Lightweight Data Management for Compositional Modeling in Problem Solving Environments

Alex Verstak*, Marc Vass*, Naren Ramakrishnan*, Cliff Shaffer*, Layne T. Watson*,
Kyung Kyoon Bae†, Jing Jiang†, William H. Tranter†, and Theodore S. Rappaport†,
Department of Computer Science*
Bradley Department of Electrical and Computer Engineering†
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061
Contact email: averstak@vt.edu

Keywords: Data management, markup languages, wireless system design, problem solving environments.

Abstract

We outline various design considerations and implementation options pertaining to lightweight data management in problem solving environments (PSEs). The emphasis is on compositional modeling, in the context of a PSE for wireless communications system design (S^4W). By focusing on a restrictive subset of markup languages, we describe how facilities such as management of the execution environment, experiment management, and reasoning about model sequences can be provided.

1. INTRODUCTION

Compositional modeling refers to the ability to 'combine representations for different aspects of a computation to create a representation of the system as a whole' [4]. It serves as a powerful programming abstraction for designing complex problem solving environments (PSEs), such as those envisaged under the NSF's Next Generation Software (NGS) program. Examples of systems that provide compositional modeling include the Linear System Analyzer (LSA) [6], the Component Architecture Toolkit (CAT) [1] at Indiana University, the ZOO desktop experiment management environment at the University of Wisconsin [8], Application Visualization System of Advanced Visual Systems, Inc. [14], Nayak's theoretical framework [12], and the SCIRun computational steering system at the University of Utah [9].

Data management for compositional modeling involves:

1. the automatic generation and execution of solution components by selecting and composing model fragments from a library, using general and domain-specific constraints on their structure,
2. mechanisms to support declarative modeling and visualization of model sequences as an aid in graphical and collaborative composition, and
3. algorithms for reasoning about model sequences, given constraints on performance and representational goals.

Traditionally, PSEs support one or at most two of the above three goals. For example, the LSA and CAT systems support (1) by using mechanisms such as distributed OO and inheritance to achieve composition. SCIRun, ZOO, and LSA support (2) but do not provide sophisticated capabilities for reasoning since interfaces to low level code are hardwired. AVS uses a data-flow programming model to support both (1) and (2) but it is more oriented towards generating visualization applications from well-understood and ready-made network components. In particular, goal (3) is not one of the motivating considerations. Nayak's compositional modeling framework uses causal approximations to find 'satisficing' (sic) model sequences (goal (3)), but its considerations are not performance-driven and are more motivated by the desire to formalize the modeling process using an underlying domain theory.

The goal of lightweight data management is to resolve these differing (and often conflicting) PSE design goals, by providing expressive and high performance access to objects and streams (for experiment management) with minimal overhead (in terms of traditional database functionality such as transaction processing and integrity maintenance). Such additional services, if needed, are typically provided at a higher level of abstraction [5].

In this paper, we describe our experiences with creating a lightweight data and experiment management system for a wireless system design PSE (S^4W). S^4W is a collaborative application design and support system that incorporates high-fidelity site-specific propagation and channel models, parallel computing, recommender systems, a design optimization loop, and a composition environment. S^4W is designed to provide superior software performance by (i) developing fundamentally better wireless communication models, (ii) constructing better simulation systems composed from the component wireless models via a recommender system, and (iii) the transparent use of parallel high-performance computing hardware via the composition environment's access to distributed resources.

Our experiences with S^4W show that it is possible to achieve lightweight data management by focusing on a restrictive subset of XML-based markup languages. We outline the design of our language and describe its implementation for a family of wireless propagation and channel models. Relevant issues including search space representation (for model composition), the software engineering of conversion utilities (for overcoming impedance mismatch), implications of database design choices, and algorithms for efficient and effective querying and mining of model spaces are briefly mentioned.

The rest of the paper is organized as follows: Section 2 introduces and motivates the design of the S^4W system. Sections 3, 4, and 5 describe three aspects of S^4W and their implications for data management. Section 6 provides a summary discussion.

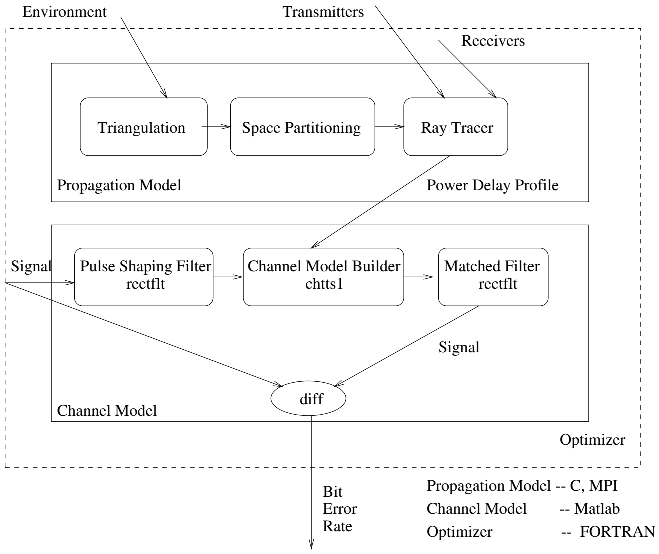

Figure 1: An example model from the S^4W project.

2. OVERVIEW OF S^4W

Broadband wireless systems are essential for the success of the Next Generation Internet (NGI) and future generations of portable multimedia communicators, but adequate design and analysis tools do not presently exist. Remarkable improvements in computing power and new satellite imaging techniques have recently led to fundamentally new approaches to predicting wireless system performance through the use of geographical information, specific building placement and architectural information, along with numerical electromagnetic propagation and ray-tracing models. The reader is referred to [13] for an in-depth treatment.

While primitive software tools exist for cellular and PCS system design, none of these tools include models adequate to simulate broadband wireless systems, nor do they model the multipath effects due to buildings and other man-made objects. This is a key factor that limits the performance of devices such as wireless modems. Furthermore, currently available tools do not adequately allow the inclusion of new models into the system, visualization of results produced by the models, integration of optimization loops around the models, validation of models by comparison with field measurements, and management of the results produced by a large series of experiments. Our collaborative PSE — 'Site-Specific Systems Simulator for Wireless Communications' (S^4W) — is motivated by all of these concerns.

The operational strength of S^4W relies on efficient data modeling that supports the experiment definition, data acquisition, data analysis, and inference processes. In this paper, we concentrate on managing a high performance execution environment, experiment management, and reasoning about model sequences. In a future paper, we intend to address higher level problem solving abstractions such as recommendation and knowledge-based reasoning.

3. MANAGEMENT OF THE EXECUTION ENVIRONMENT

One of the most basic functionalities expected of a PSE is management of the execution environment, specifically, mapping abstract models and specifications onto actual codes that can be executed on the available hardware. While PSEs almost universally use the dataflow paradigm to represent compositions of components, a variety of low-level issues have to be addressed for mapping onto architectures such as grids. Resource management, fault tolerance, and distributed scheduling services have to be provided before a graph of abstract components, which we call a *model*, can be executed. We do not discuss these further in this paper but the reader is referred to [5] for an in-depth treatment.

A simplified example of an S^4W model is provided in Fig. 1. It addresses the wireless design problem of finding the positions of base stations in a square mile area of a large city such that the coverage is optimal. In terms of execution, this model is a graph of computational components enclosed in an optimization loop. The curved rectangles correspond to the components that are a mix of C, Matlab, and Fortran programs and functions. The ray tracer is a parallel MPI program that runs on a Beowulf cluster of workstations. Features such as aggregation are used to simplify the interfaces to the optimizer and/or support creation of new components in a visual programming style, as opposed to developing them in a language like Matlab. In Fig. 1, the straight rectangles express aggregation, e.g. the propagation model is a component that consists of three subcomponents: triangulation, space partitioning, and ray tracing.

Several interesting observations about the data flows in this example can be made:

1. Data is exchanged between model executions in three different languages. PSEs typically force data conversion to a language-neutral intermediate form for interoperability.

2. Such a simple solution will not work for realizing the optimization loop because data is streaming between the optimizer and the simulation. In addition, it will not support scenarios such as computational steering [9].

3. A variety of intermediate results are produced, not all of which are direct performance data. Such data can be cached to improve performance, visualized at different stages of the execution, or simply saved for later inspection. PSEs typically provide rudimentary solutions in the form of database systems.

4. In a multidisciplinary application such as S^4W, the components are developed at different times by different researchers (in our project, many are are still under active development). Their I/O specifications hence cannot be enumerated in advance to achieve matching of components. Further, the possibilities of how components could be cascaded and combined can itself evolve over time. For example, the static channel model builder can be replaced with one for dynamic and mobile environments thus enabling a fundamentally different simulation methodology.

This scenario, typical for a PSE, is not well supported by traditional database designs such as relational schemas and object oriented (OO) design. Relational data is resistant to changes in the schema; OO data and various kinds of IDLs are not well-suited for PSEs either, because of their inability to work with legacy software in Matlab and/or Fortran. The impedance mismatch between objects in a OO database system and functions and routines in scientific codes implies that only components designed from a pure OO starting point (e.g., CAT [1]) will work seamlessly.

Our emphasis here is on *semi-structured* data representations; a very popular example is XML [15], which is language-neutral, can be streamed, stored in a database (e.g., by Lore [10]), visualized, and converted (e.g., by XSL [17]) in a generic fashion. XML is not without its disadvantages either. There exists a plethora of tools that perform various document processing functions, but simply reading a list of polygons encoded in XML into a C ray tracing program can require over a hundred lines of code. We address binding XML data to PSE components in the following sections. Fortunately, the XML mindset makes it possible to design these bindings in a generic manner.

Binding

Binding refers to the process of converting XML data to an appropriate representation in a scientific computing language (the reverse process is fairly straightforward). There are three basic forms of binding:

1. Binding of values to language variables.

2. Converting an XML format to some native format that can be read directly by the component.

3. Generating source code for a stub that contains embedded data and a call to the appropriate language function using these data as parameters.

The latter two forms of binding can be reduced to associating values with language variables and then printing/storing these values in an appropriate format, so these can employ solutions to the first problem.

The first step towards such bindings is a validation language, e.g., XML Schema [18] or SOX. However, validation languages are targeted at small documents with sophisticated constraints, which makes their use for voluminous scientific data impractical. Custom solutions for specific languages, e.g., [11], are too inflexible to work in a multi-lingual PSE framework. Various IDLs, e.g., SIDL [3], make the assumption that the interfaces are homogeneous, which does not hold for PSEs. Generic parsers like YACC are too low-level to be useful because the amount of parsing code can exceed the amount of code that performs the computation. It appears that the right language is at the intersection of these ideas. We call such languages BSMLs (Binding Schema Markup Languages).

Our BSML associates user-specified blocks of code with user-specified blocks of an XML file. 'Blocks' can be primitive datatypes, sequences, selections, and/or repetitions. Intuitively, primitive datatypes denote single values, such as double precision numbers; sequences denote structures; selections denote multiple choices of conveying the same information; and repetitions denote lists. While not particularly expressive, this notation is meaningful to PSE component developers, simple and efficient implementation-wise, and general enough to allow the building of more complex data representations.

Consider, for example, representing a power delay profile (PDP) in XML. A PDP is a two-column table that describes the power received at a particular location during a specified time interval. Statistical aggregates derived from power delay profiles are used, for example, to optimize transmitter placement in S^4W. A PDP can look like this:

```
<pdp>
 <rmsDelaySpread>49.4872</rmsDelaySpread>
 <meanExcessDelay>57.1429</meanExcessDelay>
 <peakPower>-34.7712</peakPower>
 <time>0</time>   <power>-Inf</power>
 <time>10</time>  <power>-Inf</power>
 ...
 <time>90</time>  <power>-34.7712</power>
 <time>100</time> <power>-Inf</power>
 ...
 <time>190</time> <power>-36.0206</power>
</pdp>
```

A BSML description of a class of XML documents that contain PDPs may then be the following:

```
<element name='pdp'>
 <sequence>
  <element name='rmsDelaySpread'
                       type='double'/>
  <element name='meanExcessDelay'
                       type='double'/>
  <element name='peakPower'
                       type='double'/>
  <code component="optimizer">
   <bind>print "$peakPower\n"</bind>
  </code>
 </sequence>
 <repetition>
  <sequence>
   <element name='time' type='double'/>
   <element name='power' type='double'/>
   <code component="chtts1|chttm">
    <bind>print " $time $power\n"</bind>
   </code>
  </sequence>
 </repetition>
 <code component="chtts1|chttm">
  <begin>print "M = [\n"</begin>
  <end>print "];\n"</end>
 </code>
</element>
```

Applying a parser generated from this BSML document to a PDP will yield the following Matlab source. Adding a call to the appropriate channel model builder will complete an executable Matlab script that contains embedded data.

```
M = [
 0 -Inf
 10 -Inf
 ...
 90 -34.7712
 ...
 190 -36.0206
];
```

In other words, we can rapidly prototype new simulations with this technique. Similarly, we can use the same BSML source to provide bindings for the optimizer. The feedback will be a sequence of peak powers, one number per line. Some twenty five lines of BSML source can therefore take care of data interchange problems for three components. Storing these PDPs in a database is also facilitated.

To summarize, XML data representations can be used to advantage in PSEs for managing the execution environment. Although originally targeted at document processing and business data interchange, the few design decisions provided here support XML as a lightweight format for PSEs.

Format Conversions and Change Management

One of the benefits of semistructured data is automatic format conversion. This feature is useful in the following situations:

1. A component is changed over time, but data corresponding to the older versions has already been recorded in the database system. An example from S^4W is the evolution of the space partitioning parameters in the ray tracer. After we have realized that placing polygons at the internal nodes of the octree can improve space usage by an order of magnitude, more parameters have been added to space partitioning.

2. Several components need essentially the same parameters, but are not truly plug-and-play interchangeable. Minor massaging is necessary in order to make their I/O specifications match.

We model the following changes: insertions, deletions, replacements, and unit conversions. Insertions and deletions correspond to additions and removals of parameters. For example, a moving channel builder takes the same inputs as a static one, plus the velocity of the receiver. Thus, any input to a moving channel builder can be converted to the input to a static one by projecting out the receiver's velocity. Replacements represent changes in parameter representation, for example, a conversion between spherical and rectangular coordinates. Unit conversions are a special case of conversions that are quite common and can be easily automated, for example, conversions between Watts and decibel milli-Watts. Unit conversion can be performed by equation and constraint solvers [4].

In our XML representation, insertions can be handled by requiring default values for new parameters. Removals amount to deleting the old values. Replacements and unit conversions require user-supplied or automatically generated conversion filters. The modeling literature abounds in such conversions, but it is important to realize that conversion facilities are adhoc by nature, and therefore only work for small changes in the schema. Typically, it is not necessary to find a globally optimal conversion sequence. A thorough treatment of change detection can be found in [2].

4. EXPERIMENT MANAGEMENT

We define an *experiment* as a collection of models instantiated with input, intermediate, and output data. This captures the notion of applying multiple models to multiple inputs to generate a database of simulation results and performance data.

In the database paradigm, an experiment can be represented as a view. Executing the experiment corresponds to materializing the view. The query behind the view is a join over models and data. In order to be meaningful, an experiment must further satisfy some *syntactic* and/or *semantic* constraints. Syntactic constraints ensure that the experiment can indeed be executed. Each simulation run must be given enough data and the data must conform to the appropriate schemas. Semantic constraints ensure that the models are meaningful in the specific problem domain. We will describe semantic constraints in the next section. In S^4W, users can impose custom constraints, such as 'use only the datasets from last week.' Experiment specification therefore maps naturally into a database query.

Consider the following scenario. A developer of ray tracing propagation models has added a model that takes diffraction into account. She now wants to re-calculate the PDPs for the environments where diffraction is most significant, e.g., for urban outdoor environments. Experiment specification in an XML-QL-like notation [16] may look like:

```
<experiment id='diff. prop.'>
 WHERE <environment id='$id'>
  <meta><type>urban</type></meta>
 </environment> CONTENT_AS $env IN "envs"
 CONSTRUCT <experiment id='diff. prop.: $id'>
  <model>...</model>
  <inputs>
   <input>$env</input>
   ...
  </inputs>
  <outputs>...</outputs>
 </experiment>
</experiment>
```

The result of this query is an experiment, which in turn is a sequence of simulation runs. Not only is this form of experiment specification concise, it also enables us to use well-known query optimization techniques to push costly operations 'deeper' into the computational pipeline [7].

5. REASONING ABOUT MODELS

What constitutes a good model? PSEs should provide the facility to reason about a model and its constituent parts in terms of the features of the problem being solved and the desired performance constraints. A lot of domain-specific modeling [12] is required to arrive at promising model choices, but a few general rules can be outlined:

1. A model must not contain any components that make incompatible assumptions about the phenomena being modeled. Following Nayak, we call such components *contradictory*. An example of contradictory components in S^4W is a class of model builders (static and moving).

2. Some modeling choices may constrain the form of the rest of the model. For example, the signal filters of the transmitter and the receiver must *match*.

3. Components in a given class, say filters, often support similar forms of reasoning. We use the term *classification* to describe this aspect.

An example of these relations in S^4W is given in Fig. 2. The labels represent a small component library and the links represent the relations. Note that these relations are domain-specific and cannot be derived from the source in any general-purpose language. They must be supplied by the user (wireless system designer) as annotations to components. Such relations can then be used to prune the search space for recommendation and problem-solving.

6. CONCLUDING REMARKS

The eventual success of the proposed methodologies relies on the expressiveness of the representations supplied to the domain scientist and his ability to reason efficiently with such representations. By providing a lightweight data model that manages the execution environment, enables change management, and casts experiment evaluation as querying, we have shown how a system like S^4W can provide high-level problem solving capabilities. In future work, we plan to investigate knowledge-based techniques for reasoning about model sequences and incorporation of recommender systems (for selecting among various choices of simulation models) into our framework.

Acknowledgements: The work presented here was supported by National Science Foundation grant EIA-9974956. The authors acknowledge Frederica Darema, National Science Foundation, for helpful discussions on compositional modeling and recommender systems.

References

[1] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yochuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of HPDC 2000*, 2000.

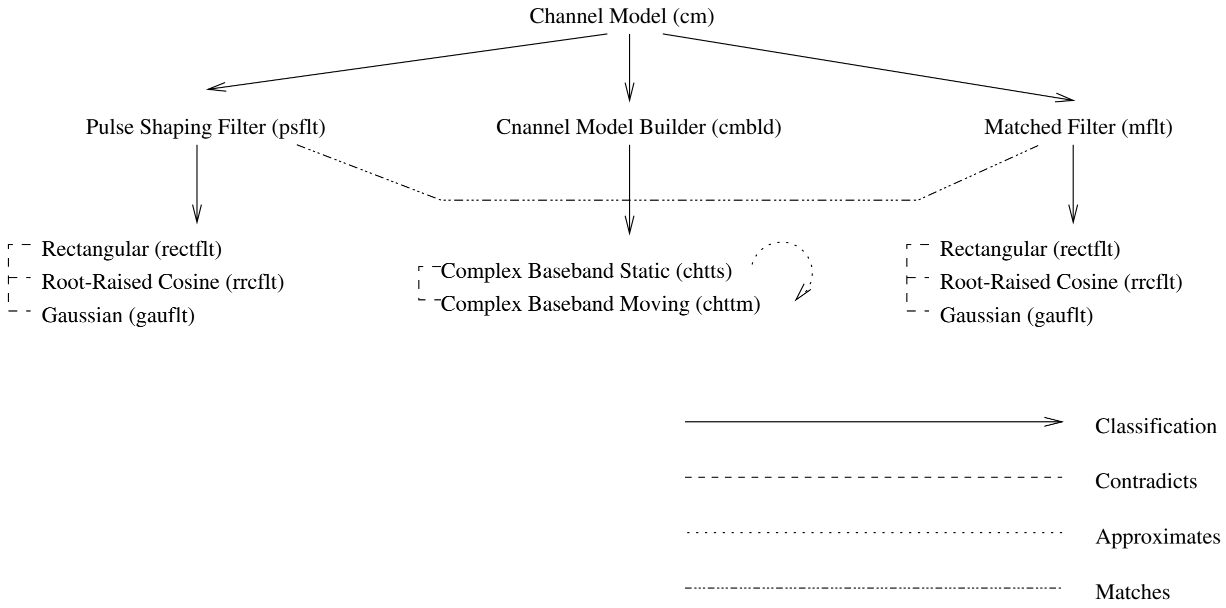

Figure 2: Relations between channel modeling components in S^4W.

[2] S. Chawathe and H. Garcia-Molina. Meaningful Change Detection in Structured Data. In *Proceedings of the ACM-SIGMOD Conference on Management of Data, Tucson, Arizona, USA*, pages 26–37, 1997.

[3] A. Cleary, S. Kohn, S.G. Smith, and B. Smolinski. Language Interoperability Mechanisms for High-Performance Scientific Applications. Technical Report UCRL-JC-131823, LLNL, 1998.

[4] K.D. Forbus. Qualitative Reasoning. In A.B. Tucker, editor, *The Computer Science and Engineering Handbook*, pages 715–733. CRC Press, 1996.

[5] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.

[6] D. Gannon, R. Bramley, T. Stuckey, J. Villacis, J. Balasubramanian, E. Akman, F. Breg, S. Diwan, and M. Govindaraju. The Linear System Analyzer. In *Enabling Technologies for Computational Science*, pages 123–134. Kluwer Academic Publishers, 2000.

[7] J.M. Hellerstein. Optimization Techniques for Queries with Expensive Methods. *ACM Transactions on Database Systems*, Vol. 23(2):pp. 113–157, September 1998.

[8] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A Desktop Experiment Management Environment. In *Proc. 22nd International VLDB Conference*, pages 274–285, 1996.

[9] C. Johnson, S. Parker, and D. Weinstein. Large-scale Computational Science Applications Using the SCIRun Problem Solving Environment. In *Proceedings of the 15th Supercomputer Conference, Mannheim, Germany*, June 2000.

[10] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, Vol. 26(3):pp. 54–66, September 1997.

[11] Sun Microsystems. Long-Term Persistence of JavaBeans, January 2001. Java Specification Request 57.

[12] P.P. Nayak. *Automated Modeling of Physical Systems*. PhD thesis, Stanford University, 1992.

[13] T.S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, 1999.

[14] C. Upson, T. Faulhaber, D. Kamins, D. Schlegel, D. Laidlaw, F. Vroom, R. Gurwitz, and A. van Dam. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications*, Vol. 9(4):pp. 30–42, 1989.

[15] W3C. Extensible Markup Language (XML) 1.0. Technical Specification, February 1998.

[16] W3C. XML-QL: A Query Language for XML. Technical Specification, August 1998.

[17] W3C. Extensible Stylesheet Language (XSL) Version 1.0. Technical Specification, November 2000.

[18] W3C. XML Schema Part 0: Primer. Technical Specification, April 2000.