Automated parametric execution and documentation for large-scale simulations

R. L. Kelsey, K. R. Bisset, and R. B. Webster

Automated parametric execution and documentation for large-scale simulations

R. L. Kelsey\textsuperscript{a} and K. R. Bisset\textsuperscript{b} and R. B. Webster\textsuperscript{c}
Los Alamos National Laboratory
\textsuperscript{a}X-8 MS-F645, \textsuperscript{b}D-5 MS-F602, \textsuperscript{c}X-3 MS-D413
Los Alamos, NM 87545

ABSTRACT

A language has been created to facilitate the automatic execution of simulations for purposes of enabling parametric study and test and evaluation. Its function is similar in nature to a job-control language, but more capability is provided in that the language extends the notion of literate programming to job control. Interwoven markup tags self document and define the job control process. The language works in tandem with another language used to describe physical systems. Both languages are implemented in the Extensible Markup Language (XML).

A user describes a physical system for simulation and then creates a set of instructions for automatic execution of the simulation. Support routines merge the instructions with the physical-system description, execute the simulation the specified number of times, gather the output data, and document the process and output for the user.

The language enables the guided exploration of a parameter space and can be used for simulations that must determine optimal solutions to particular problems. It is generalized enough that it can be used with any simulation input files that are described using XML. XML is shown to be useful as a description language, an interchange language, and a self-documented language.

Keywords: XML, self-documenting language, physical system simulation, automatic simulation execution

1. INTRODUCTION

The modeling and simulation of physical systems can be difficult. Physical systems, because of their breadth and detail, tend to be simulated with complex application codes and on large, parallel computing platforms. As these computing platforms become larger to help solve larger and more detailed problems, their use becomes more complex. Tools and processes are necessary to help manage this complexity.

In addition to managing complexity is the support of users' tasks. Some typical tasks include parameter testing, exploration and search of a parameter space, and application code to application code comparison. These tasks all share the need for executing multiple simulations. The parameter testing and parameter space search tasks both take a few parameters in the problem domain and vary their values over the course of the simulation. Application code to application code comparison may be useful for validation purposes. This research contributes to simplifying the creation, set up, and submission of physical system problems for large-scale simulation.

2. XML BACKGROUND

The Extensible Markup Language\textsuperscript{1} (XML) is a subset of the Standard Generalized Markup Language\textsuperscript{2} (SGML). Both XML and SGML are used for creating markup languages. They can also be used for creating object-based representations.\textsuperscript{3} These representations need not be full-blown ontologies. They can be small and specific to a single application. Also, implementing an object-based representation in XML for an application helps make the representation and the representation language more closely coupled with the knowledge and information being represented. The representation is also more portable.

Further author information: (Send correspondence to R.L.K)
R.L.K: E-mail: rob@lanl.gov
K.R.B: E-mail: kbisset@lanl.gov
R.B.W: E-mail: robw@lanl.gov
Like SGML, XML contains three parts: a specification, a document type definition (DTD), and a document instance. An application and/or representation is created by creating and using a DTD. A DTD is a grammar and contains three constructs. The constructs are element, attribute, and entity. An element construct can be thought of as an object. An attribute construct is a property or characteristic of an element. An entity construct is used like a macro to define something that can be expanded repeatedly throughout the DTD. Elements and their associated attributes are used to represent objects in the application of interest.

Once a DTD or representation is created, it can be used. Use occurs by creating a document instance of the representation. This is a populated representation that can be used within an application.

3. REPRESENTATIONS

3.1. Physical system representation

In this set of applications XML is used to create two different representations. The first representation is for describing a physical system for simulation. This representation includes all the objects necessary to create a description of a physical system such as a shock wave physics problem or the motion of a projectile. The representation includes objects for a computational mesh, simulation start up information, a material set, a geometric body set, and simulation output information. These are objects at the highest level of the representation and may contain sub-levels of objects as much as four levels deep. For example, a material set object contains any number of material objects. A material object is further described by an equation of state model object, a fracture model object, and a strength model object. Each object in the representation may have a number of attributes. For example, a material object includes attributes such as name, type, density, and specific internal energy.

A user can either create one of these descriptions from scratch or they can convert an existing application code input file. Typically, the majority of descriptions come from conversion. The XML-based representation serves as a data-interchange format. Input files of one application code format are converted to the XML-based representation and then to another application code format. This allows a single physical system problem description to be simulated on many different application codes.

3.2. Job control representation

The second representation is a representation and language for simulation job control. Using the language, a user can specify how many times to run an application code simulation and with what physical system description. This helps automate the execution process of simulations and consolidates some of the output products for comparison analysis.

The representation for simulation job control is simple and compact. The representation currently contains less than 12 objects. Figure 1 shows an abstract view of a portion of the representation. The objects are labeled with initial uppercase letters. Attributes of the objects are to the right of object labels and are labeled with lowercase letters. At the top of the taxonomy is an Execution Plan object. The Execution Plan object has two attributes: <input file name> and <input file source>. The <input file name> specifies the name of a physical system input file. The input file contains a complete physical system problem description. The <input file source> attribute specifies whether the input file is located in a file or a database.

The Execution Plan object contains or aggregates two other objects: the Problem Statement object and the Run List object. The Problem Statement object allows the user to specify a comment or statement about the problem. The Run List object contains one or more Run objects. Neither the Problem Statement nor the Run List object have any attributes.

The Run object has two attributes: <number> and <application code>. The <number> attribute specifies a unique identification number for a particular run or execution. The <application code> attribute specifies a particular simulation application code for this run or execution. The Run object contains zero or more Substitute objects.

The Substitute object is used to specify substitutions in the physical system input file. This allows the user to automatically create a different problem (than the physical system input originally specified) for each run. Each of these problems is based on the originally specified physical system input file. The Substitute object has two attributes: <type> and <name>. The <type> attribute specifies whether the construct to be substituted is an element or an attribute. The <name> attribute identifies the name of the element or attribute to be substituted. The Substitute object contains one <With object>. 
The With object specifies the replacement. The With object has two attributes: source and name. The source attribute specifies where the replacement comes from, which may be a file, a database, a function, or specified here. The name attribute identifies the name of the file, a query to the database, or the function specification depending on the value of the source attribute. If the value of the source attribute is here, then the With object can contain content which specifies the replacement.

4. SOME EXAMPLES

The following paragraphs present and discuss some examples of the language for simulation job control. The examples contain the correct syntax for use. They demonstrate some of the power and flexibility of the language and its use. Also discussed is the processing of the language and what happens during actual use.

Figure 2 shows an example instance of the simulation job control language. The physical system description being executed is called shockwave and comes from a file. The run list contains two different runs. The first run is identified as run number 1 and is a run of an application code called fluid-dynamics. Run number 1 has no substitutions, which means it is an as-is execution of the shockwave problem with the fluid-dynamics application code. The next run is identified as run number 2 and has a single substitution. Run number 2 is another execution of the fluid-dynamics application code but, the MATERIAL LIST (element) in the shockwave physical system description is being substituted with the contents of a file named material-file-2.

```xml
<EXECUTION_PLAN infile_name="shockwave" infile_source="file"/>
<PROBLEM_STATEMENT>This is a shockwave problem test.</PROBLEM_STATEMENT>
<RUN_LIST>
  <RUN number="1" application_code="fluid-dynamics"/>
  <RUN number="2" application_code="fluid-dynamics">
    <SUBSTITUTE type="element" name="MATERIAL_LIST">
      <WITH source="file" name="material-file-2"/>
    </SUBSTITUTE>
  </RUN>
</RUN_LIST>
<EXECUTION_PLAN>
```

Figure 2. An example instance of the simulation job control language.

Figure 3 shows another example of the simulation job control language. In this example the physical system description is called material-study and comes from a database (signified by db). The run list contains four different runs each using a different application code to simulate the physical system. The application codes for each respective
run are code-a, code-b, code-c, and code-d. None of the runs contain any substitutions. This example illustrates how different application codes can be compared using the same test, in this case, the physical system description called material-study.

Figure 3. An example of application code comparison.

Figure 4 shows a third example. In this example the physical system description is called test-123 and comes from a database. The run list contains only one run. The run contains five substitutions. In the first substitution the MESH object is being replaced with the contents of a file called mesh-file-32. The next three substitutions replace various MATERIAL objects identified by their name attribute. The material named al-1 is being replaced with a material named s-steel. The material named al-2 is being replaced with a material named composite-b and the material named cu is being replaced with a material named pb. All three of the replacement materials are coming from a database. The name attributes of the SUBSTITUTE and WITH objects are using an XPath type syntax. XPath is a language that is designed for accessing parts of an XML document. In this case it is being used for querying the physical system description and a database that contains material data described in an XML form. The final substitution is the OUTPUTS object and is being replaced with the contents of a file called output-defs.

Figure 4. An example with multiple substitutions.
The third example illustrates that even though an execution plan and all of its runs are limited to a single physical system description, there is nothing to prevent substituting most or all of the objects in that description. This means that a particular run need only be based on an originating physical system description. Substitution allows a new description to be created based on a known description. This also illustrates how a description can be created by drawing together separate and potentially disparate pieces. For example, a mesh can be borrowed from a colleague. Individual materials can be queried from a material database or come from other problem descriptions. Physical system descriptions can be put together much like one puts together a puzzle.

An instance of the simulation job control language is processed by a number of Java routines. Some of these routines will be discussed in more detail in the following section. The processing begins when the instance is parsed and the instructions interpreted. The (base) physical system problem description is retrieved from its source and a copy is made for each run. Substitutions, if any, for each run are made in the corresponding copy of the problem description. Each copy of the problem description is then converted to the input format of the application code specified for each run. Finally, the individual runs are submitted for execution. Figure 5 shows the migration of a single run from its base physical system description and substitutions to the physical system description of the desired problem (creation) to the input file format (conversion) of the application code of interest.

![Figure 5](image)

Figure 5. The migration of a single run as specified in the simulation job control language.

## 5. SUPPORT ROUTINES

### 5.1. Documentation

Support routines are available to generate several types of documentation. The documentation ranges from descriptive help-type messages to formal printable reports. Much of the documentation is leveraged from the self-documenting or self-describing nature of XML. A user can issue a command to display a brief narrative description of a physical system description and/or a job control instance. Figure 6 shows the brief documentation generated from the example instance of simulation job control language shown in Figure 2. More lengthy and detailed narratives can be generated for formal reports and presentations. It is also possible to include output products such as plots and illustrations in the reports. All of these documents are designed to be more human-readable than the original sources.

Any type of documentation that is generated from an XML-based instance or representation utilizes the Extensible Stylesheet Language (XSL) XSL Transformations (XSLT), and Xalan Java. XSL and XSLT are used to create a style sheet that represents the target document format and what content (from the XML instance) belongs in that format. Xalan Java is a tool which converts the XML instance according to the style sheet into the target document. Thus, there is a style sheet defined for each type of document.

Plots and illustrations can be created with scalable vector graphics (SVG). SVG is a language for describing two-dimensional vector graphics and is implemented in XML. The output products when represented with SVG can be rendered using SVG viewers and/or converted into other document formats such as portable document format...
Problem Statement

This is a shockwave problem test.

This execution plan contains 2 runs of the shockwave input deck.
Run number 1 is a run of the fluid-dynamics application code.
This run has no modifications.

Run number 2 is a run of the fluid-dynamics application code.
This run has 1 modification. The MATERIAL_LIST element is being substituted with the contents of the file material-file-2.

Figure 6. A brief document generated from the job control instance shown in Figure 2.

5.2. Simulation drivers

In order to actually submit a job for execution some knowledge and information of the simulation application code and computing platform is necessary. Rather than represent this knowledge in the job control language, it is put in individual driver routines. This keeps the job control language more generalized and less complex to the end user. Anytime that a new simulation application code and/or new computing platform is to be used, then a driver must be created.

Any knowledge or information that is specific to the application code and computing platform should go in the driver. This type of knowledge and information may include location of input data and where to write the output data, application code initialization parameters, and other environmental variables. Computing platform specific knowledge and information may include the number of processors, run time cutoffs, queuing of jobs, and job priority.

Most of the information must be explicitly spelled out in the driver routine. However, it may be possible to create or write portions of the driver as necessary right before job submission. Some of the environmental variables and user-specific information that does not change could be put in a user profile or property file. Some of the platform dependent information could be determined with metrics or rules. For example, the physical system description has information about the computational size or resolution of the problem and a metric or rule could suggest a reasonable number of processors for executing such a problem.

6. RESULTS AND CONCLUSIONS

Initial results indicate that this system of problem specification and run submission will be most useful for quick-turn around simulations. Large and long running simulations can run on the order of many hours, days, and even weeks. Typically, a user who runs this type of simulation will study the results in detail before re-submitting. Users who do parameter study simulations are expressing the most interest in the system. These users make small changes to one or two parameters continually over the course of many runs. These runs can sometimes number in the hundreds.

Early testing has helped reveal the impact of being able to create physical system descriptions from disparate pieces. Especially important is the use of a database as a repository of known problems and commonly used data, such as materials information. This ability to put together a problem description from interchangeable parts is seen as particularly useful. It creates a flexibility in problem design and creation that was not previously available.

Another useful aspect of the system is the ability to automatically generate documentation. Since the documentation is automatically generated from the physical system description and the simulation job control instance, there is no need for the documentation to get out of sync or to become stale. It is not a separate entity that must be updated by hand. When the physical system description or job control instance is changed, then fresh documentation can be generated. Also, the ability to embed representations of simulation output data (such as plots) directly and automatically into documentation is of particular interest to users.

The simulation job control language can potentially be used with any simulation system that uses XML to describe simulation input files. It is possible that the language could be used with other simulation input files, especially...
those with structured formats, but more time and effort would have to be put into support routines. The selection of XML provides not only a structured and object-based representation and language, but a wide variety of open source and commercial tools for working with XML.

Driver routines are necessary for the actual job submission to a computing platform. While it would be helpful to consolidate this information with the job control language, doing so would diminish from the generalness of the language. Making the language more specific would make it more limited in its use. The driver routines are a tradeoff to keep the language general enough for wide use.

Although the system is best suited for multiple, quick-turn around simulations, it holds potential for a universal simulation job submission tool. Large-scale simulations can be complex to set up and run. Adding to that is the complexity of working with massively parallel computing platforms. There are few users who are experts with more than one simulation code and/or computing platform. By using the job control language and system, a user need not have application code specific and platform specific knowledge to run problems. Even though this use is not as a multiple job submission tool, this use may have value to lay users.

Still missing from the simulation job control language are the constructs necessary to facilitate and control simulation output analysis. Since the language is well suited for multiple job submission for parameter studies, it makes sense that a user will want to make comparisons between job outputs. The needed constructs will identify output parameters for comparison and the types of comparisons to be made. Once done, users will be able to specify some basic analysis as part of their simulation runs. In turn, the results of this analysis will be added to any pertinent documents that are being automatically generated.

REFERENCES