Using Xml Technology for Data Pipeline Interactive Simulation

Sharath G S
4th sem, M.tech[SE]
Dept of CSE, AMCEC

Nirmala S
Associate Professor,
Dept of CSE, AMCEC

Abstract:- **Enhancing software that is efficient, flexible, reusable and easy to work which is a hard task for simulation developers. In this paper we propose the use of XML and its related tools (e.g. JAXB, XQuery, XSLT, and Native XML Database) for the implementation of a technology unified data pipeline targeted to interactive simulation. We introduce a technology-independent conceptual data model as the basis for every simulation framework. We show that XML is a well suited technology to be used in that context. We propose a data modeling methodology that takes its roots from Model-Driven Engineering (MDE). We would be showing a sample implementation that uses XML for transmitting data over the entire simulation loop.**

1. INTRODUCTION

Nowadays, simulation is used extensively by scientists and engineers for designing complex systems or for under-standing intricate phenomena. Typically, "batch run simulations" are exploited for extracting knowledge from vir-tual experiments. Many batch run simulations lead to wasted computation time because errors in simulations or poor choice of simulation parameters are only discovered after a run, something that could have been avoided should interactive simulation be exploited. For being considered as interactive, a simulation should take a reasonable amount of time to execute, typically a few seconds to a few minutes.

Interactive simulation can be seen as the process of steering a simulation while it is executing. Many ap-proaches have been proposed for steering simulations interactive simula-tion is a 3-step process. Firstly, a model of the simulation needs to be built in the Simulation Modeling Module. This step consists of defining the actors participating in the simulation, the properties of each actor and the interactions between actors leading to the desired behav-iors. The scope of this step depends on the architecture of the simulator and on the level of detail required for the models. In addition to the definition of the actors and the interactions, a scenario includes the initial values for the parameters of the actors and the scheduling of "outside" events that will occur during the simulation and whose effect may be, among other things, to modify behaviors of the actors.

At the second step, the Simulation Execution Module accepts the scenario and models that were designed at the modeling stage. It is clear that the Execution Module must be structured so as to understand both the models and the scenario in order to execute the simulation properly and to maintain a coherent internal state. Then, the Execution - Module runs the simulation and updates its internal state accordingly while taking into account the events scheduled in the scenario. The "simulation state" is defined as the set of variables and parameters describing the totality of a si-mulation at a given time step. This simulation state, which is available in a given data format, is sent periodically to the Simulation Analysis Module. The latter module must provide the user with interfaces that allow him to visualize what is actually occurring in the simulation, and with tools that allow him to perform different types of analyses on the data such as statistical analysis or data mining. A major difference between interactive simula-tion and batch run simulation is that, for interactive simulation, the user "closes the loop" by providing input to the models and to the Execution Modulebased on his interpretation of the results.

We propose a design methodology that facilitates the implementation of user interaction with new and existing simulators. The methodology, presented as a conceptual framework that is a formal way of thinking of the XML based conceptional model.

In this paper, we propose a design methodology that facilitates the implementation of user interaction with new and existing simulators. The methodology, presented as a conceptual framework that is a formal way of thinking, is generic and does not make any assumption on the architecture of the simulator. The paper demonstrates that the data pipeline must be designed carefully to ensure successful implementation of interactive simulations. It intro-duces a conceptual framework for this data pipeline, which comprises a data model and a generic data flow.

2. CONCEPTUAL FRAMEWORK

This section presents a conceptual framework that sets up the foundation of a methodology for facilitating interaction with simulations. Figure 1 shows the key building blocks of the conceptual framework. Large rectangular boxes rep-resent storage units, whereas rounded boxes represent processing units. Storage units encapsulate the state of data at a particular stage in the conceptual framework, whereas processing units transform input data and outputs the results.

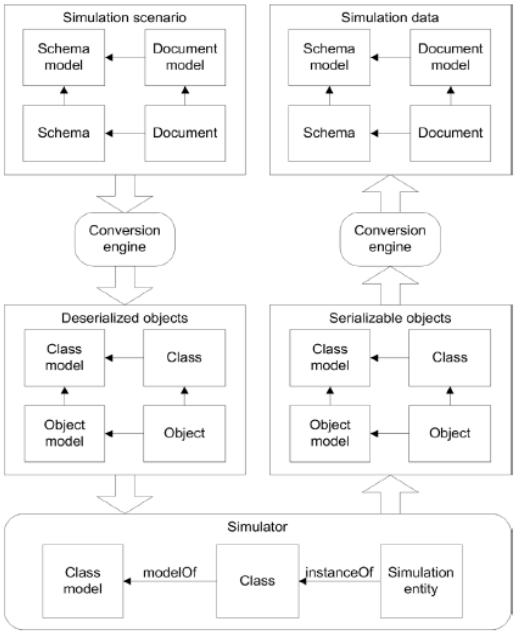

Fig 2: Technology independent conceptual framework

2.1 Data model of the conceptual framework

The first block of the conceptual framework (Figure 1) is the simulation scenario, which contains the elements described(e.g. actors, models, interactions, scheduled events). The tools that are used for building the scenarios may range from sophisticated GUIs to simple text file editors. The "simulation scenario" box, exploded in Figure 1, illustrates the data model the scenario must comply. The scenario "document," which contains the data, is an instance of a "document model." On the other hand, the "document" validates against a "schema," a model defining the document syntax. The schema itself is an instance of the "schema model," a meta-model defining the content of a schema. Once the scenario document is built and validated against its schema, it must be converted to a format that is understandable by the simulator.

For that purpose, a "conversion engine" links the si-mulation scenario and deserialized objects, which are in-termediate storage elements, by performing appropriate processing. The reason for feeding the simulator with deserialized objects instead of the scenario as such is that we want to keep the framework generic and independent of the architecture of the simulator. This genericity constraint has a direct impact on the technologiesthat need to be selected for ensuring smooth integration of the deserialized objects and the simulator. Therefore, the addition of the interactiv-ity feature to a simulator should be transparent to its inter-nal modules. It is worth noting that a single scenario usu-ally generates several deserialized objects, each being an instance of a "class."

As shown at the bottom of Figure 1, we assume that simulation entities exist in the internals of a simulator, regardless of its architecture and implementation. The simulator executes interactions between entities, which generates data that must be serialized and converted back to a format that a user understands in order to be available for analysis.

The serializable objectsare data containers in which the simulator writes the simulation state. They own the same data model as deserialized objects. A conversion engine translates serializable objects to simulation data, which shares the same data model as the simulation scenario.

2.2 Generic dataflow in the conceptual framework

Figure 2 shows a generic dataflow that is suitable for inter-active simulation. The simulation modelingstep shows that a user exploits a scenario editorto produce a scenario document. This document is usually a computer file. The elements contained in the scenario document are converted to deserialized objects and affected to the simulation state. The simulation engine performs calculations on elements composing the simulation state and updates involved enti-ties of the simulation state accordingly. Then, the simula-tion state is copied to serializable objects.

We implemented a mechanism to meet the interactive simulation requirements; serializable objects can be saved to checkpoint files, which act as simulation state containers that a user can modify and reload back to the simulator. On the other hand, the developer of the simulation modeling module of an interactive simulator needs to interface existing code with deserialized and serializable objects in order to load and save the simulation state.

A user is able to control the flow of a simulation with the simulation analysis step, which includes the following modules:

- A **data manager** whose role is to communicate with the simulation for retrieving simulation data into a stream, to manage incoming data, and to process user interaction;

- A **database** that stores selected information in-coming from the simulator data streams;

- A **data stream query** filter that selects, accord ing to the user's needs, information relevant to the simulation analysis;

- A **data analysis** module that is exploited by the user for exploring the simulation and acquiring knowledge of the phenomenon under study.

In summary, we propose in this section a framework that simulation practitioners should implement in order to convert existing software to an interactive simulator. The proposed framework allows the use of traditional simulation methodologies such as batch simulation because simulation data is stored in a database for later consultation.

3. XML AS A UNIFYING TECHNOLOGY

It is claimed that XML is a technology that is well adapted for the implementation of the conceptual framework described. It shows how the generic structure shown in Figure 2 can actually exploit XML technol-ogy to implement the conceptual framework. The suggested XML-based implementation makes the assump-tion that the simulator is implemented in object-oriented technology. We aim towards a design methodology that could be used by simulation practitioners to decrease the development effort when building an interactive simulator from existing software.

Originally, XML was developed as a subset of SGML, intended for web applications. It now describes data in several application areas such as semantic web, mathemat-ics, biological simulations, and military decision making. Some authors propose guidelines to follow, so that researchers use XML technology only where it belongs.

Many tools exist in order to parse and validate XML files, bind XML entities to objects of different programming languages, store XML data in databases, visualize XML documents and schemas, transform XML documents and query XML files (Wikipedia 2008). An advantage of XML over other data formats is its self-description. XML describes its structure, field names and values. The integra-tion of metadata in an XML stream is also straightforward. The resulting plain text is human- and machine-readable and fully portable on different system architectures.

On the negative side, XML is verbose, which results in a waste of bandwidth when transmitted over a network. However, some binary XML formats compress data, making it less redundant and more efficient for processing. Al-so, every piece of data is a string, eliminating the intrinsic data type support that is available in most programming languages. For the conversion between XML and common data types, the marshalling operation transforms common data types (e.g. double, float, integer) to XML strings, whereas the unmarshalling operation transforms an XML string to common data types. Nonetheless, these conversions require considerable amount of processing time and need an XML schema that defines the node types and structure of a given XML file.

3.1 Detailed XML data model

Despite the weaknesses identified above for XML, its portability and simplicity make it an excellent choice for developing the framework for interactive simulation. In addition many tools for processing and handling XML data are available and reduce the development time. It shows how XML is exploited in the conceptual framework described in Figure 1. More specifically, the scenario is stored in an XML document. This document must conform to a set of rules defined in a companion XML Schema. It should be noted that a schema validates only the syntax of an XML document, not the semantics. Therefore, a higher-level mechanism must take care of maintaining the coherence of the scenario.

XML node entities contained in a document are un-marshalled to objects, which are instances of object-oriented programming language classes (e.g. C++, Java, C#), via XML data binding that refers to the process of representing elements of a XML document as objects in computer memory.

A simple mechanism transposes deserialized objects to simulation entities objects. It is usually implemented by copying class attributes to corresponding fields in simula-tion entities objects. Then, the marshalling process, included in the binding library, transforms objects back to XML format. It should be noted that deserialized objects and serializable objects do not necessarily share the same data model. However, adopting the same model for both concepts is recommended because of the resulting uniformity in processing. The same remark applies for the simulation scenario and the simulation data. Both can embody the totality or a fragment of a global and unique schema.

3.2 Design methodology

It highlights important characteristics emerging from the use of XML as the basic building block of the data flow. Boxes with a blue background represent the data modeling methodology of the XML-based framework de-tailed at the left of.

The first step in the methodology consists of building a static UML simulation model. The UML model should contain classes associated to every simulation element. Each class should contain its attrib-utes, the set of classes and attributes making what we call a simulation state,that includes elements part of the initial scenario (e.g. entities with their initial properties and links), as well as others essential at runtime.

The second step in the data modeling methodology is to generate an XML schema from the static UML class di-agram. Tools exist that perform the conversion from UML to XSD, which conforms to the W3C XML schema syntax (Wikipedia 2008a). It shows the schema resulting from the static UML diagram. It starts with the definition of elements, each associated with a class in the UML diagram. Then, every element is defined according to the static view specifications. The XML schema preserves the cardinality as well as types es-tablished in the UML diagram. It is thus the model of sub-sequent XML documents, such as initial scenarios or checkpoint files. It is also the template for object classes that constitute the data model inside the simulation architecture.

The third and final step in the data modeling method-ology is the binding of the XML schema to an object-oriented programming language. This step consists of generating a class associated to every element defined in

the schema. Several existing software suites or libraries can perform this binding task for various object-oriented programming languages. The Java class that is produced by the Java XML Binding (JAXB) compiler using the XML schema presented in. It shows protected attributes that are accessible via public get/set methods. The class also contains several annotations that help the Java runtime environment and compiler to perform object serialization and deserialization.

3.3 XML-specific data pipeline

The data modeling methodology presented in the previous section can be used for setting up a data pipeline. It starts off with an XML document containing the user-defined initial scenario (or a checkpointed simulation). Then, with the XML binding library's unmarshaller, XML elements are automatically converted to objects that are instances of XML bound classes. These intermediate storage elements format depends on the chosen programming language. Then, through native calls, simulation entities are filled with data on which they will execute mathematical operations. Then, the XML binding library marshaller converts these objects.

4. SAMPLE USE OF THE METHODOLOGY

The proposed methodology was applied successfully to the implementation of an interactive simulator using existing open-source software. The system is described below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Entity" type="Entity"/>
  <xsd:element name="InitialScenario" type="InitialScenario"/>
  <xsd:element name="Position" type="Position"/>
  <xsd:element name="Random" type="Random"/>
  <xsd:element name="SimulatedEntity" type="SimulatedEntity"/>
  <xsd:element name="Simulation" type="Simulation"/>
  <xsd:element name="Terrain" type="Terrain"/>
  <xsd:complexType name="Entity">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="InitialScenario">
    <xsd:sequence>
      <xsd:element ref="Terrain"/>
      <xsd:element     maxOccurs="unbounded"     minOccurs="0"
name="entitiesList" type="Entity"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Position">
    <xsd:sequence>
      <xsd:element name="x" type="xsd:int"/>
      <xsd:element name="y" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Random">
    <xsd:sequence>
      <xsd:element name="Seed" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="SimulatedEntity">
    <xsd:sequence>
      <xsd:element name="lifeLeft" type="xsd:double"/>
      <xsd:element name="position" type="Position"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Simulation">
    <xsd:sequence>
      <xsd:element name="timeStep" type="xsd:int"/>
      <xsd:element ref="Random"/>
      <xsd:element ref="InitialScenario"/>
      <xsd:element     maxOccurs="unbounded"     minOccurs="0"
name="simulationEntitiesList" type="SimulatedEntity"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Terrain">
    <xsd:sequence>
      <xsd:element name="width" type="xsd:int"/>
      <xsd:element name="height" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Figure 3. Sample XML Schema

4.1 Pythagoras as a simulator

Pythagoras is a free, open source, agent-based simulator that models agent entities having behaviors as well as several properties (e.g. life left, position, side color) and evolving on a terrain having its own properties. This software was part of Project Albert, which aimed at using high performance computing in order to "understand theunexpected" in a military context. Pythagoras employs brute force computing in order to investigate the proposed problems. It also includes a utility for batch simulation, but does not offer facilities for interactive simulation steering. Pythagoras is implemented in Java and already exploits XML binding technology to load the scenario into its kernel.

4.2 Applying the methodology

Pythagoras lacks several features that are needed to make it interactive. Using Pythagorasoriginal source code and through a thorough reverse engineering process, we implemented various functionalities, in compliance. However, since the simulator was already programmed, the approach of reverse engi-neering from source code to UML, then UML to XML schema, would have slowed down the design process. Hence, the XML schema was edited by adding essential elements for dumping a simulation state to a file. The to XML in the form of streams or documents. The check-point operation becomes trivial; it consists of writing the entire simulation state to an XML document.

4.3 Lessons learned

The following lessons were drawn from the experience of applying the proposed methodology with the Pythagoras simulator as a test bed:

- Open source code facilitates the application of the methodology. The more control a user has on software, the easier it will be for him to modify existing functionalities and add new ones. For the current demo, the learning curve was steep. How-ever, reverse engineering tools help developers to get a better hand on complex software architec-tures.

- MDE generally applies if a model is already available. In completing the current work, we did not use MDE, but rather did modify the XML schema manually. It was easier to do so because the advantage of using current software engineer-ing tools over manual techniques was not clear. Future work section discusses the use of auto-mated tools for solving that problem.

- The use of standard data formats facilitates integration.Since XML was initially used by Py-thagoras as the basic data format, the integration with the proposed methodology was straightfor-ward. Other data formats should provide serializa tion and deserialization methods in order to allow for the proposed methodology to be exploited suc-cessfully. Several data formats other than XML exist that meet these requirements. However, XML offers additional functionalities that other data formats are lacking (e.g. data manipulation with XQuery and XSLT, direct data binding, sim-ple visual representation).

- The compiled schema semantics during XML binding is limited.The XML schema binding compiler that we used in the current implementa-tion does not support relations other than “aggre-gation” and “attribute” between elements. Rela-tions such as “inheritance” were added through the development of a compiler plug-in that offers a richer semantic to be added to the data model.

- The methodology requires a minimal set of essen-tial functionalities to be supported by the simulator.The only essential feature that a simulator must implement is the ability to checkpoint and resume simulations. Other features ease the development of advanced interaction, but are not mandatory.

5. FUTURE WORK AND CONCLUSION

We showed that the methodology presented in this paper can be successfully applied for developing a specific application. However, it can be improved in several ways. First, the data modeling framework could be fully automated. In fact, using a stereotype on appropriate classes, the UML diagram could be converted to XSD, the XSD compiled to source code and methods that copy data to/from objects automatically generated. This process is relevant for a new simulator design and one that was reversed engineered.

Also, we are currently designing and implementing a generic visualization environment that will allow its users to manipulate data in an immersive virtual reality environment. We plan on integrating our entire data pipeline, so that multiple simulation instances can be visualized simul-taneously.

Finally, the transformation of Pythagoras from a batch run type of simulator to an interactive simulator is the beginning of a long term project. We plan on modifying several additional simulators and experiment whether or not users perform better in the understanding of a complex sys-tem model using the interactive version.

REFERENCES

[1] Boyno, E. A. 2006. XML: What, What, Who and Where. ISECON.
[2] Atkinson, C., and T. Kuhne. 2003. Model-driven develop-ment: a metamodeling foundation. Software, IEEE 20(5): 36-41
[3] Brooke, J. M., P. V. Coveney, J. Harting, S. Jha, S. M. Pickles, R. L. Pinning, and A. R. Porter. 2003. Compu-tational Steering in RealityGrid. UK e-Science All Hands Meeting: 2–4.
[4] Kent, S. 2002. Model Driven Engineering. Integratted Formal Methods. Third International Conference, IFM: 15-18.
[5] McGrath, R. E. 2003. XML and Scientific File Formats. 2003 Seattle Annual Meeting.
[6] IBM 2008. Rational Rose.Available via <http://www.rational.com>[accessed June 23, 2008].