Knowledge-Intensive Software Design Systems: *Can too much knowledge be a burden?*

Richard M. Keller

Sterling Software
NASA Ames Research Center - Artificial Intelligence Research Branch
Mail Stop 269-2, Moffett Field, CA 94035-1000

(415) 604-3388 (Phone); 604-3594 (FAX); Keller@ptolemy.arc.nasa.gov

Abstract

While acknowledging the considerable benefits of domain-specific, knowledge-intensive approaches to automated software engineering, it is prudent to carefully examine the costs of such approaches, as well. In adding domain knowledge to a system, a developer makes a commitment to understanding, representing, maintaining, and communicating that knowledge. This substantial overhead is not generally associated with domain-independent approaches. In this paper, I examine the downside of incorporating additional knowledge, and illustrate with examples based on our experience building the SIGMA system. I also offer some guidelines for developers building domain-specific systems.

1. Introduction

One of the long-prevailing tenets of artificial intelligence research is that "knowledge is power" -- the more knowledge made available to a system, the better. The knowledge-based software engineering (KBSE) community, as evidenced by its self-designation, embraces this philosophy no less than other disciplines within AI. Traditionally, the knowledge represented and used by practitioners of KBSE has been knowledge about the programming discipline, itself. Increasingly, however, researchers are recognizing the utility of representing and using knowledge about the target programming domain (e.g., business, manufacturing, science, telecommunications, engineering, etc.) to facilitate automation of various facets of the software engineering process [1,2,3]. In fact, the seductive "knowledge is power" maxim has even found a receptive audience in the mainstream software engineering community, where several workshops on the topic of "Domain Modeling" have been held over the past few years [4].

The migration toward domain-specific systems comes as no great surprise. Despite progress in developing general-purpose methods for automated software engineering [5], the practical application of these techniques has met with limited success. In some cases, these methods have failed to scale up appropriately; in other cases, the methods have proven too mathematically-sophisticated to appeal widely to the practicing community of software engineers. However, by incorporating additional domain knowledge and constraints, it becomes possible to specialize and simplify these methods to a point where they are more tractable and less daunting to apply. While acknowledging the considerable benefits of domain-specific, knowledge-intensive approaches to automated software engineering, it is prudent to carefully examine the costs of such approaches, as well. In adding domain knowledge to a system, a developer makes a commitment to understanding, representing, maintaining, and communicating that knowledge. This substantial overhead is not generally associated with domain-independent approaches. In this paper, I examine the downside of incorporating additional knowledge, and question whether adding knowledge introduces as many new problems as it solves.

Over the past several years, I have been involved in the development of a domain-specific software design system for scientific modeling. To ground my remarks, I will briefly describe this system and its knowledge requirements. Then I will describe some of the additional burden placed on the developers as a result of the knowledge-intensive nature of this system. Finally, I will attempt to generalize from our experience and present some guidelines and caveats for others developing domain-specific KBSE systems.

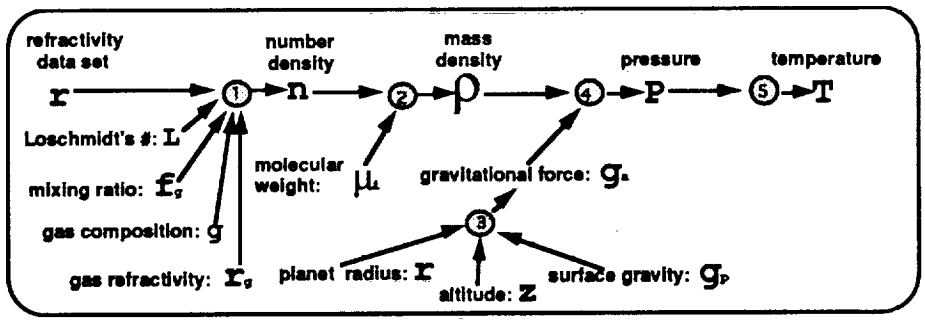
2. SIGMA : A knowledge-based scientific software environment

The goal of the SIGMA project [6] 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. Although computer models play a crucial role in the conduct of science today, scientists lack adequate software engineering tools to facilitate the construction, maintenance, and reuse of modeling software. Usually, scientific models are implemented using a general-purpose computer programming language, such as FORTRAN. Because this type of general-purpose language is not specifically customized for scientific modeling problems, the scientist is forced to translate scientific constructs into general-purpose programming constructs. This manual translation process can be very complicated, labor-intensive, and error-prone. Furthermore, the translation process obfuscates the original scientific intent behind the model, and buries important assumptions in the program code that should remain explicit. The resulting software is typically complex, idiosyncratic, and difficult for anyone but the primary scientific author to understand.

We are building a knowledge-based software environment that makes it easier for scientists to construct, modify, and share scientific models. The SIGMA (Scientists' Intelligent Graphical Modeling Assistant) system functions as an intelligent assistant to the scientist. Rather than construct models using a conventional programming language, scientists will be able to use SIGMA's graphical interface to "program" visually using a more natural high-level graphical data flow modeling language. The terms in this modeling language denote scientific concepts (e.g., physical quantities, scientific equations, and datasets) rather than general programming concepts (e.g., arrays, loops, counters). The scientist-user interacts with the system to construct a syntactically and semantically valid data flow graph, such as the one illustrated in Figure 1. In this graph, the lettered nodes represent scientific quantities, such as temperature, pressure, and density. These quantities are input to scientific equations (depicted by numbered nodes in Figure 1) which calculate output quantities.

The data flow graph in Figure 1 represents part of a planetary atmospheric model developed at NASA Ames Research Center [7]. The model computes the temperature (T) at some altitude point above a planetary surface based on input data (r) measuring the extent to which a radio signal refracts upon penetrating the atmospheric gases at that altitude.

Although visually simple, the graph masks a number of non-trivial technical problems that must be addressed to actually execute the corresponding program. For example, the input refractivity value is a vector quantity, not a scalar, so there is an implicit iteration being performed. Note also that Equation # 4 is a differential equation that must be numerically integrated to solve for P. In addition, the scientific units specified for the various inputs to an equation may not be compatible and must be



$$① \; n = \frac{r}{\Sigma_g f_g \frac{r_g}{L}} \qquad ② \; \rho = n \Sigma_i f_i \frac{\mu_i}{N0} \qquad ③ \; g_s = g_p \left(\frac{r}{r+z}\right)^2 \qquad ④ \; \frac{dP}{dz} = -\rho g \qquad ⑤ \; n = \frac{P}{kT}$$

Figure 1: Data flow graph representing a portion of a planetary atmospheric model. Letters represent physical quantities. Numbered circles correspond to equation application nodes.

converted to a common unit system before that equation can be applied. SIGMA's interpreter handles these details automatically for the user.

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 (Khoros/Cantata [8], Iconicode/IDF [9], AVS [10], apE [11]), scientific instrument design (LabVIEW [12]), and simulation (STELLA/IThink [13], Extend [14]). 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 ϕ_0 [15]. ϕ_0 is a domain-specific automatic programming system constructed to assist in generating oil well log interpretation software. The system was designed for direct use by petroleum scientists, who would use it to construct geological models expressed as a set of quantitative equations relating geological parameters of interest. Like SIGMA, ϕ_0 makes extensive use of scientific domain knowledge to aid in the program synthesis process. The next section describes SIGMA's domain knowledge.

3. SIGMA's Domain Knowledge

SIGMA's domain knowledge is represented and stored in a hierarchically-structured, frame-based knowledge base of over 500 concepts which contain 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.

SIGMA's knowledge can be partitioned into four categories:

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

2. **Area-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.

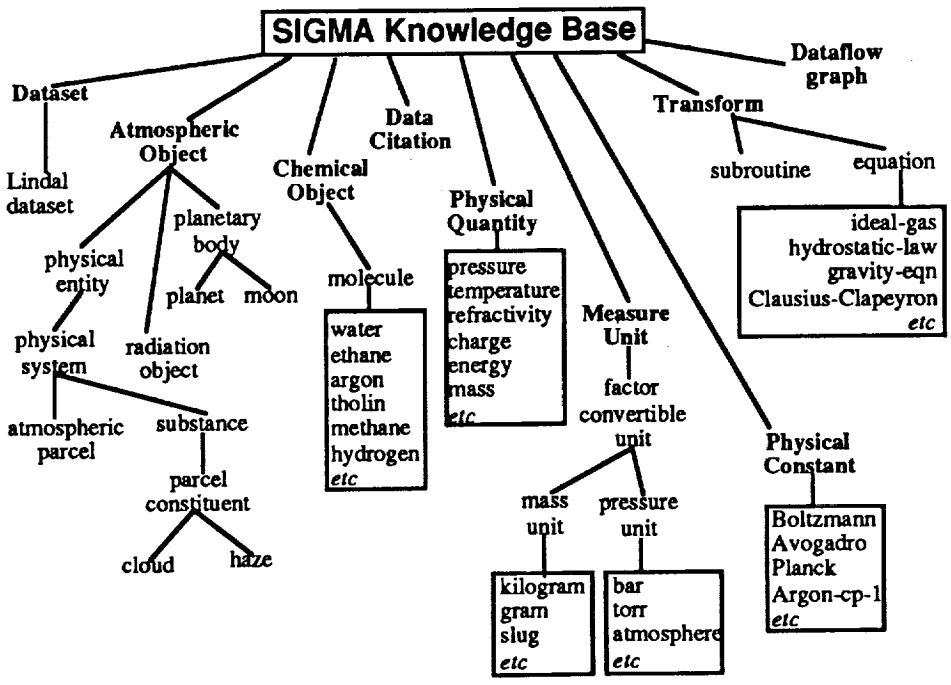


Figure 2: Overview of SIGMA's knowledge base

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 data flow interpreter.)

Although a detailed discussion of SIGMA's knowledge base and representational structures is outside the scope of this paper, I will briefly describe one of the key elements: SIGMA's equation representation.

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 an attribute of 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 among each other. Consider Figure 3, which illustrates how Equation 1 of Figure 1 is represented internally within SIGMA. Equation 1 states that the number density (n) of a gas mixture (i.e., the number of particles per volume of mixture) 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 3, the semantics of this equation are represented in terms of the domain objects that the equation interelates, namely the gas mixture (called an atmospheric-parcel), the homogeneous pure-gas subcomponents of the mixture (called constituents), and the individual gases that are included in the mixture. The symbols "r" and "n" in the equation are linked to the refractivity and number-density attributes of the same atmospheric-parcel. The subscript "g" is identified with the constituents attribute of that same atmospheric-parcel. The constituents attribute stores a pointer to each constituent within the atmospheric-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 gas within the atmospheric-parcel. The symbol "r_g" represents the refractivity attribute of a gas that is contained by the constituent. Finally "L" refers to a physical-constant called Loschmidt's Number.

In essence, this representation provides a set of domain constraints that must be satisfied for the equation to apply legitimately in a given domain situation. As a scientist builds up a data flow graph such as the one in Figure 1, he or she is unknowingly constructing an invisible constraint network of domain objects and relations similar to the one illustrated in Figure 3. This constraint network provides a sound semantic interpretation for the graph.

4. SIGMA's Knowledge Burden

The rationale behind our decision to invest considerable time and energy into representing domain knowledge for SIGMA is simple and, we believe, compelling: How can a machine interact intelligently and synergistically with a scientist to create modeling software if the machine has no understanding of the scientific problem under study? Without this shared understanding, SIGMA would have to rely on user guidance for many of the functions it now performs automatically. Our users have expressed an impatience with systems that need to be "spoon-fed"; given an option, they would rather drop down into FORTRAN and code the model themselves! Our only alternative, it seems, is the knowledge-intensive route.

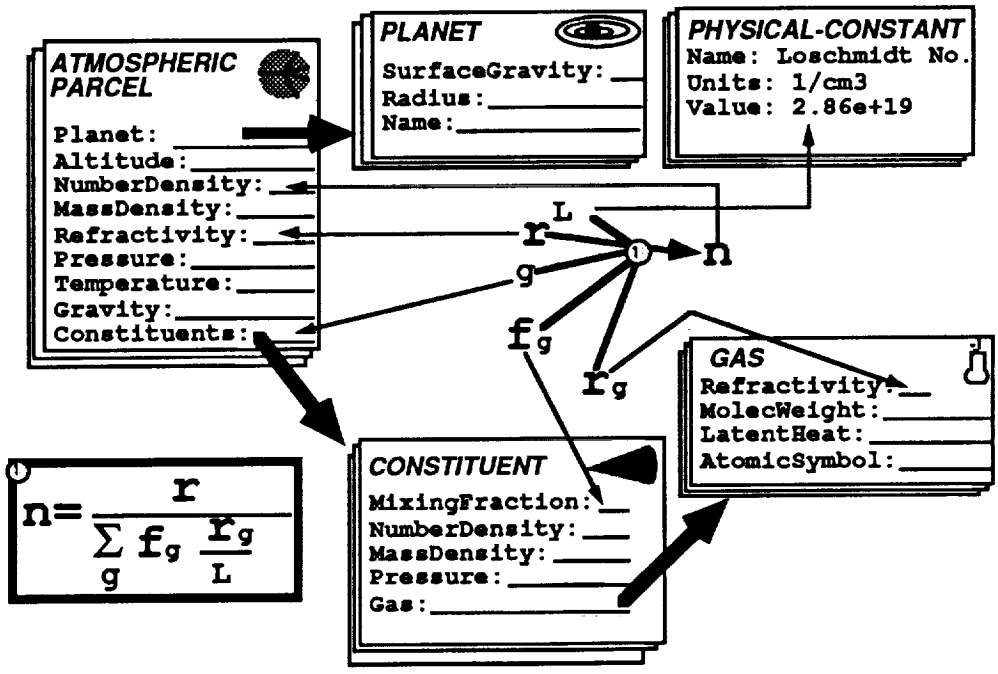


Figure 3: Representation for Equation 1 in Figure 1.

The Catch-22 in this situation is that the addition of domain knowledge imposes burdens on the developer, maintainer, and users of the interactive software design system:

- **The Comprehension Burden:** System developers must analyze and understand the application domain and the class of problems to be solved.

  Our experience with SIGMA is that a significant amount of time (several person-months of effort) is required to sufficiently understand the scientific modeling problems presented by our collaborators in planetary and ecosystem sciences. Of course the difficulty is a function of many variables, including the developer's prior background knowledge and experience in the application domain, the caliber of expert advice and guidance, the complexity of the scientific modeling problem, etc.

- **The Representation Burden:** Developers must design suitable representations to capture the knowledge.

  In our experience, the problem of representing domain knowledge is a significant modeling problem in itself. Within SIGMA, we have identified a need for representing quantities, quantitative and qualitative relationships, part-whole and subsumption relationships, temporal and spatial relationships, modeling assumptions, and other difficult representational constructs. A comprehensive treatment of all of these issues is beyond the scope of any single project. (However, see [16] for an ambitious effort in this vein.)

- **The Maintenance Burden:** System maintainers or users must add new knowledge, update old knowledge as it becomes outdated, and generally maintain the integrity of the knowledge base.

  For example, novice and intermediate SIGMA users will want to enter new equations and new physical quantities into the system. Sophisticated SIGMA users may wish to modify the original domain theory that was captured and encoded as a by-product of discussions with our expert collaborators. In fact, the domain theory (i.e., the domain objects, attributes, and relations) is as much a part of the scientist's model as the equations. Because the equations are intimately linked to the underlying domain theory (as discussed in Section 3), entering a new equation is complicated, and modifying the domain theory has wide-ranging implications. As a result, the current version of SIGMA does not permit users to modify the domain theory.

- **The Communication Burden:** Developers must implement tools and techniques that adequately convey the system's knowledge to the user, and vice versa.

  Consider once again SIGMA's equation representation. It is non-trivial to convey this type of a representation scheme to a naive user without exposure to knowledge-based or object-oriented techniques. Building an adequate user-friendly editor for SIGMA will be a challenging (and no doubt time consuming) task. Navigating and editing the concepts in the knowledge base pose similar difficulties.

Although these problems are significant, most of them are pose no greater or lesser challenge than those faced by developers, maintainers, and users of *any* sophisticated knowledge-based system. Software engineering, after all, is just another application area for knowledge-based techniques.

5. Easing the Burden

Despite the extra effort involved, and the new problems introduced, I still believe it is worth the effort to incorporate domain knowledge as an integral part of an automated software engineering environment. I believe the newly-introduced problems are challenging, but tractable. And without incorporating additional knowledge, I see no way to provide more intelligent and domain-sensitive tools to practitioning software engineers. In this spirit of pragmatism, I offer the following recommendations to those building knowledge-intensive, domain-specific tools:

- **Generality:** Keep the knowledge base and the representations general, without going overboard. This will facilitate entry of new information into the knowledge base, and encourage reuse of existing knowledge and representational constructs in new, similar domains.

- **Stability:** Choose an application for which the domain knowledge is relatively stable. This will minimize the maintenance burden.

- **Scope:** Choose an application for which knowledge is well-circumscribed, yet broad enough to make the endeavor worth your effort. If the knowledge can be reused in other applications, the development costs can be amortized over a shorter period of time.

- **Content:** Choose an application for which the domain theory is well-understood and commonly accepted. This will simplify the process of building an acceptable domain theory and reduce maintenance and communication costs.

- **Terminology:** Use vocabulary that is as familiar as possible to users. This will ease the communication burden.

- **Grainsize:** Avoid modeling phenomena in more detail than necessary for the task -- unless warranted due to generality and subsequent reusability.

Of course the developers of software systems do not always have control over the selection of an application domain. In this case, the above recommendations can be used to evaluate the suitability of domain-specific approaches with respect to a particular domain.

6. Conclusion

Yes, I still believe in the "knowledge is power" axiom. But more than ever, I feel it is important to heed its most-overlooked corollary: "There is no such thing as a free lunch". Caveat emptor!

Acknowledgments

Thanks to the SIGMA group, and especially to Michal Rimon, who implemented the current version of our system. Thanks also to Pandu Nayak who provided us with his RML representation language.

References

[1] D.Barstow, "Domain-Specific Automatic Programming", IEEE Transactions on Software Engineering, Vol. SE-11, No. 11, pp. 1321-1336, Nov. 1985.

[2] E.Kant, F.Daube, W.MacGregor, and J.Wald, "Scientific Programming by Automated Synthesis", in *Automating Software Design*, pp. 169-206, M.R.Lowry and R.D.McCartney (eds.), AAAI Press, Menlo Park, CA, 1991.

[3] D.Setliff, "On the Automatic Selection of Data Structure and Algorithms",in *Automating Software Design*, pp. 207-226, M.R.Lowry and R.D.McCartney (eds.), AAAI Press, Menlo Park, CA, 1991.

[4] N.Iscoe, "Domain Modeling -- Evolving Research",*Proc. Sixth Annual Knowledge-Based Software Engineering Conference*, pp. 234-236, IEEE Computer Society Press, Los Alamitos, CA, 1991.

[5] M.R.Lowry and R.Duran, "Knowledge-Based Software Engineering", chapter in *Handbook of Artificial Intelligence, Vol. IV*, A.Barr and P.Cohen (eds.), Addison-Wesley, New York, 1989.

[6] R.M.Keller and M.Rimon, "A Knowledge-based Software Development Environment for Scientific Model-building", AI Research Branch technical report #FIA-92-12, NASA Ames Research Center, Moffett Field, CA, forthcoming July 1992.

[7] C.P.McKay, J.B.Pollack, and R.Courtin, "The Thermal Structure of Titan's Atmosphere", *Icarus*, vol. 80, pp. 23-53, 1989.

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

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

[10] AVS software product, Stardent Computer, Inc., Sunnyvale, CA.

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

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

[13] STELLA and IThink software products, High Performance Systems, Lyme, NH.

[14] Extend software product, Imagine That, Inc., San Jose, CA.

[15] D. Barstow, R. Duffey, S. Smoliar, and S. Vestal, "An Overview of Φnix", in *Proc. National Conference on Artificial Intelligence* (AAAI-82), pp.367-369, Pittsburgh, PA, August 1982.

[16] R.V.Guha and D.B.Lenat, "Cyc: A Mid-Term Report", *AI Magazine*, 11(3), 1990.