

## CODE ANALYSIS AND THE CS-XML

Kara A. Olson and C. Michael Overstreet  
Department of Computer Science  
Old Dominion University  
Norfolk, VA 23529-0162  
{kara, cmo}@cs.odu.edu

E. Joseph Derrick  
Department of Information Technology  
Radford University  
Radford, VA 24142-6933  
ejderrick@radford.edu

**ABSTRACT.** The automated analysis of model specifications is an area that historically receives little attention in the simulation research community but which can offer significant benefits. Since a common objective in simulation is enhanced understanding of a system, analysis of a model specification can provide insights not otherwise available. In addition, it can result in both time and cost savings to model development efforts. The Condition Specification (CS) [5] represents one of the model specification forms that is amenable to useful and informative analysis.

Current Web-based technologies such as XML offer exciting new approaches to extend our knowledge in this and other areas of simulation research. This paper discusses the motivations for and the creation of an XML Schema for the Condition Specification; a translator for the CS grammar into an XML-based Condition Specification (CS-XML); and a translator for the CS-XML into a fully-executable C/C++ program. It proposes immediate future work using CodeSurfer [1], a software static analysis tool, for model analysis. In conclusion, it is argued that the CS-XML can provide an essential foundation for Web Services that support the analysis of discrete-event simulation models.

### INTRODUCTION AND MOTIVATION

Model analysis can be beneficial in many ways. Such analysis can support development and verification of a model, aid in debugging, or help a modeler gain additional insights into the model being constructed. Observing and analyzing the behaviors produced by the simulation are the main techniques for improving understanding of a system being simulated. However, static analysis of the model specification itself can often reveal characteristics of a model not readily apparent from observing merely its run-time behavior. Furthermore, automated model diagnosis supports model verification and validation in the early stages of the model development process, thereby leading to savings in

project development time and costs, and yielding improvements to overall process quality [2].

The Condition Specification (CS) is a way of organizing primitives by which time and state relationships can be formalized [5] and is discussed in Section 2. Condition Specifications and Simulation Graphs [10] are among the few specification formalisms that have demonstrated promise and amenability to automated diagnostic techniques.

Extensible Markup Language (XML) has become a standard for representing data and information in a way that is easy and convenient for storage, retrieval, sharing, and processing in a distributed environment and among Web-based component applications. It is “playing an increasingly important role in the exchange of a wide variety of data on the Web

and elsewhere” [11]. Consequently, the authors decided to modernize the CS specification using XML.

The first step that was addressed was updating the Condition Specification to have a complete, modern grammar <sup>1</sup>. Next, an XML Schema was created <sup>1</sup>. Using this Schema, a translator was written that translates a given CS into an XML-based CS; this result is called a CS-XML, discussed in Section 3. With a CS-XML, we derive several important benefits:

- Semantic power of the XML representation due to its “extensible” nature,
- Ease and adaptability of use as a markup language document over other formats such as binary, fixed-length, or even delimited text data [8],
- Portability and supportability of its text-based format between diverse systems and platforms promoting the transfer of model specification data, and, ultimately,
- Wider availability of model diagnosis and analysis techniques to the simulation community.

From this foundational representation of the CS-XML, the authors have written a second translator to convert the CS-XML into a fully functional C/C++ program for additional analyses; translation into a conventional programming language provides access to additional existing analysis tools. Potential analyses are discussed in Section 4.

Standard processing via Web Services (e.g., providing various techniques for model diagnosis) is now possible; these plans are discussed in Section 5. Conclusions are discussed in Section 6.

#### MODEL DIAGNOSIS AND THE CONDITION SPECIFICATION

The Condition Specification was created to facilitate automated transformation among the classical world views of event scheduling, activity scanning, and process interaction. Serendipitously, supporting these transformations requires a representation that also enables several forms of useful diagnostic and informative analysis. The diagnostic capabilities of the CS are detailed in [6, 7]; an overview of its structure and possible analyses based on it are described herein.

In a CS, a model consists of a set of Objects; the state of each Object is captured in a set of Object Attributes. Similar to Finite State Machines and Zeigler’s DEVS formalism [12], model execution consists of a sequence of changes to Object Attributes. While a complete CS has several components, only

the Transition Specification is of immediate interest for the analysis discussed here. A Transition Specification describes both what triggers Attribute changes and how new values for them are computed. The triggers are called Conditions and the changes are called Actions. Table 1 illustrates, in conceptual form, a Transition Specification for a CS.

Condition	Actions
Condition 1	Action Sequence 1
Condition 2	Action Sequence 2
...	...
Condition $n$	Action Sequence $n$

TABLE 1. Structure of Transition Specification

The Conditions are boolean expressions in Object Attributes and are of three basic types. Those that only depend on the value of simulation time (enabling actions to be scheduled to occur at a particular future time) are called Time-based, or Alarms. Those that depend on Object Attributes not including simulation time are called State-based. Those that depend on both simulation time and other Object Attributes are called Mixed. For our purposes, Alarms are considered booleans that are only true at the instant that simulation time matches their scheduled time.

At (exactly) the beginning of a simulation, the special boolean condition Initialization, which must be included in a Transition Specification, is true. Hence, the Action associated with Initialization occurs only once, at start-up. It may schedule one or more Alarms for future times or it may change the values of several Object Attributes so that some Condition that was not previously true becomes true. The simulation proceeds accordingly with Actions causing some Conditions to become true, either in the same instant as the Action occurrence or at a future value of simulation time using Alarms.

With this structure, a Condition Specification can be analyzed to provide data and information which will potentially lead modelers and users of models to a better understanding of the system under consideration. The results of these analyses include items such as (1) explicit identification of causal relationships among Actions, (2) possible sequences of actions, (3) detection of certain types of errors in a specification, (4) assistance in creating efficient model implementations, and (5) creation of helpful

<sup>1</sup>Details are available from the authors.

model documentation. The CS has extensive analytic and diagnostic capabilities that offer significant benefits to modelers in the areas of *analytical* (existence of certain properties), *comparative* (differences among model representations), and *informative* (extraction or derivation of characteristics) diagnostic assistance. A variety of directed graph and matrix structures are derivable which effectively include the embedded relationships among object attributes and from which cause-effect relationships among these attributes can be readily determined. Analysis of these structures reveal information regarding attributes (utilization, classification, initialization, completeness, and consistency), the strength of relationships between equivalent conditions and the associated actions (called “cohesion”), and model components (connectedness, accessibility). This information can be extremely useful to modelers during the model development phases (e.g., measuring model complexity, simplifying model representations). Studies have been completed on the direct execution of CS forms for both sequential and parallel execution [7]; continuing research focuses on maximizing the utility of CS diagnostic capabilities (e.g., simplification techniques to recognize and eliminate redundancies [4]).

An example of analyses supported in a CS is the identification of both possible *causes* and *consequences* of each Condition-Action Sequence pair. This is an informative form of analysis with several potential benefits; for example, a modeler might detect either unexpected or missing causes or consequences for a Condition-Action Sequence pair, indicating a modeling error; in cases where no error has occurred, it might provide additional insight into model characteristics not easily observed during model execution. It also has obvious use as model documentation.

#### AN EXAMPLE OF CS-XML

The authors have built a Condition Specification parser that produces an XML representation, dubbed a CS-XML, as output. The feasibility of building static code analysis tools [4, 3] has been demonstrated locally, using standard parsing, data- and control-flow techniques in said tools. However, the authors have come to realize that in order to extend this work, they must build on existing tools which use standard representations. XML is such a representation and is well-supported by several existing tool sets. (Some of these tools include an abundance of editors for creating XML documents, tools based on the mature DOM (Document Object Model) and SAX (Simple API for

XML) technologies for reading and parsing XML documents, as well as a host of other associated developer APIs and tools for XML processing, validation, XSL (XML Stylesheet Language) transformation, and Web Services that are part of the standard Sun Java JDK/SDK distributions.)

Figures 1 and 2 provide an example of a CS-XML. Since the complete CS-XML for even a simple model is quite large, only a pair of snippets are provided: part of the Transition Specification for a model, and the corresponding CS-XML generated by the translator from this part of the Specification. In Figure 1, the first line is a comment. The second contains the boolean Condition (in parentheses after the key word *when*). The remaining three lines are the Action Sequence that is to occur whenever the Condition holds; it includes a *set alarm* for the Alarm *arr\_facility*. Figure 2 presents the corresponding CS-XML for this part of the Transition Specification.

As can be seen from Figure 2, the CS-XML contains sufficient details from a Transition Specification to support both traditional static code analysis and other types of analysis technologies which could be incorporated into Web Services.

```

// travel to facility
when ((for some i: facility[i].failed == true) &&
      (repairman.status == available)) {
  j := closest_failed_fac(facility, repairman.location);
  set alarm(repairman.arr_facility, j,
            traveltime(repairman.location, j));
  repairman.status := traveling;
}

```

FIGURE 1. Part of a Transition Specification

#### POTENTIAL ANALYSES

Both static and dynamic analyses can provide insight to both modelers and to model users. Static analysis of the model specification can reveal relationships that may not be readily apparent by simply observing model output. Dynamic analysis can reveal specifics of which events caused which events, which cannot be determined prior to run-time. Each type of analysis has unique benefits; consequently, we intend to explore a bit of both.

Figure 3 is a dependency graph of a machine repair model. The main purpose of this graph, derived from source code, is to show which events can cause which events. A solid line indicates that event *a* can cause event *b* to occur at the same instance in time, whereas a dashed line indicates that event *a* will cause event *b* at a future instance in time.

```

<transition>
  <boolean_expression>
    <for_some>
      <index>i</index>
      <comparison>
        <attribute>facility[i].failed</attribute>
        <is_equal_to />
        <value>>true</value>
      </comparison>
    </for_some>
    <and />
    <comparison>
      <attribute>repairman.status</attribute>
      <is_equal_to />
      <value>available</value>
    </comparison>
  </boolean_expression>
  <assignment>
    <attribute>j</attribute>
    <equals />
    <result>
      <function>
        <name>closest_failed_fac</name>
        <argument>facility</argument>
        <argument>repairman.location</argument>
      </function>
    </result>
  </assignment>
  <set_alarm>
    <argument>repairman.arr_facility</argument>
    <argument>j</argument>
    <function>
      <name>traveltime</name>
      <argument>repairman.location</argument>
      <argument>j</argument>
    </function>
  </set_alarm>
  <assignment>
    <attribute>repairman.status</attribute>
    <equals />
    <value>traveling</value>
  </assignment>
</transition>

```

FIGURE 2. Corresponding part of CS-XML

In Figure 3, the simulation is started with the event *initialization*. *Initialization* can cause *travel\_to\_failure*, *travel\_to\_idle*, or *termination* to occur at start-up, and will cause *failure* to occur at a future time; and similarly for the other nodes in the graph. During any given run, determining if one event causes another is undecidable; hence the need

for static analysis. However, consider for example event *travel\_to\_idle*: while static analysis is useful for determining what events could cause *travel\_to\_idle*, in general only dynamic analysis can aid in determining which event actually caused *travel\_to\_idle* (*failure*, *travel\_to\_failure*, *end\_repair*, *initialization*, *arrive\_idle*, or *travel\_to\_idle*). Thus, model understanding could be enhanced using both dynamic and static analysis techniques.

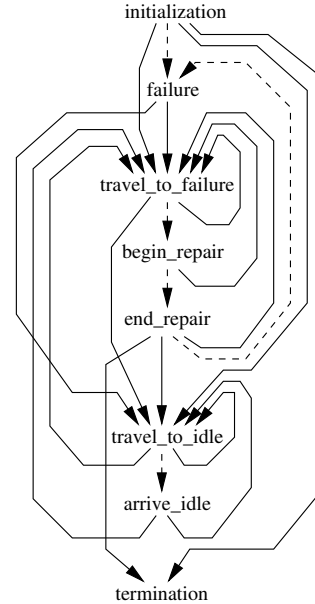


FIGURE 3. Machine Repair dependence graph

#### FUTURE WORK AND INTERESTS

As mentioned in Section 1, once a CS-XML has been created, a translator can be used to produce a fully functional C/C++ program. While this in and of itself is extremely useful, two of the authors' primary interest is in code analysis. Hence, we are planning immediate work of applying CodeSurfer for model analysis, as we are confident in its potential for useful analyses.

We also plan to use XML parser tools for Simple API for XML (SAX) and Document Object Model (DOM) [9] to process the CS-XML in order to produce various and appropriate graph-based results and to accomplish the analytical, comparative, and informative diagnostic tests discussed in Section 2. The intent is to create and distribute these tests as Web Services using a Service-Oriented Architecture (SOA) approach, making the analyses readily accessible to both developers and users. The first

service now under development is a validation service to validate a given CS-XML against its XML Schema. Furthermore, Extensible Stylesheet Language Transformations (XSLT) will be investigated for transforming the CS-XML and displaying these results.

#### SUMMARY AND CONCLUSIONS

This paper has established the CS-XML and discussed its background and development. The authors have provided a sample of the CS to CS-XML translation and discussed a CS-XML to C/C++ translator, as well as multiple directions of future work, including the creation of Web Services for discrete-event simulation model diagnosis using the CS. The CS-XML provides access in to the many diagnostic capabilities of the CS - until now, a resource that has been largely untapped - and opens the doors of many potential research opportunities into new and advanced analysis techniques.

#### REFERENCES

- [1] P. ANDERSON, T. W. REPS, T. TEITELBAUM, AND M. ZARNIS, *Tool support for fine-grained software inspection*, IEEE Software, 20 (2003), pp. 42–50.
- [2] O. BALCI AND R. E. NANCE, *Simulation model development environments: A research prototype*, J. Opl Res. Soc., 38 (1987), pp. 753–763.
- [3] R. D. CHERINKA, C. M. OVERSTREET, AND J. A. RICCI, *Maintaining a COTS integrated solution - are traditional static analysis techniques sufficient for this new programming methodology?*, in 14th IEEE International Conference on Software Maintenance (ICSM '98), 1998, pp. 160–169.
- [4] R. E. NANCE, C. M. OVERSTREET, AND E. H. PAGE, *Redundancy in model specifications for discrete event simulation*, ACM Trans. Model. Comput. Simul., 9 (1999), pp. 254–281.
- [5] C. M. OVERSTREET AND R. E. NANCE, *A specification language to assist in analysis of discrete event simulation models*, Commun. ACM, 28 (1985), pp. 190–201.
- [6] C. M. OVERSTREET, E. H. PAGE, AND R. E. NANCE, *Model diagnosis using the Condition Specification: From conceptualization to implementation*, in Proceedings of the 1994 Winter Simulation Conference, December 1994, pp. 566–573.
- [7] E. H. PAGE AND R. E. NANCE, *Parallel discrete event simulation: A modeling methodology perspective*, in Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94), 1994, pp. 88–93.
- [8] K. QIAN, R. ALLEN, M. GAN, AND B. BROWN, *Java Web Development Illuminated*, Jones and Bartlett, Boston, MA, 2007.
- [9] E. T. RAY, *Learning XML*, O'Reilly & Associates, Inc., second ed., September 2003.
- [10] L. SCHRUBEN, *Simulation modeling with event graphs*, Comm. ACM, 26 (1983), pp. 957–963.
- [11] World Wide Web Consortium (W3C). *W3C Architecture Domain: Extensible Markup Language (XML)*, <http://www.w3.org/XML/>.
- [12] B. P. ZEIGLER, *Object-Oriented Simulation with Hierarchical, Modular Models*, Academic Press, Boston, MA, 1990.