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) [7] is a model specification form that is amenable to analysis; an XML-based representation of such is said to be in CS-XML. This paper discusses the motivations for CS-XML and presents some results from analysis efforts using CodeSurfer [1], a software static analysis tool. CS-XML also provides an essential foundation for Web Services that support the analysis of discrete-event simulation models.

INTRODUCTION

Simulations are often used to enhance model understanding of simulated systems by both model builders and model users. Observing animations during model executions can help users understand a system’s behavior. Likewise, model builders can often detect modeling errors and sometimes gain better understanding of their models.

We have a long interest in model analysis and have noticed that analyses can often reveal important aspects of systems that are not readily observable in model-driven animation or even in examining data produced by a simulation during execution.

Our interest is in providing both model builders and model users additional insight into their models.

CS-XML

The Condition Specification (CS) is a way of organizing primitives by which time and state relationships can be formalized [7]. Condition Specifications and Simulation Graphs [12] are among the few specification formalisms that have demonstrated promise and amenability to automated diagnosis of discrete-event systems.

In a CS, a model consists of a set of Objects (such as a repairman); the state of each Object is captured by a set of Object Attributes (such as his status or location). Similar to Finite State Machines and Zeigler’s DEVS formalism [15], model execution consists of a sequence of changes to Object Attributes over simulation-driven time advances.

Figure 1 contains a snippet of a Condition Specification. In it, the first line is a comment. The second contains a boolean condition (in parentheses after the keyword when). The remaining three lines are the Action Sequence that is to occur whenever the condition holds; it includes, among other things, a set alarm for the Alarm arr_facility. Analyses supported by the CS include the identification of possible causes and possible consequences of each Condition-Action Sequence pair.
// 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 Condition Specification

The feasibility of building static code analysis tools has been demonstrated locally [5, 3] using standard parsing, data- and control-flow analysis techniques. However, the authors have come to realize that in order to extend this work, they must build on existing tools which use standard representations. Extensible Markup Language (XML) is such a representation and is well-supported by several existing tool sets.

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” [14]. Consequently, the authors decided to modernize the Condition Specification using XML. We are developing a Condition Specification parser that produces an XML-based representation of a Condition Specification; we say this representation is in CS-XML.

With CS-XML, we derive several important benefits:

- Semantic power of 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 [9];
- 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 translator to convert an XML-based CS into a fully functional C/C++ program for additional analyses; translation into a conventional programming language provides access to additional existing analysis tools.

Figure 2 provides the CS-XML corresponding to the snippet given in Figure 1. The complete CS-XML for even a simple model is quite large. As can be seen from Figure 2, CS-XML contains sufficient detail to support both traditional static code analysis as well as other analysis techniques which could be incorporated into Web Services.

Figure 2: Corresponding part in CS-XML
Analyses

CodeSurfer is a software static analysis tool based on more than ten years of research on system dependence graphs [1]. Using CodeSurfer and working with backward slices [13], the authors are able to generate action cluster interaction graphs [6]. The main purpose of this type of graph, derived from source code, is to show which events can cause which events.

In these graphs, 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 $a$ will cause event $b$ at a future instance in time.

$\begin{align*}
  a & \rightarrow b \quad a \text{ can cause } b \text{ now} \\
  a & \rightarrow \leftarrow b \quad a \text{ will cause } b \text{ in the future}
\end{align*}$

In the traveling repairman model from Palm [8] and from Cox and Smith [4], a repairman tends to a number of machines which fail over time and need repair. This model can be used to study how many machines or repairmen are needed, effects of machine modifications, and production rates.

Consider some of what can be learned from Figure 3. The simulation is started with the event initialization. An initialization event can cause a travel_to_facility, travel_to_idle, or termination event to occur at start-up, and will cause a failure event to occur at future time. A failure event can cause a travel_to_facility or travel_to_idle event; and similarly for the other nodes in the graph. For any given model, determining if one event causes another can be undecidable. However, consider for example event travel_to_idle: static analysis is useful for determining what events could cause travel_to_idle; dynamic analysis can aid in determining which event actually caused travel_to_idle (failure, travel_to_facility, begin_repair, end_repair, initialization, arrive_at_idle, or travel_to_idle). Accordingly, model understanding can be enhanced using both dynamic and static analysis techniques.

In the harbor model from Buxton and Laski [2] and from Schriber [11], 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 can be used to study tug boat utilization and ship in-harbor time.

Considering Figure 4, initialization can cause enter, move_tug_to_ocean, deberth, or terminate to happen now, and will cause arrival to happen in the future. Similarly, unload can cause move_tug_to_pier, move_tug_to_ocean, or deberth now, and will cause end_unload in the future.

These figures – one having only eight events, the other only 12 – also illustrate a prime problem with model descriptions, whether in textual or graphical notations: even in simple models, descriptions are often difficult to fully comprehend (even when 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 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 their current interests, as they examine a model. Additionally, as the problem of interest changes, what information is considered relevant also changes.

Continuing Work and Interests

As mentioned, a translator from a CS to an XML-based CS can be written; however, 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.
These techniques are used in the Software Engineering community and are largely based on work done on computer language compilers for support of code optimization. Many of the things an optimizing compiler must discover about code in order to optimize it can also be informative to the creators of that code. We believe this concept also applies to modelers and model users.

We also plan to use XML parser tools for Simple API for XML (SAX) and Document Object Model (DOM) [10] to process CS-XML and produce various and appropriate graph-based results as well as analytical, comparative, and informative diagnostic tests. Our 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 simulationists.

**Summary**

People regularly build models to learn things and better understand systems of interest. Analyses of model specifications provide largely unexplored opportunities for assisting people in their understanding of systems for many purposes: debugging during development, exploration of the impact of possible changes to a system, and education of new users of a system.

This paper has discussed CS-XML and some of the analyses that can be done with it. CS-XML provides access to some of the diagnostic capabilities of the Condition Specification and opens many potential research areas into new and advanced analysis techniques.

**Acknowledgment**

Thank you to Virginia Space Grant Consortium for three years of fellowship.

**References**


