Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

CODE ANALYSIS AND CS–XML

Kara A. Olson
C. Michael Overstreet

Department of Computer Science
Old Dominion University
Norfolk, VA 23529–0162, U.S.A.

E. Joseph Derrick

Department of Information Technology
Radford University
Radford, VA 24242–6933, U.S.A.

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. A common objective in simulation is enhanced understanding of a system; model specification analysis can provide insights not otherwise available as well as time and cost savings in model development. The Condition Specification (CS) (Overstreet and Nance 1985) represents a model specification form that is amenable to analysis. This paper discusses the motivations for and the creation of CS-XML; a translator for CSes into XML-based Condition Specifications; and a translator for CS-XML into fully-executable C/C++ code. It presents initial results from analysis efforts using CodeSurfer (Anderson et al. 2003), a software static analysis tool, and discusses future work. 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.

1 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 both during and after completion. Observing and analyzing the behaviors produced by the simulation are currently 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 as well as yielding improvements to overall process quality (Balci and Nance 1987).

The Condition Specification (CS) is a way of organizing primitives by which time and state relationships can be formalized (Overstreet and Nance 1985) and is discussed in Section 2. Condition Specifications and Simulation Graphs (Schruben 1983) are among the few specification formalisms that have demonstrated promise and amenability to automated diagnosis of discrete-event systems.

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” (World Wide Web Consortium 2006). Important XML-based related research includes:

- The OWL Web Ontology Language (Lacy and Gerber 2004), an evolving open standard initiative of the World Wide Web Consortium (W3C) to improve the representation of semantic information on the Web and enable better processing by supporting software applications;
- The Extensible Modeling Simulation Framework (XMSF), “a set of Web-based technologies and services, applied within an extensible framework” that bridges the gap between the new commercial technologies and defense systems and enables the creation of cutting-edge distributed modeling and simulation applications (Tolk 2004);
- Base Object Models (BOMs) (Gustavson and Chase 2004), an XML standard to enable building simulations and environments and promote interoperability and reusability; and
- The Simulation Reference Markup Language (SRML) (Reichenthal 2004), an XML technology for describing model data and behavior and for enabling execution of the model using a simulation engine.

Given the focused interest and significant existing body of work in model diagnosis and analysis, the authors decided to modernize the Condition Specification using XML to explore additional ideas in Web-based model analysis. This work should support the growing body of work in model understanding.

The first step that was addressed was updating the Condition Specification to have a complete, modern grammar. Next, an XML Schema was created (details are available from the authors). Using this Schema, a translator was written that translates a given CS into an XML-based CS; this result yields CS-XML, discussed in Section 3. With 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 (Qian et al. 2007),
- 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.

The authors have written a second translator to convert CS-XML into fully functional C/C++ code for additional analyses; translation into a conventional programming language provides access to additional existing analysis tools without incurring the overhead of converting said tools to process XML.

Standard processing via Web Services (e.g., providing various techniques for model diagnosis) is now possible; these future plans are discussed in Section 4. Conclusions are discussed in Section 5.

2 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 Overstreet, Page, and Nance (1994) and Page and Nance (1994); an overview of its structure and possible analyses based on it are described herein. Our goal in this on-going research is to assess the feasibility and benefits of model analysis. We have chosen the CS as a basis for this work because of its demonstrated support of model analysis. If the results of this effort are encouraging, we expect to explore analyses based on additional model specification forms in the future.

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 (Zeigler 1990), model execution consists of a sequence of changes to Object Attributes over simulation-driven time advances. While a complete CS has several components, only the Transition Specification is of immediate interest for the analysis discussed herein. 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.

Table 1: Structure of transition specification.

Conditions	**Actions**
Condition 1	Action Sequence 1
Condition 2	Action Sequence 2
...	...
Condition n	Action Sequence n

The Conditions are boolean expressions of 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 startup. 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 explicit identification of causal relationships among Actions, possible sequences of actions, detection of certain types of errors in a specification, assistance in creating efficient model implementations, and creation of helpful 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 model events 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 (Page and Nance 1994); continuing research focuses on maximizing the utility of CS diagnostic capabilities – for example, simplification techniques to recognize and eliminate redundancies (Nance, Overstreet, and Page 1999).

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.

3 AN EXAMPLE OF CS-XML

The authors have built a Condition Specification parser that produces an XML representation, dubbed CS-XML, as output. The feasibility of building static code analysis tools (Nance, Overstreet, and Page 1999, Cherinka, Overstreet, and Ricci 1998) has been demonstrated locally, using standard parsing, data- and control-flow analysis 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 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 work *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.

```
// 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.

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.

4 INITIAL ANALYSES

Using CodeSurfer and working with backward slices (Weiser 1984), the authors were able to find several unnoticed errors in Nance, Overstreet, and Page (1999), involving a simulation of a harbor model with a simple dependency graph as most models go. The main purpose of this graph, derived from source code, is to show which events can cause which events.

In the harbor model from Nance, Overstreet, and Page (1999), ships arrive at a harbor and wait for both a berth and a tug boat to become available. A ship is then escorted to a berth, unloaded, and escorted back to sea. This model is used to study tug boat utilization and ship in-harbor time. This version of the harbor model appears in Buxton and Laski (1963) and Schriber (1974).

In Figures 3 and 4, a solid line indicates that event *a* can cause event *b* to occur at the same instance in time; a dashed line indicates that event *a* can cause event *b* at a future instance in time. For example, in Figure 3, a *move_tug_to_ocean* event can cause a *deberth* event to occur; also, a *deberth* event can cause a *move_tug_to_ocean* event to occur. A dotted line in Figure 4 indicates a correction or addition.

These figures also illustrate a prime problem with model descriptions whether in textual or graphical notations: even in simple models, the descriptions are often difficult to

```
<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.

fully comprehend (even if automatically derived from source code). The type of tools we hope to develop may help with the problem of having "too much information" by allowing the interactive exploration of a model so that only relevant information is presented. A central goal in this research is to determine the feasibility of presenting modelers with only relevant information, based on current interests, as they examine a model. Model documentation as well as other static representations of models can easily obscure useful information in the volume of what is presented. Additionally, as the problem of interest changes, what information is considered relevant also changes.

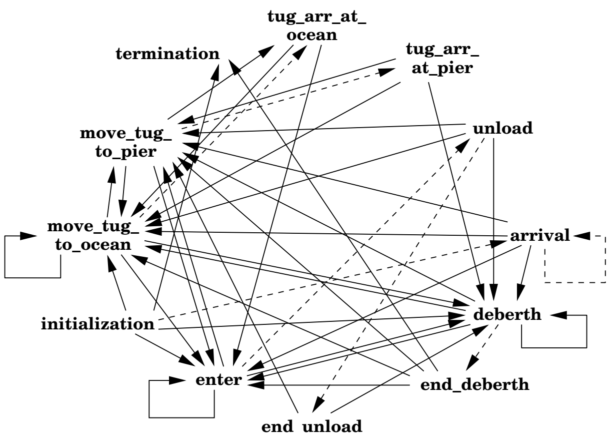

Figure 3: The original dependence graph.

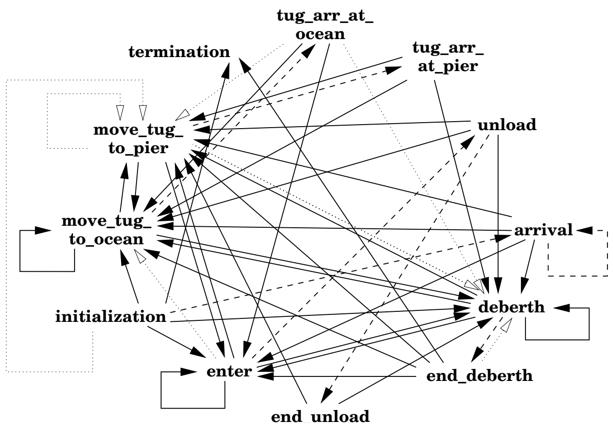

Figure 4: The updated dependence graph.

5 FUTURE WORK AND INTERESTS

As mentioned in Section 1, once an XML-based CS 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 continued work of applying CodeSurfer for model analysis, as we are confident in its potential for additional useful analyses.

We also plan to use XML parser tools for Simple API for XML (SAX) and Document Object Model (DOM) (Ray 2003) to process 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 utilizing a Service-Oriented Architecture (SOA) approach, making the analyses readily accessible to both developers and simulationists. The first service now under development is a validation service to validate a given XML-based CS against its XML Schema. Furthermore, Extensible Stylesheet Language Transformations (XSLT) will be investigated for transforming the CS-XML and displaying these results.

6 SUMMARY AND CONCLUSIONS

This paper has established 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. 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

Anderson, P., T. W. Reps, T. Teitelbaum, and M. Zarnis. 2003, July/August. Tool support for fine-grained software inspection. *IEEE Software* 20 (4): 42–50.

Balci, O., and R. E. Nance. 1987, August. Simulation model development environments: A research prototype. *J. Opl Res. Soc.* 38 (8): 753–763.

Buxton, J. N., and J. G. Laski. 1963. Control and simulation language. *Comput. J* 5 (3): 194–199.

Cherinka, R. D., C. M. Overstreet, and J. A. Ricci. 1998. 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)*, 160–169.

Gustavson, P., and T. Chase. 2004, December. Using XML and BOMS to rapidly compose simulations and simulation environments. In *Proceedings of the 2004 Winter Simulation Conference*, 1467–1475.

Lacy, L., and W. Gerber. 2004, December. Potential modeling and simulation applications of the web ontology language - OWL. In *Proceedings of the 2004 Winter Simulation Conference*, 265–270.

Nance, R. E., C. M. Overstreet, and E. H. Page. 1999, July. Redundancy in model specifications for discrete event simulation. *ACM Trans. Model. Comput. Simul.* 9 (3): 254–281.

Overstreet, C. M., and R. E. Nance. 1985, February. A specification language to assist in analysis of discrete event simulation models. *Commun. ACM* 28 (2): 190–201.

Overstreet, C. M., E. H. Page, and R. E. Nance. 1994, December. Model diagnosis using the condition specification: From conceptualization to implementation. In *Proceedings of the 1994 Winter Simulation Conference*, 566–573.

Page, E. H., and R. E. Nance. 1994. Parallel discrete event simulation: A modeling methodology perspective. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, 88–93.

Qian, K., R. Allen, M. Gan, and B. Brown. 2007. *Java web development illuminated*. Boston, MA: Jones and Bartlett.

Ray, E. T. 2003, September. *Learning XML*. Second ed. O'Reilly & Associates, Inc.

Reichenthal, S. W. 2004, December. SRML case study: simple self-describing process modeling and simulation. In *Proceedings of the 2004 Winter Simulation Conference*, 1461–1466.

Schriber, T. J. 1974. *An introduction to simulation using GPSS/H*. New York, NY: John Wiley & Sons, Inc.

Schruben, L. 1983, November. Simulation modeling with event graphs. *Comm. ACM* 26 (11): 957–963.

Tolk, A. 2004, December. XML mediation services utilizing model based data management. In *Proceedings of the 2004 Winter Simulation Conference*, 1476–1484.

Weiser, M. 1984, July. Program slicing. *IEEE Trans. Softw. Eng.* SE-10 (4): 352–357.

World Wide Web Consortium 2006. W3C architecture domain: Extensible markup language (XML). <www.w3.org/XML>.

Zeigler, B. P. 1990. *Object-oriented simulation with hierarchical, modular models*. Boston, MA: Academic Press.

AUTHOR BIOGRAPHIES

KARA A. OLSON is pursuing her doctorate in Computer Science at Old Dominion University. She holds a Master of Science, Computer Science, Bachelor of Computer Science and Bachelor of Science, Mathematics from Old Dominion. Her research interests include analysis of simulation models in order to facilitate user understanding. She is a member of ACM, IEEE, IEEE Computer Society, and SIAM. She can be reached by e-mail at <kara@cs.odu.edu>.

C. MICHAEL OVERSTREET is an Associate Professor of Computer Science at Old Dominion University and is currently Interim Associate Dean of the College of Sciences. A member of ACM and IEEE/CS, he is a former chair of SIGSIM, and has authored or co-authored over 80 refereed journal and conference articles. He received a B.S. from the University of Tennessee, an M.S. from Idaho State University and an M.S. and Ph.D. from Virginia Tech. He has held visiting appointments at the Kyushu Institute of Technology in Iizuka, Japan, and at the Fachhochschule für Technik und Wirtschaft in Berlin, Germany. His current research interests include analysis of simulation models to

enhance model understanding and static code analysis. Dr. Overstreet's home page is <www.cs.odu.edu/~cmo>. He can be reached by e-mail at <cmo@cs.odu.edu>.

E. JOSEPH DERRICK is an Assistant Professor in the College of Science and Technology at Radford University. He received Ph.D. and M.S. degrees in Computer Science from Virginia Tech and a B.S. in Electrical Engineering from the United States Naval Academy. His research interests include software engineering, network security, Web-based simulation and emerging Web technologies. He is a member of the IEEE Computer Society and the ACM. Dr. Derrick's home page is <www.radford.edu/ejderrick>. He can be reached by e-mail at <ejderrick@radford.edu>.