1) its breadth of scope, (2) the immediacy of its analysis (timeliness of response time), and (3) its emphasis on visualization. Unlike previous domain-specific research efforts, ConceVE (and the

MS-SDF) represents a framework, not a tool. Furthermore, this tool is not domain specific; it can be applied to any problem. In addition, previous efforts have employed traditional model checkers where user response time is measured in minutes, hours, and sometimes days. However, ConceVE takes advantage of the advances in finite space constraint solving encapsulated within Alloy. As a result, user response time is typically measured in seconds. Finally, ConceVE emphasizes visualization of model checking output through domain-specific tools. This is significant. Researchers have found that creating a model checking output interface with a low barrier of entry is just as important to practitioners using model checkers as the language used to express a formal specification [Davies et al. 2006].

4. CASE STUDIES

We conducted two case studies tp evaluate ConceVE, each featuring a different and established DSL. The first case study features the Web Ontology Language (OWL) used to process data on the Semantic Web. In the second case study, the Coalition Battle Management Language (C-BML) for command and control of military operations is featured. Each case study showcases the benefits of developing models in ConceVE as opposed to requiring users to directly interact with formal methods.

4.1. Web Ontology Language

The Semantic Web extends the current Web by giving content a well-defined meaning and enabling cooperation. The Resource Description Framework (RDF) is a foundational technology for processing Semantic Web metadata. RDF descriptions provide a simple ontology system to support the exchange of knowledge via the RDF Schema, which consists of primitive *properties* and *type* description constructs [Antoniou and Harmelen 2009].

The OWL language has extended RDF by providing (1) richer constructors for building type (*class*) and *property* descriptions and (2) capabilities to ensure that those constructs adhere to specified axioms (*statements*). These three constructs (classes, properties, and statements) interact according to the following [Antoniou and Harmelen 2009]:

—*Classes* are interpreted as sets of objects that represent the individuals in the domain of discourse.
—*Properties* are binary relations that link classes and statements, and are interpreted as sets of tuples, which are the subsets of the cross product of the objects in the domain of discourse.
—*Statements* reflect facts about properties that classes hold. Statements are always true.

The case study that follows features OWL as a DSL used to describe the relationships of classes within a user's conceptual model. The utility of describing simulation conceptual models using OWL has been established by previous research [Miller et al. 2004]. We chose to employ OWL in this case study because of its applicability to M&S, widespread popularity, and structure. Although our algorithms only describe how to translate a conceptual model described in OWL to Alloy, it also applies to any conceptual model expressed in a syntax that can also be translated into OWL. The implementation of our algorithms is available [Gore et al. 2012a].

4.1.1. Parsing and Translating OWL. ConceVE employs an XML parser to partition an OWL 2.0 document into classes, properties, and statements [Motik et al. 2009]. This parser is modeled after existing OWL Application Programming Interfaces [Knublauch 2006; Horridge et al. 2007]. Once parsed, each class, property, and statement is

ALGORITHM 1: *OWL-To-Alloy*—The main function used to convert OWL to Alloy.

Input: C, P, S—the respective set C of classes, a set P of properties, and a set S of statements.
Output: Σ—an ASL model.

```
begin
    for each c ∈ C do
        Σ ← Σ + GenAlloySig(c, S);
    end
    for each s ∈ S do
        if s.predicate.name = "disjointWith" then
            if s.subject.subClassOf ≠ s.object.subClassOf then
                Σ ← Σ+ "pred {no c1:" +s.subject.name+", c2:" +s.object.name+ "| c1 = c2}";
            end
        end
        if s.predicate.name = "complementOf" then
            Σ ← +"pred {"+s.subject.name+ "=" +C − s.object.name+" }";
        end
    end
    return Σ;
end
```

converted into a *signature*, *predicate*, *fact*, or *assertion* in the ASL. This conversion is performed through Algorithm 1, *OWL-To-Alloy*. Algorithm 1 converts the inputs classes (denoted as C), properties (denoted as P), and statements (denoted as S) into an ASL model.

OWL-To-Alloy begins by initializing Σ, a model written in ASL. Then, for each class c, there will be a corresponding signature of the same name added to Σ constructed by applying $GenAlloySig(c, S)$, which is described later in this section.

Once $GenAlloySig(c, S)$ has been applied to each class, the statements parsed from the OWL document are considered. An OWL statement s has a name ($s.name$) and is composed of three segments ($s.subject$, $s.predicate$, and $s.object$). Each segment has a name (i.e., $s.subject.name$) and is a descendant from another OWL class ($s.object.subClassOf$). The second half of Algorithm 1 ensures that classes that are specified to be *disjoint from* and *complements of* each other are translated and encoded correctly in the ASL model Σ.

Recall that *OWL-To-Alloy* uses the algorithm *GenAlloySig* to generate a signature for a class c parsed from an OWL document. It begins by producing a signature that is named after the input class c. Then, *GenAlloySig* determines whether class c is subclass of another by checking if $c.subClassOf$ is not empty. If c is a subclass of another, then the algorithm determines the relationship between the classes.

GenAlloySig also encodes OWL properties into the signature for a class c. This is achieved through two helper functions: $domain(p, S)$ and $range(p, S)$. For each property p, $domain(p, S)$ is used to determine whether c is its domain. If it is, then the $range(p, S)$ algorithm looks up the range of p.

The subalgorithms $domain(p, S)$ and $range(p, S)$ are very similar in structure. In Algorithm 3, we describe $domain(p, S)$. Given the description of $domain(p, S)$, the pseudocode comprising $range(p, S)$ can be easily derived. $domain(p, S)$ is used to obtain the domain of the property p. For a property p, if there is a statement s that meets the specified conditions, then the name of $s.object$ is considered to be the domain of p. Otherwise, the domain of p's parent is also its domain.

Algorithms 1 through 3 work in combination to translate an OWL document into an Alloy model Σ. In the next section, we will apply these algorithms in a case study that elucidates the validation capabilities that an SME gains by employing ConceVE.

ALGORITHM 2: *GenAlloySig*—The helper function used to generate Alloy signatures.

Input: c, S, P—a class c, a set S of Statements and a set P of Properties.
Output: Σ—a signature to an ASL model.
begin
  $\sigma \leftarrow$ "sig" $+c.name$;
  if $c.subclassOf \neq \omega$ **then**
    $\sigma \leftarrow \sigma +$ "extends"$+c.subClassOf.name$;
    $\sigma \leftarrow \sigma +$ "{";
    for *each* $p \in P$ **do**
      if $domain(p, S) = c.name$ **then**
        $\sigma \leftarrow \sigma + p.name +$ ":"$+range(p, S)$;
        $\sigma \leftarrow \sigma +$ "}";
      end
    end
  end
  return σ;
end

ALGORITHM 3: *domain*—A subalgorithm function used to obtain the domain of p.

Input: p a property p, a set S of Statements.
Output: Σ—the domain of p.
begin
  if $s.subject = p \wedge s.predicate =$"domain" **then**
    $p.domain \leftarrow s.object.name$;
  end
  else
    $p.domain \leftarrow domain(parent(p), S)$;
  end
  return $p.domain$;
end

4.1.2. An Erroneous OWL Ontology. Existing tools for ontology reasoning struggle to provide rich answers that refer to a particular ontology instantiation as opposed to an ontology as a whole [Golbreich 2004]. For example, one of the most advanced ontological reasoners, the OWL Instance Store (iS), only supports a very limited form of answering instance retrieval queries with respect to an ontology and a set of axioms asserting class-instance relationships. Although the iS able to process much larger numbers of individuals, it cannot assert that OWL instance x is related via OWL property p to OWL instance y [Bechhofer et al. 2005]. This analysis is straightforward for Alloy and thus ConceVE [Jackson 1999, 2006]. Here, we demonstrate (1) how the lack of any validation capability is problematic and (2) how it is addressed with ConceVE.

An erroneous OWL ontology describing Mammals is shown in Figure 3. Recall that in this case study, OWL is being used as a DSL to describe the relationships of instances and classes within a user's conceptual model. There are four classes defined: *Mammal*, *Male*, *Doe*, and *Female*. *Mammal* is the base class, *Male* and *Female* are disjoint subclasses of the base class, and *Doe* is a subclass of *Female*. There are also four properties in the ontology: *hasFather*, *hasParent*, *hasChild*, and *femaleHasFather*. The properties *hasParent* and *hasChild* are inverses of each other, the property *hasFather* is a subproperty of *hasParent*, and *femaleHasFather* is a subproperty of *hasFather*. Applying Algorithms 1 through 3 to this ontology yields the Alloy model shown in Figure 4.

```
<owl:Class rdf:ID="Mammal"/>
<owl:Class rdf:ID="Male">
     <rdfs:subClassOf rdf:resource="#Mammal"/>
</owl:Class>

<owl:Class rdf:ID="Female">
     <rdfs:subClassOf rdf:resource="#Mammal"/>
     <owl:disjointWith rdf:resource="#Male"/>
</owl:Class>

<owl:Class rdf:ID="Doe">
     <rdfs:subClassOf rdf:resource="#Female"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="hasParent">
     <rdfs:domain rdf:resource="#Mammal"/>
     <rdfs:range rdf:resource="#Mammal"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: ID="hasFather">
     <rdfs:subPropertyOf rdf:resource="hasParent"/>
     <rdfs:domain rdf:resouce="Mammal"/>
     <rdfs:range rdf:resource="#Male"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: ID="hasChild">
     <rdfs:inverseOf rdf:resource="hasParent"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf: ID="femaleHasFather">
     <rdfs:subPropertyOf rdf:resource="hasFather"/>
     <rdfs:domain rdf:resouce="Female"/>
     <rdfs:range rdf:resource="#Doe"/>
</owl:ObjectProperty>
```

Fig. 3. An erroneous OWL ontology.

```
sig Mammal{
     hasParent: Mammal,
     hasFather: Male,
     hasChild: Mammal
}

sig Female extends Mammal {
     femaleHasFather: Doe
}
sig Male extends Mammal {}
sig Doe extends Female {}
pred inverseOf {
     hasParent = ~hasChild
}
pred subPropertyOf {
     (all m:Mammal | m.hasFather in m.hasParent) and
     (all f:Female.femaleHasFather | f in Animal.hasFather)
}
```

Fig. 4. An Alloy translation of Figure 3.

ConceVE generates four signatures and two predicates from the OWL ontology in Figure 3. *Female* and *Male* both extend from the *Mammal* signature, and *Doe* extends from the *Female* signature. Furthermore, the four properties in the OWL Document are included within the Alloy model. It is important to note that this entails converting the OWL keywords (**inverseOf** and **subPropertyOf**) into Alloy predicates. The former is used to encode that *hasParent* is the inverse of *hasChild*, whereas the latter reflects that the range of *hasFather* is a subset of *hasParents* and the range of *femaleHasFather* is a subset of *hasFather*.

The fault in the ontology lies in the description of the property *femaleHasFather*. By making *Male* the range of *hasFather* and *Doe* the range of *femaleHasFather*, the SME has created a subproperty that applies to nothing; the two ranges (*Male* and *Doe*) are always completely disjoint!

This type of analysis is at the foundation of formal methods such as those employed in ConceVE. ConceVE immediately identifies the disjoint ranges and reports them to the user via the Protege plugin OntoViz [Sintek 2003]. OntoViz is capable of depicting the error detected by Alloy as a Venn diagram revealing the complete lack of overlap between *Male* and *Doe*. Through this familiar visualization, the user can correct the range of the *femaleHasFather* property to *Male*. Once correctly modified, ConceVE offers two capabilities that increase the users confidence: (1) the Venn diagram regarding the two disjoint sets no longer appears, and (2) a visualization of a possible model instantiation showing the application of the *hasFather* relation and *femaleHasFather* relation is provided.

4.2. Coalition Battle Management Language

C-BML is an unambiguous language used to (1) command and control forces and equipment conducting military operations and (2) provide for situational awareness and a shared, common operational picture. C-BML employs XML to define rigorous, well-documented, situational contexts, mission tasks, and entity reports that reflect a subset of the Command & Control Information Exchange Data Model (C2IEDM) [Tolk et al. 2007].

There is a need to check C-BML documents to determine if tasks specified in missions are contradictory to military doctrine or can lead to failure [Tolk et al. 2006]. However, this need has not been addressed. In what follows, we present a solution to this problem with ConceVE.

4.2.1. Parsing and Translating C-BML. ConceVE employs an XML parser to partition a C-BML document into its three components: *contexts*, *tasks* and *reports*. Once parsed from C-BML, each context, task, and report is converted into a formal specification in ASL. This conversion is performed through the application of a series of algorithms. The first algorithm, *CBML-To-Alloy*, marshals the inputs (*contexts* denoted as C, *tasks* denoted as T, and *reports* denoted as R) into a series of subalgorithms that generate ASL signatures and predicates to form an Alloy model Σ.

CBML-To-Alloy begins by calling *Context-To-Alloy* for each context included in the C-BML document. *Context-To-Alloy* takes each component of a C-BML context element and translates it into an Alloy signature with the appropriate attributes. The algorithm checks for every possible element in a C-BML context component; if the element exists, it is translated into an attribute within the Alloy signature.

Once *Context-To-Alloy* has encoded every C-BML context, *CBML-To-Alloy* employs *Task-To-Alloy* to translate each C-BML task into an Alloy predicate. First, the algorithm creates a skeleton for the Alloy predicate. Then, it generates a check to ensure that the predicate is only applied when the task should be performed, and it

ALGORITHM 4: *CBML-To-Alloy*—The main function used to convert C-BML to Alloy.

Input: C, T, R—the respective set C of C-BML contexts, a set T of C-BML tasks, and a set R of C-BML reports.
Output: Σ—an ASL model.
begin
  for *each* $c \in C$ **do**
    $\Sigma \leftarrow \Sigma +$ *Context-To-Alloy(C)*;
  end
  for *each* $t \in T$ **do**
    $\Sigma \leftarrow \Sigma +$ *Task-To-Alloy(T)*;
  end
  for *each* $r \in R$ **do**
    $\Sigma \leftarrow \Sigma +$ *Report-To-Alloy(R)*;
  end
  return Σ;
end

creates an action that updates the state of (1) the tasked entity, (2) the referred-to area, and (3) the entity referred to in the task.

Finally, each report is processed using the function *Report-To-Alloy*. Each report serves as an axiom that gives the initial position and end position of units during the mission. These reports are translated into predicates that are enforced as facts for the initial and end state of the Alloy model Σ.

Algorithms 4 through 7 work in combination to translate a C-BML document into an Alloy model Σ. In the next section, we will apply these algorithms in a case study that elucidates how ConceVE enables validation in C-BML without exposing SMEs directly to formal methods.

4.2.2. Multinational Company Scenario. The Commanding Officer and Multinational Company C-BML plan describes the steps required to take possession of Oscar 1, a house surrounded by an enemy force. The steps required to complete the mission are:

(1) Move to a secure area and place a device monitoring Oscar 1.
(2) Move along a discrete path to a location close to Oscar 1 to settle in for an assault.
(3) Assault the enemies at Oscar 1, and seize possession of the house.
(4) Fall north of Oscar 1, and install a monitoring device to the North, West, and East of the seized location.

If steps 1 through 4 are completed in order successfully, the invasion is complete and the Multinational Company mission is considered successful. If the enemy ENI forces overtake the company, then the mission is considered a failure. The model is used by military organizations to explore various Rules Of Engagement (ROEs) and resulting casualty scenarios for the Multinational Company.

The C-BML document specifies each of the entities included in the scenario: the constituents of the Multinational Company (U.S. and French Platoon), Oscar 1, the phase line, the enemy forces, and the environment (forests, hills, etc). Then, the location and associations between these entities are flushed out via reports in C-BML. Finally, each step within the mission is expressed in C-BML tasks. Using Algorithms 4 through 7, the C-BML scenario can be translated into an Alloy model, and intricacies within the mission can be explored. For example, in this case study, we explore how ROEs affect the success of the Multinational Company. ROEs are the directives issued by competent military authority that delineate the circumstances and limitations under which military forces will initiate and/or continue combat engagement with other forces

ALGORITHM 5: *Context-To-Alloy*—The function used to convert C-BML contexts to Alloy signatures.

Input: C—a set of C-BML contexts.
Output: γ—a subset of an ASL model.
begin
  for *each* $c \in C$ **do**
    $\gamma \leftarrow \gamma +$ "one sig" $+ c.name +$ "extends" $+ c.supportingType +$ " {";
    $\gamma \leftarrow \gamma +$ "oid:" $+ c.oid$;
    if $c.commandFunctionIndicatorCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "commandFunctionIndicatorCode:" +;
      $c.commandFunctionIndicatorCode$;
    end
    if $c.serviceCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "serviceCode:" $+ c.serviceCode$;
    end
    if $c.categoryCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "categoryCode:" $+ c.categoryCode$;
    end
    if $c.armCategoryCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "armCategoryCode:" $+ c.armCategoryCode$;
    end
    if $c.decoyIndicatorCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "decoyIndicatorCode:" $+ c.decoyIndicatorCode$;
    end
    if $c.subCategoryCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "subCategoryCode:" $+ c.subCategoryCode$;
    end
    if $c.sizeCode \neq \omega$ **then**
      $\gamma \leftarrow \gamma +$ "sizeCode:" $+ c.sizeCode$;
    end
    $\gamma \leftarrow \gamma +$ "}";
  end
end

ALGORITHM 6: *Task-To-Alloy*—The function used to convert C-BML tasks to Alloy predicates.

Input: T—a set of CBML tasks.
Output: τ—a subset of an ASL model.
begin
  for *each* $t \in T$ **do**
    $\rho \leftarrow \rho +$ "pred TASK_" $+ t.name +$ "[s, s': State] {";
    $\tau \leftarrow \tau +$ "s.current" $+ t.name +$ "implies s'." $+ t.name +$ "_AREA = s." $+ t.name +$
    "_AREA - s." $+ t.taskeeWho$;
    $\tau \leftarrow \tau +$ "}";
  end
end

encountered. These directives are intended to reduce the chance of friendly fire incidents and recognize international law regarding the conduct of war, particularly the need to protect civilians.

However, without validation, these restrictions can limit the ability of commanders and companies to accomplish mission plans. Using ConceVE, SMEs can explore if all of the tasks within the mission can be carried out if the Multinational Company never uses force against the Enemy ENI force. Figure 5 reflects output from Alloy, visualized

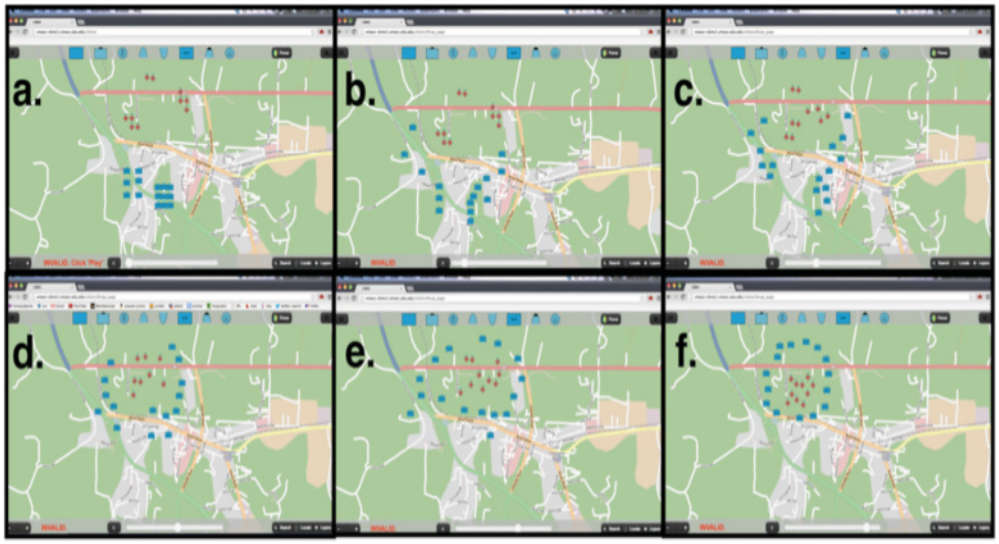

Fig. 5. A ConceVE counterexample generated by Alloy, visualized by a C-BML support tool.

ALGORITHM 7: *Report-To-Alloy*—The function used to convert C-BML reports to Alloy predicates.

Input: R—an ordered set of reports.
Output: ρ—a subset of an ASL model.
begin
  $\tau \leftarrow \tau +$ "fact initialState{ init[first] }";
  $\tau \leftarrow \tau +$ "pred init [s, s': State] {";
  for *each* $r \in R$ **do**
    $\rho \leftarrow \rho +$ "s." $+ r.reportedWhoRef +$ ".coordinates $+ =$ ";
    for *each* $point \in r.location$ **do**
      $\rho \leftarrow \rho +$ "Lat["$+ point.latitude +$ "] $\rightarrow$ Long[" $+ point.longitude +$ "]";
    end
  end
  $\rho \leftarrow \rho +$ "}";
end

by a C-BML support tool, which reveals that all of the tasks cannot be carried out if the Multinational Company does not use force. The Multinational Company (shown in blue) advances toward the Enemy ENI force but cannot assault the enemy and seize possession of the house. The company reaches an impasse where they surround the enemy but cannot progress further.

These types of subtleties in mission planning are notoriously difficult to explore at the scenario specification level but incredibly important to achieve success [Spiegel et al. 2005]. However, employing ConceVE enables (1) different ROEs to be tested at the specification level and (2) users to determine exactly if and how the ROEs inhibit or guarantee mission success. This is a crucial step in enabling SMEs in military domains, regardless of their familiarity with formal methods, to produce valid conceptual models.

5. CONCLUSION

The process of developing, verifying, and validating models and simulations should be straightforward. Unfortunately, following conventional development approaches can render a model design that appeared complete and robust into an incomplete, incoherent, and invalid simulation during implementation. An alternative approach is to attack the needed exactness head-on by employing formal methods for SMEs to describe their models. However, this approach is rarely used in practice due to the intimidating syntax and unfamiliar semantics of model checkers and theorem provers.

As a result, we developed the MS-SDF. Here, we implemented a critical portion of the MS-SDF via ConceVE. ConceVE leverages DSLs for model specification, their supporting visualization mechanisms, and the Alloy toolset to realize an approach to developing valid and verifiable conceptual models that is accessible to all SMEs regardless of their familiarity with formal methods. Our claim that ConceVE makes formal methods more usable for SMEs are based on its ability to provide domain-specific wrappers around the ASL and the output of the Alloy Analyzer. Although this is not a statistical evaluation or a user study, the utility of DSLs is well established. In future work, we will further corroborate this claim with more quantitative user evaluations.

ConceVE is constrained by the underlying technologies used to support DSLs and model checking. Within ConceVE, DSL to Alloy Specification translation is DSL specific—a specific translator must be developed for each DSL. Currently, the only DSLs that ConceVE supports are OWL and CBML. This is a limitation of ConceVE that our future work will look to address. We have established that DSL to Alloy Specification translation is possible, and in the future, we will make it accessible to more conceptual modeling DSLs.

In terms of its model-checking capabilities, ConceVE is constrained by the Alloy Analyzer. Analysis within the Alloy Analyzer is not complete; it only examines a finite space of test cases. However, the space of test cases explored is huge, on the order of billions, and it therefore offers a degree of coverage unattainable in testing. In the future, we will continue to realize the MS-SDF by adding the ability to automatically generate valid simulations from specified conceptual models.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

ACKNOWLEDGMENTS

We gratefully acknowledge help for our friends and colleagues at the Virginia Modeling, Analysis and Simulation Center at Old Dominion University.

REFERENCES

Alessandro Aldini, Marco Bernardo, Roberto Gorrieri, and Marco Roccetti. 2001. Comparing the QoS of Internet audio mechanisms via formal methods. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 11, 1, 1–42.

Robert J. Allen, David Garlan, and James Ivers. 1998. Formal modeling and analysis of the HLA component integration standard. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'98/FSE-6)*. ACM, New York, NY, 70–79. DOI:http://dx.doi.org/10.1145/288195.288251

Grigoris Antoniou and Frank van Harmelen. 2009. Web ontology language: Owl. *Handbook on Ontologies*, 91–110.

Grégory Batt, Delphine Ropers, Hidde De Jong, Johannes Geiselmann, Radu Mateescu, Michel Page, Dominique Schneider, et al. 2005. Analysis and verification of qualitative models of genetic regulatory networks: A model-checking approach. In *International Joint Conference on Artificial Intelligence*, Vol. 19. Lawrence Erlbaum Associates Ltd., 370.

Sean Bechhofer, Ian Horrocks, and Daniele Turi. 2005. The OWL instance store: System description. In *Automated Deduction–CADE-20*. Springer, 177–181.

Louis G. Birta and F. Nur Özmizrak. 1996. A knowledge-based approach for the validation of simulation models: The foundation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 6, 1, 76–98.

Maria Paola Bonacina. 2010. On theorem proving for program checking: Historical perspective and recent developments. In *Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming*. ACM, 1–12.

Jonathan Bowen and Victoria Stavridou. 1993. Safety-critical systems, formal methods and standards. *Software Engineering Journal* 8, 4, 189–209.

Don Brutzman, Michael Zyda, J. Mark Pullen, and Katherine L. Morse. 2002. *Extensible Modeling and Simulation Framework (XMSF): Challenges for Web-Based Modeling and Simulation*. Naval Postgraduate Schoo, Monterey, CA.

Jerry R. Burch, Edmund M. Clarke, Kenneth L McMillan, David L. Dill, and Lain-Jinn Hwang. 1992. Symbolic model checking: 1020 states and beyond. *Information and Computation* 98, 2, 142–170.

Senthilanand Chandrasekaran, Gregory Silver, John A. Miller, Jorge Cardoso, and Amit P. Sheth. 2002. XML-based modeling and simulation: Web service technologies and their synergy with simulation. In *Proceedings of the 34th Conference on Winter Simulation: Exploring New Frontiers*. Winter Simulation Conference, 606–615.

Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. 2000. *Model Checking*. MIT Press.

Bernardo Copstein, Michael da Costa Móra, and Leila Ribeiro. 2000. An environment for formal modeling and simulation of control systems. In *Proceedings of the 33rd Annual Simulation Symposium (SS 2000)*. IEEE, 74–79.

Islay Davies, Peter Green, Michael Rosemann, Marta Indulska, and Stan Gallo. 2006. How do practitioners use conceptual modeling in practice? *Data and Knowledge Engineering* 58, 3, 358–380.

Nicholas V. Findler and Neal M. Mazur. 1990. A system for automatic model verification and validation. *Transactions of the Society for Computer Simulation International* 6, 3, 153–172.

Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. 2009. *Embedded System Design: Modeling, Synthesis and Verification*. Springer.

Christine Golbreich. 2004. Combining rule and ontology reasoners for the Semantic Web. *Rules and Rule Markup Languages for the Semantic Web*, 6–22.

Ross Gore, Saikou Diallo, and Jose Padilla. 2012a. Owl to Alloy Documentation. Old Dominion University, Suffolk, VA. VMASC-2012-09.

Ross Gore, Saikou Diallo, and Jose Padilla. 2012b. Statistical debugging for simulations. [PhD Thesis]. Dept. of Computer Science. University of Virginia. Charlottesville, VA.

John V. Guttag, James J. Horning, Withs J. Garl, Kevin D. Jones, Andres Modet, and Jeannette M. Wing. 1993. Larch: Languages and tools for formal specification. In *Texts and Monographs in Computer Science*. Citeseer.

Aditya Harbola, Deepti Negi, and Deepak Harbola. 2012. Infinite automata and formal verification. *International Journal of Advanced Research in Computer Science and Software Engineering* 2, 3, 285–289.

David Harel. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming* 8, 3, 231–274.

Charles Antony Richard Hoare. 1978. Communicating sequential processes. *Communications of the ACM* 21, 8, 666–677.

Matthew Horridge, Sean Bechhofer, and Olaf Noppens. 2007. Igniting the OWL 1.1 Touch Paper: The OWL API. In *OWLED*, Vol. 258. Citeseer, 6–7.

Daniel Jackson. 1999. A comparison of object modelling notations: Alloy, uml and z. Unpublished Manuscript.

Daniel Jackson. 2006. *Software Abstractions: Logic, Language, and Analysis*. MIT Press.

Cliff B. Jones. 1986. *Systematic Software Development Using VDM*, Vol. 66. Prentice Hall.

Holger Knublauch. 2006. *Protege-OWL API Programmers Guide*.

Robert P. Kurshan. 1997. Formal verification in a commercial setting. In *Proceedings of the 34th Annual Design Automation Conference*. ACM, 258–262.

Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. 2006. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'06)*. ACM, New York, NY, 602–616. DOI:http://dx.doi.org/10.1145/1176617.1176632

Leslie Lamport. 1994. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 16, 3, 872–923.

Averill M. Law. 2009. How to build valid and credible simulation models. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE, 24–33.

Nancy A. Lynch and Mark R. Tuttle. 1987. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*. ACM, 137–151.

Leon McGinnis and Volkan Ustun. 2009. A simple example of SysML-driven simulation. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*. IEEE, 1703–1710.

Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)* 37, 4, 316–344.

John A. Miller, Gregory T. Baramidze, Amit P. Sheth, and Paul A. Fishwick. 2004. Investigating ontologies for simulation modeling. In *Proceedings of the 37th Annual Simulation Symposium*. IEEE, 55–63.

Robin Milner. 1982. *A Calculus of Communicating Systems*. Springer-Verlag, New York, NY.

Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2009. Owl 2 web ontology language: Profiles. *W3C Recommendation* 27, 61.

Michael J. North, Nicholson T. Collier, and Jerry R. Vos. 2006. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 16, 1, 1–25.

Dale K. Pace. 2000. Ideas about simulation conceptual model development. *Johns Hopkins APL Technical Digest* 21, 3, 327–336.

Ernest H. Page, Bradford S. Canova, and John A. Tufarolo. 1997. A case study of verification, validation, and accreditation for advanced distributed simulation. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 7, 3, 393–424.

Marco Antoniotti, Frank Park, Alberto Policriti, Nadia Ugel, and Bud Mishra. 2002. Foundations of a query and simulation system for the modeling of biochemical and biological processes. In *Pacific Symposium on Biocomputing 2003: Kauai, Hawaii, 3–7 January 2003*. World Scientific, 116.

Stewart Robinson. 2004. *Simulation: The Practice of Model Development and Use*. Wiley.

Stewart Robinson. 2006. Conceptual modeling for simulation: Issues and research requirements. In *Proceedings of the 38th Conference on Winter Simulation*. 792–800.

Robert G. Sargent. 2005. Verification and validation of simulation models. In *Proceedings of the 37th Conference on Winter Simulation*. 130–143.

Michal Sintek. 2003. *Ontoviz Tab: Visualizing Protégé Ontologies*.

Michael Spiegel, Paul F. Reynolds Jr., and David C. Brogan. 2005. A case study of model context for simulation composability and reusability. In *Proceedings of the Winter Simulation Conference*. IEEE, 8 pp.

J. Michael Spivey. 1988. *Understanding Z: A Specification Language and Its Formal Semantics*. Vol. 3. Cambridge University Press.

Maurice H. Ter Beek, Antonio Bucchiarone, and Stefania Gnesi. 2007. Formal methods for service composition. *Annals of Mathematics, Computing and Teleinformatics* 1, 5, 1–10.

Chris Tofts and Graham Birtwistle. 1998. A denotational semantics for a process-based simulation language. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8, 3, 281–305.

Andreas Tolk, Saikou Diallo, and Chuck Turnitsa. 2006. Merging protocols, grammar, representation, and ontological approaches in support of C-BML. In *Proceedings of the 2006 Fall Simulation Interoperability Workshop*.

Andreas Tolk, Saikou Diallo, and Chuck Turnitsa. 2007. A system view of C-BML. In *Proceedings of the 2007 Fall Simulation Interoperability Workshop*.

Andreas Tolk, Saikou Y. Diallo, Jose J. Padilla, and Heber Herencia-Zapana. 2013. Reference modelling in support of M&S foundations and applications. *Journal of Simulation* 7, 2, 69–82.

Wil M. P. Van der Aalst. 2005. Pi calculus versus Petri nets: Let us eat humble pie rather than further inflate the Pi hype. *BPTrends* 3, 5, 1–11.

Arie Van Deursen, Paul Klint, and Joost Visser. 2000. Domain-specific languages: An annotated bibliography. *ACM Sigplan Notices* 35, 6, 26–36.

Eelco Visser. 2008. WebDSL: A case study in domain-specific language engineering. *Generative and Transformational Techniques in Software Engineering II*, 291–373.

Jim Woodcock, Peter Gorm Larsen, Juan Bicarregui, and John Fitzgerald. 2009. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)* 41, 4, 19.

Received March 2013; revised July 2013; accepted November 2013