A Knowledge-based Software Development Environment for Scientific Model-building

Richard M. Keller
Michal Rimon
Sterling Software
Artificial Intelligence Research Branch
NASA Ames Research Center
Keller@ptolemy.arc.nasa.gov

Abstract
The construction of scientific software models is now an integral part of the scientific exploration process. Even though scientific models are difficult and time-consuming to build, completed scientific models are infrequently distributed and shared within the scientific community. A major barrier to distribution is typically the low-level, obtuse, and idiosyncratic nature of the implemented code. To simplify the building and sharing of models, we have designed a special-purpose, high-level dataflow language for use in specifying scientific models. We have also constructed a prototype software tool that uses extensive background domain knowledge to aid in acquiring and executing data flow specifications. We illustrate our approach with examples from the scientific domain of planetary atmospheric modeling.

1 Introduction
1.1 Motivation
Although model-building is an integral part of the scientific enterprise, there is little software engineering support available to the scientist performing this task. Without such support, scientific model-building can be a time-intensive and painstaking process, frequently involving the design and development of large, complex pieces of software. Unfortunately, the modeling software developed as part of this process cannot easily be distributed and shared with other scientists; the implemented code is typically too low-level, idiosyncratic, and complicated for anyone outside the original scientific development team to understand. In particular, the high-level structure and content of the scientific model is not obvious from the low-level code. In this respect, scientific models suffer from the same types of reuse problems associated with conventional software [1].

Nevertheless, scientific models are unique in at least one important respect: the code constitutes a scientific theory. As such, the code is a genuine scientific artifact that should be shared and validated within the scientific community. The code should come under close scrutiny by professional colleagues, just as journal articles are carefully reviewed prior to publication. Typically, however, many important scientific details and modeling assumptions remain implicit in the code, and never appear in a scientific publication. These implicit assumptions cannot be easily recovered or modified by a scientist interested in reusing portions of a scientific model. Thus scientists are legitimately hesitant to reuse unfamiliar code; the inappropriateness of implicit underlying assumptions could invalidate the scientific results.

Aside from comprehensibility, there are other important problems associated with building scientific modeling software. Often, due to scarce resources, scientists must serve in the dual role of scientific domain expert and programmer. Their programming skills may not be as finely honed as their scientific skills, and coding can take a disproportionate amount of time and effort. In all likelihood, a scientist's valuable time could be better spent analyzing scientific results rather than programming. Scientific coding itself presents special challenges to a programmer. Often the computations are numerically sensitive and require application of sophisticated numerical analysis techniques. The coding may also involve setting up I/O operations to access large numerical datasets. Finally, there can be significant bookkeeping effort required to keep track of scientific units and to handle scientific units conversion as necessary to maintain the consistency of the numeric computation throughout.

1.2 Approach
To address the problems associated with development and reuse of scientific models, we are building an interactive knowledge-based software development environment. The specific goal of the SIGMA (Scientists' Intelligent Graphical Modeling Assistant) project at NASA Ames Research Center is to provide computational support for scientists engaged in computer modeling and simulation of physical systems. Examples of such
systems include planetary atmospheres, forest ecosystems, and biochemical systems. Generally, these systems can be modeled as a set of algebraic and ordinary differential equations, where the terms in the equations interrelate the physical quantities of interest.

The SIGMA system provides scientist-users with a high-level graphical data flow language in which to express scientific models. Within this language, users can specify how scientific quantities of interest can be computed by applying equations to known input data. The terms in SIGMA'S language reference familiar domain-specific scientific constructs (e.g., specific physical quantities, scientific equations, and datasets). The language is at a sufficiently high level that it omits most implementation details involving data structures and control. SIGMA'S reasoning components infer these details by referencing extensive domain knowledge and by carrying out clarifying interactions with the user. Once the specification is complete, SIGMA can execute the scientific model and provide visual feedback on the results by displaying data plots.

1.3 Example

As an example of the type of model that can be expressed in SIGMA'S language, consider a fragment of a planetary atmospheric model developed to investigate the thermal properties of Saturn's moon Titan [2,3]. The purpose of this fragment is to develop a profile of Titan's atmosphere that describes the pressure, temperature, and density of gases at various altitudes above its surface. The major source of relevant experimental data is the Voyager-I flyby of Titan back in November 1980. As Voyager-I reached the far side of Titan, it sent back radio signals that passed through Titan's atmosphere and then on to receiving stations on Earth. Due to the density of gases in the atmosphere, the radio waves were refracted slightly as they passed through the atmosphere, resulting in a diminished signal picked up on Earth. The amount of refraction was measured at different altitudes above the surface. This refraction data serves as a starting point for inducing the desired atmospheric profile.

The data flow graph in Figure 1 describes this atmospheric model fragment. In this graph, the lettered nodes represent scientific quantities and the numbered nodes represent scientific equations. The model computes the temperature (T) at some altitude point above Titan's surface based on input refractivity data (r) from Voyager-I. From this, the atmospheric profile is determined as follows (see Figure 1). First, for each atmospheric point profiled, the measured refractivity data (r) is used to compute the number-density (n) of the gases at that altitude. (The number-density of a mixture of gases is defined to be the number of molecules per volume of gas.) If the identity and relative percentages of gases in a mixture is known, the number-density can be computed as a function of refractivity using Equation 1. Next, using the molecular weight of the various gases in the mixture, the mass-density (ρ, or mass per volume of mixture) can be computed from the number-density using Equation 2. The hydrostatic law can then be used to determine the pressure (P) from the mass-density by numerically integrating the weight of the atmosphere above each profile point. Finally, the temperature (T) can be determined from the mass-density and the pressure by applying an equation of state, such as the ideal gas law (Equation 5).

This data flow graph represents a kind of high-level specification for a scientific model. The specification is expressed at the level of abstraction at which a physicist might describe the model, and is far more comprehensible and reusable than the corresponding FORTRAN code. The goal of the SIGMA project is to enable scientists to construct and manipulate models at this level of abstraction, thereby facilitating modification and reuse.

1.4 Technical challenge

The goal of acquiring scientific models in the form of high-level data flow graphs represents a significant specification acquisition challenge. This can be illustrated in terms of the data flow graph in Figure 1. Although domain specialists may intuitively grasp the meaning of this data flow graph, its semantics are incompletely specified and ill-formed. For the graph to represent a well-formed, executable program, each symbol must be uniquely and unambiguously described. For example, to uniquely describe the role of the symbol p in the graph, that symbol must first be identified as a type of mass-density quantity, rather than a type of refractivity quantity, for instance. But this alone is insufficient, as there are many legitimate mass-density quantities that could be referenced by p. In this instance, p is the mass-density associated with a mixture of gases in the atmosphere -- rather than the mass-density of the moon Titan, for example. The mass-density quantity and the associated mixture of gases are as much a part of the meaning of the data flow graph as the syntactic symbol p itself. In other words, the semantics of the symbols in the data flow graph are bound up with scientist's underlying conceptual model of the phenomenon being studied. The symbols in the graph do not make sense in a vacuum -- they only make sense with reference to objects and quantities that describe Titan's atmosphere. What this suggests is that the symbols in the data flow graph must be tied to the domain semantics, either by direct user specification or by inference.
Our approach to dealing with this specification acquisition problem is extremely knowledge-intensive. We believe that the system must have extensive knowledge of the scientific problem under study in order to interact intelligently and synergistically with a scientist to create modeling software. Without this shared understanding, SIGMA would have to rely on user guidance to repeatedly disambiguate user intent during the model-building process. Frequent intervention would likely increase user frustration and decrease the utility of the system to an unacceptable level. A beneficial side-effect of utilizing domain knowledge is SIGMA's ability to verify the consistency of an evolving specification and to advise the user of potential specification problems.

In the balance of this paper, we describe how SIGMA uses domain knowledge to interact intelligently with the scientist-user and to cooperatively construct an executable data flow specification such as the one shown in Figure 1. Section 2 begins by introducing SIGMA's knowledge base, which provides the foundation for the model-building interaction. Section 3 describes the overall model-building and model-execution process. Section 4 reviews related research. Finally, Section 5 discusses current status and future directions, and Section 6 concludes.

2 SIGMA's domain knowledge

SIGMA's domain knowledge is represented and stored in a hierarchically organized, frame-structured knowledge base of over 500 concepts. These concepts encode a variety of different types of knowledge, including information about scientific equations, physical quantities, scientific units, numerical programming methods, scientific domain concepts, and bibliographic citations. A partial overview of the knowledge base is depicted in Figure 2. The construction of this knowledge base represents a considerable expenditure of resources on our part. Our careful attention to construction of the knowledge base and the associated ontology is motivated by the goal of reuse; we want to make the knowledge reusable across a variety of scientific modeling domains.
SIGMA's knowledge can be partitioned into four categories:

1. **Cross-disciplinary scientific knowledge**: Knowledge available to persons with a general scientific background, including knowledge about various physical quantities, scientific domain objects, scientific measure units, foundational equations, and scientific handbook data.

2. **Discipline-specific scientific knowledge**: Quantities, domain objects, equations, and data pertaining to a specific scientific discipline (e.g., biology, ecology, physics).

3. **Problem-specific knowledge**: Domain objects and relations pertaining to the specific physical system being modeled by the scientist.

4. **Programming knowledge**: Knowledge about numerical programming methods, data structures, control, etc. (In the current version of SIGMA, much of this knowledge is implicit in the model interpreter.)

SIGMA's representation language, called RML [4], is built on top of CommonLisp and CLOS. RML features standard inheritance, plus an integrated constraint language that enables specification of arbitrary first order constraints on slot values. Each slot assertion is translated into a horn clause, and constraints can be checked against these clauses using a Horn-clause theorem prover. The RML representation language is based on CYCL, the language used in the CYC project [5]. A more detailed discussion of SIGMA’s knowledge base and representational structures is outside the scope of this paper.

### 3 Model-building in SIGMA

In this section, we provide an overview of model-building within the SIGMA system. The first subsection introduces the problem of model-building as a planning problem, and presents the inputs to the SIGMA's model-builder. The next subsection describes how users interact with the system to build models. The final subsection discusses how SIGMA's model-interpreter executes these data flow models.

#### 3.1 Model-building as a planning problem

Within SIGMA, we view the task of building a computational model as a specialized type of planning problem. Given an initial world state in which the values of some quantities are known and others are not, the goal is to create a computation plan that calculates a desired unknown quantity from the known quantities. The planning operators for this planning problem correspond to the scientific equations that can be applied to produce new quantities from known quantities. The final computation plan corresponds to a data flow graph model.

SIGMA's model-builder module requires as input (1) an initial world state, (2) a goal quantity to compute, and (3) a set of equations. We discuss these inputs in the following subsections.

![Figure 2: Overview of SIGMA's knowledge base](image)
Initial world state: The initial world state takes the form of a network structure of interrelated domain objects, each of which has an associated set of known and unknown physical quantities. This representation is similar to the lumped parameter models developed for qualitative reasoning about physical systems [6]. The initial world state reflects the scientist’s understanding of the relevant objects and relations that are necessary to describe the physical system being modelled. To illustrate, some instantiated objects and relations in the initial world state for the Titan atmospheric modeling domain are shown in Figure 3. The atmosphere of Titan is modelled by a sequence of atmospheric-parcel objects stored in the “parcels” slot of the “Titan” object. Each atmospheric-parcel represents a volume of gas at a specified altitude above Titan’s surface. The atmospheric-parcel contains a mixture of pure gases, each of which is described by a constituent object. The atmospheric-parcel and the constituents are being irradiated by a common radiation-source: the Voyager-radiation object. The radiation-interaction object represents the collision of the radiation with the irradiated material. Quantities that are a function of both the radiation source and the irradiated material are stored in this radiation-interaction object. For example, the amount of refraction caused when the Voyager radio signal intercepted the atmospheric parcel is a function of both the wavelength of the Voyager signal and the properties of the gaseous material in the parcel.

Goal quantity: The goal quantity to be computed by the model can be any unknown quantity of an object found within the initial world state. For example, the temperature of atmospheric-parcel-32 is an unknown quantity in the world state depicted in Figure 3. Using the SIGMA interface, the scientist can specify the goal quantity he or she wishes to compute.

Equations: The set of equations used by SIGMA’s model-builder consists of a set of generic physics equations that can be applied to compute new quantities from known quantities in a given world state.

Each SIGMA equation consists of a syntactic equation formula plus a semantic interpretation for each of the symbols in the formula. Each symbol is identified with a quantity relevant to some class of domain objects in SIGMA’s knowledge base. The domain objects associated with the various equation symbols are constrained to obey specified relationships to each other. Consider Figure 4, which illustrates how Equation 1 of Figure 1 is represented internally within SIGMA. Equation 1 states that the number-density (n) of an atmospheric-parcel (i.e., the number of particles per volume of gaseous mixture in the atmospheric-parcel) is equal to the refractivity index (r) of the entire mixture divided by a weighted sum of the refractivity indices (r_g) of the individual gases within the mixture.

As shown in Figure 4, the semantics of this equation are represented in terms of the domain objects that the equation interrelates, namely the atmospheric-parcel, the homogeneous pure-gas constituents of the parcel, and the individual gases composing the constituents. The symbols "r" and "n" in the equation are linked to the refractivity and number-density quantities of an atmospheric-parcel. Moreover, "r" and "n" are associated with the same atmospheric parcel, not two distinct parcels. The subscript "g" in the equation is identified with the set of constituents associated with this same atmospheric-parcel, as well. The "constituents" slot in the atmospheric-parcel stores a pointer to each constituent within the parcel. The symbol "f_g" is linked to the mixing-fraction of a constituent, and stores the percentage of this constituent as a fraction of the total quantity of gaseous mixture within the atmospheric-parcel. The symbol "r_g" represents the refractivity attribute of the gas that is contained by a given constituent. Finally "L" refers to a physical-constant called Loschmidt’s Number.

In essence, this representation provides a set of domain

![Figure 3: Initial world state for Titan modeling domain](image-url)
constraints that must be satisfied for the equation to apply legitimately in a given domain situation. In particular, for a given generic equation to apply to a world state, its pattern of semantic equation constraints must match or unify with the pattern of constraints exhibited by the instances in the world state.

Note that aside from representing explicit scientific equations, SIGMA’s constraint network representation is also used to represent black-box FORTRAN subroutines. Subroutine inputs and outputs are typed and constrained using the same mechanisms as for equations, but execution is handled by making a foreign function call from LISP to FORTRAN.

3.2 Model-building interaction

At the start of the model-building interaction, the world state is initialized and the known data values are loaded into appropriate quantity slots of the domain object instances. The user begins by specifying a goal quantity to be computed by the model. In response, the system begins a simple depth-first backchaining process that forms a plan to compute the goal quantity. First, SIGMA locates all equations that could potentially be applied to compute the goal quantity. Effectively, this includes all equations in which the goal quantity can be algebraically manipulated onto the left hand side of the equation. Each such equation is matched against the initial world state to determine its applicability. If the requisite equation constraints hold in the initial world state, the equation is a candidate for application. In the event that there is more than one candidate, the user makes a selection. Once a candidate equation has been chosen, SIGMA checks whether the value of each input variable (i.e., each variable on the right hand side of the equation) is known. If so, the model is complete; if any value is unknown, the equation application process recurses and a data flow subgraph is constructed to compute the unknown quantity. If there are no equations available to compute a needed quantity, the system backtracks to the nearest choice point and selects an alternative equation to apply. (If that choice is ambiguous, the user intervenes to make the choice.) The result of this model-building interaction is a data flow model of the type illustrated in Figure 1.

The process of constructing the data flow model is considerably simplified by the specification of semantic constraints on the symbols in each equation. The constraints serve both to specify the conditions under which the equation applies and to guide the binding of symbols in the equation. For example, without constraints, the system would propose applying equations in situations where they semantically meaningless to apply. For example, suppose SIGMA does not know that the symbol “n” in Equation 1 must bind to a number-density quantity associated with an atmospheric-parcel object (see Figure 4). Without this knowledge, the system would attempt to suggest binding “n” to a number-density quantity associated with an atmospheric-parcel object. However, this would be semantically invalid; the equation can only be applied to calculate the number-density of a mixture of gases, and cannot be applied to a single constituent gas.

Aside from defining when equations apply, the constraints also describe how they apply. This relieves the user from the burden of exhaustively specifying how each symbol in each modeling equation binds to quantities in the world state. In effect, the equation constraints enable the system to infer part of the model specification automatically. Consider an example illustrated by Figure 5. Suppose the user specifies a number-density quantity (n) as a goal quantity to compute, and the system offers Equation #1 as a candidate for computing that quantity.

---

1 In the current version of SIGMA, the initial world state is fixed and cannot be altered by the user. We expect to relax this restriction in subsequent versions of the system.

2 The matching process can be complicated by the presence of repeated, isomorphic structures of objects in the initial world state. For example, the structure illustrated in Figure 3 is repeated for each altitude level sampled in the Voyager experiment. We handle such isomorphic structures by instantiating a representative structure, and operating on that structure as a surrogate for the others during specification acquisition. At execution time, the system implicitly iterates over all the members of a representative structure.
To unambiguously specify how to apply the equation, the user must only identify the atmospheric-parcel instance to which the equation should be applied. The system uses its representation of the generic equation (Figure 4) to determine which subsidiary objects are involved in equation application. To do this, the system follows pointers (see Figure 5) from atmospheric-parcel-32 to constituent-32a, and finally to hydrogen. So even though the user never explicitly specified the binding for the symbols n, r, L, g, f_g, or f_f in the equation, the system infers what quantities the user intends based on its semantic understanding of the equation as expressed by the constraints of the generic equation.

To extend the example further, now suppose the user has specified mass-density (ρ), rather than number-density (n) as the goal quantity. In this case, the computation of number-density appears as a subgoal necessary to compute input "n" in Equation #2 of Figure 5. Because the output from Equation #1 is an input to Equation #2, the symbols in those equations can be unified, and the bindings established for one equation can be propagated through to the other equation. For example: g is unified with i; f_i is unified with f_g. In other words, the user's initial bindings can be automatically propagated through the data flow graph via a constraint propagation mechanism. This relieves the user of the tedium of repeatedly specifying bindings when they can be easily inferred using appropriate domain knowledge. In addition, through the constraint propagation mechanism, SIGMA automatically enforces consistency throughout the model, and ensures that the user is constructing a semantically meaningful computation.

3.3 Executing the model

Once the model has been constructed, basic execution is relatively straightforward. Any equation with all of its input values known is placed on a queue to be fired. SIGMA's model-interpreter selects and executes equations one at a time from this queue. Once an equation is executed and a new quantity is computed, additional equations that require the new quantity may queue themselves for firing. Selection and execution of equations continues until the goal quantity has been computed.

Although basic model execution is straightforward, there are a number of factors that complicate the execution process:

- **Array-valued quantities**: Certain quantities in a data flow graph may represent arrays of data values, rather than scalars. For example, the symbol "r" in Figure 1 represents an array of refractivity values -- one for each altitude sampled in the Voyager experiment. If any of the input quantities for an equation is array-valued, the system performs an implicit iteration and computes an array-valued output.

- **Scientific units**: Before an equation is executed, the system must ensure that the input values are converted to a common, consistent set of scientific units. This conversion is performed by applying conversion factors associated with scientific unit descriptions in SIGMA's knowledge base.

- **Numerical integration**: In some cases, the system must solve a differential equation to produce an output value. For instance, the Hydrostatic Law (Equation #4 in Figure 1) is a differential equation that must be numerically integrated to solve for P. The specialized numerical integration method implemented in SIGMA requires the user to specify a boundary condition, as well as a method for approximating data values between the sampling points.

4 Related work

On the surface, SIGMA appears similar to a large class of data flow based visual programming environments that have been developed recently. These systems help users graphically construct software in a variety of application areas, including image processing and scientific
visualization [8, 15, 16, 17, 18], scientific instrument design [19], and simulation [20, 21]. In all of these cases, however, the software tool has fairly limited knowledge of the application domain. Although the tools enforce simple syntactic checks on the data flow graphs and perform some type-checking, none of these tools has a deep semantic understanding of what the data flow program is doing and whether the operations on the data make sense. As a result, it is possible with these tools to create a syntactically valid flow graph that is semantically meaningless to a domain specialist. In contrast, SIGMA assists the scientist during the model-building process and checks the model for consistency and coherency as it is being constructed. In particular, SIGMA’s domain knowledge assists the system in interpreting the user’s intentions and in constructing a semantically meaningful program.

SIGMA is closer in spirit to the body of work on applying knowledge-based techniques to various aspects of scientific and engineering computation [6, 9, 11, 13, 23, 25, 26]. We review some of the particularly relevant systems for comparison below.

The ECO project [11, 12] shares many goals with SIGMA. The ECO program was designed to enable ecologists to build simulation models using a special-purpose sorted logic designed to express ecosystem concepts. The models are conveyed in terms of the Systems Dynamics formalism [24], which uses “reservoirs” and “flows” to express differential equations. SIGMA and ECO differ primarily in terms of their representational power and their scope. SIGMA uses a general-purpose frame-based representation to express domain concepts, whereas ECO uses a more expressive, specialized logic formalism that we believe would be more difficult for scientists to use. SIGMA has been designed as a general model-building assistant, intended to assist scientist in a number of domains, whereas ECO is more narrowly scoped to ecological models and the Systems Dynamics formalism.

SIGMA is a domain-specific automatic programming system constructed at Schlumberger to assist in generating oil well log interpretation software. The system was designed for direct use by petroleum scientists, who were to use it to construct geological models expressed as a set of quantitative equations relating geological parameters of interest. Like SIGMA and ECO, 90 makes extensive use of scientific domain knowledge to aid in the model-building process. However, SIGMA’s equation representation appears to incorporate more domain knowledge and constraints than representation used in the 90 prototype. SIGMA’s equation representation itself has roots in the work on qualitative reasoning about physical systems [6].

In a similar vein, [7] describes the SINAPSE system under development at Schlumberger. SINAPSE helps scientist build mathematical modeling software used in the context of data interpretation tasks such as seismic interpretation. In particular, the system synthesizes finite-difference programs that implement partial differential equation models. SINAPSE can be compared with SIGMA on a variety of dimensions. In SIGMA, the emphasis is on specification acquisition, rather than code synthesis; in SINAPSE, the emphasis is reversed. The grainsize and nature of the scientific modeling problem addressed by the two systems is quite different. SIGMA provides assistance in specifying a complete model from end to end, whereas SINAPSE focuses on the subtask synthesizing efficient algorithms for modeling steps involving the solution of partial differential equations. Conceptually, SINAPSE could be called as a subroutine to synthesize code for one of the modeling steps specified using SIGMA. Finally, the “domain” of expertise for these two systems is different. SIGMA’s “domain knowledge” consists of knowledge about the quantities, equations, objects, and constraints necessary to model a particular physical system of interest to the scientist. SINAPSE’s “domain knowledge” consists primarily of knowledge about mathematics, rather than knowledge about the scientific domain under study.

Our motivation and approach to knowledge-intensive specification acquisition is similar in spirit to the philosophy behind ARIES [22]. ARIES helps requirements analysts construct specifications by providing defined concepts and terminology which are stored in a large knowledge base that employs representational mechanisms similar to those available in SIGMA’s representation language. The ARIES knowledge base stores knowledge about the air traffic control domain. In general, SIGMA can be viewed as an example of a new class of software tool that uses extensive domain knowledge to assist with various aspects of the software engineering task [9, 14].

5. Status and limitations

The current version of SIGMA focuses only on the acquisition of specifications for scientific modeling software, and ignores issues related to code synthesis, such as efficiency, numerical stability, database access, and platform dependencies. As such, the acquired specifications represent only a starting point in the end-to-end code generation process. Nevertheless, for applications where such concerns are minimal, the specifications produced by the system can be executed to produce useful results for the user in a fraction of the time it would take to produce running FORTRAN code. SIGMA is implemented in CommonLisp on a Sun SPARCStation 2.

At present, it is premature to evaluate the appropriateness and utility of the SIGMA framework for scientists. At present, there are two bottlenecks that prevent proper evaluation. First, the current interface is a simple text-based interface, and communication with the
of domain-specific knowledge.

6 Summary and conclusions

In this paper, we described a knowledge-based software development environment for scientific modeling. The SIGMA system provides scientist-users with a high-level data flow language for specifying scientific models. This language is intended to be both easier to use and easier to understand than low-level scientific code. Thus the language and our software tool should facilitate sharing among users. SIGMA makes use of an extensive store of background domain knowledge to ease the specification acquisition process and to maintain the consistency of an evolving specification. In addition, SIGMA uses its background knowledge to aid in the execution of specified models. The availability of background knowledge enables the system to automatically infer portions of the scientific model specification that normally would be tediously specified by the user. In our experience, this functionality is absolutely critical to widespread acceptance of this kind of software engineering tool by scientists, and represents significant "value-added" over FORTRAN, which is still the standard-bearer in the scientific computation community today.

Acknowledgments

This research was co-funded by NASA's Office of Space Science and Applications, and the Office of Aeronautics and Space Technology. We wish to thank the other members of the SIGMA Project for their intellectual contribution to the ideas expressed in this paper. These members include Michael Sims, David Thompson, Christopher McKay, Jennifer Dungan, Yaron Gold, and Caitlin Griffith. Special thanks to Pandu Nayak for providing the RML representation language, for maintaining and augmenting the language to suit our needs, and for tutoring us in its use. Mike Lowry provided helpful comments on an earlier draft of this paper, as did anonymous reviewers.

200
References


[15] Khoros/Cantata software product, Khoros Consortium, EECE Department, University of New Mexico, Albuquerque, NM.

[16] Iconicode and IDF software products, Iconicon, Palo Alto, CA.


[18] apE 2.0 software product, Ohio Supercomputer Center, Columbus, OH.

[19] LabVIEW software product, National Instruments, Austin, TX.

[20] STELLA and IThink software products, High Performance Systems, Lynne, NH.


