A Domain-Specific Modeling Language for Scientific Data Composition and Interoperability

Hyun Cho
University of Alabama at Birmingham
Department of Computer and Information Science
Birmingham, AL, USA 35294
robusta@uab.edu

Jeff Gray
University of Alabama
Department of Computer Science
Tuscaloosa, AL, USA 35487
gray@cs.ua.edu

ABSTRACT
Domain-Specific Modeling Languages (DSMLs) can offer assistance to domain experts, who may not be computer scientists, by providing notations and semantic constructs that align with abstractions from a particular domain. In this paper, we describe our design and application of a DSML in the area of data composition and interoperability. In particular, we introduce our recent effort to design a DSML to assist with interoperability issues across scientific software applications (e.g., composing scientific data in different file structures and integrating scientific data with data gathering devices). Currently, several different scientific data file specifications have been proposed (e.g., CID, netCDF, and HDF). Each file specification is optimized to manage a specific data type efficiently. Thus, each file specification has evolved with slightly different notions and implementation technologies. These differences led to the need for an environment that provides interoperability among the different specification formats. In this paper, we introduce our framework, supported by a DSML, that provides functionality to visually model the data composition and integration concepts independent from a particular data file specification.

Categories and Subject Descriptors
D.2.12 [**Interoperability**]: Data mapping – *abstract data types, polymorphism, control structures.*

General Terms
Management, Design

Keywords
Domain-Specific Modeling Language (DSML), File Format, Data Composition, Data Integration, Verification, Metamodeling.

1. Introduction
According to [7], a Domain-Specific Modeling Language (DSML) uses a metamodel to "express the definition of a modeling language that represents the key abstractions and intentions of an expert in a particular domain." DSMLs are more concise than General-Purpose Modeling Languages (GPMLs), such as UML, and can express the semantics of a specific domain very clearly with visual notations that map to concepts in the domain. In addition, model transformations can synthesize a model expressed in a DSML into other artifacts (e.g., source code or some other representation) using transformation and generative techniques [3]. Automated code generation from a model eliminates the tedious and mundane effort of manually converting a model into a source code implementation. Such automation can also minimize the probability of error injection often caused by manual adaptation. DSMLs have been used in many domains, such as multimedia [8][10], financial products [1], telephone switching systems [9][12], protocols [6][14], operating systems [15], and robot languages [4].

In this paper, we describe our effort to design and implement a DSML to manage scientific data files that are used to store a high volume of data (e.g., sensor-based networks with terabytes of data representing data types such as images, graphs, tables, and numbers). Scientific data files have several unique characteristics. For example, they often include metadata to describe the organization and type of contents represented in the file. Such file formats also have their own archiving and restoring mechanism to backup content efficiently. In addition, each scientific data specification provides an API that supports several different programming languages, which allows users to develop their own applications. Due to these different characteristics, managing scientific data files presents several challenges: 1) visualizing the contents of the file, 2) transforming the contents from one file format to another, 3) managing the evolution of APIs while supporting backward compatibility, and 4) maintaining the stability of user applications during API evolution.

To resolve these issues, we designed a DSML and a supporting framework that provides functions to model data composition and integration visually and independently from the data file specification format. The rest of the paper is organized as follows. Section 2 introduces several different scientific file formats and Section 3 presents the applications of a DSML for scientific data management. Section 4 summarizes the results of applying this approach.

2. Scientific Data File Formats
Many research areas (e.g., earth science, biology, physics, and medicine) need to integrate research data from various sources and different types (e.g., graphic and/or numerical data). Such integrated data is used to analyze the causes and effects of a certain phenomena or to provide an experimental proof. With the advent of low-cost high-end devices (e.g., high resolution imaging devices and fast data collecting devices), and the popularization of those devices, the growth rate of data production has experienced phenomenal increase in need and capacity. Thus, the existing file formats are no longer appropriate to organize, archive, and manipulate data in the face of enormous growth of scientific data

in size and complexity. To tackle these issues, many file formats have been proposed. For example, CIF (Crystallographic Information Framework) [16] was proposed in 1991 and has evolved under the support of the IUCr Working party. CIF is a free-format archive file format and designed for the electronic exchange of crystallographic data between individual laboratories, journals and databases. NetCDF (Network Common Data Form) [17], which is maintained by University Corporation for Atmospheric Research, originally was designed to manage real-time weather data but now it has been extended to support the creation, access, and sharing of array-oriented scientific data in a machine-independent manner. Recently, the HDF group proposed another file format, called HDF (Hierarchical Data Format) [18], to manage scientific data in a more generic way by supporting an unlimited variety of data types, providing flexible and efficient I/O, and allowing high volume and complex data.

However, these scientific file formats, which organize, archive, and manipulate high volume and complex data, have several common characteristics, as follows:

- **Self-descriptive**: Unlike usual file formats, such as image files (e.g., *.jpeg, *.bmp), data files (e.g., *.dat), and executable files (e.g., *.exe, *.com), the popular scientific data file formats contain metadata to inform what kind of data types are stored in the file and how they are organized.
- **Directly accessible**: The volume of scientific data files are usually high and scientific files can contain complex and different data types, such as images, tables, and array-oriented data. Thus, unlike traditional file formats, scientific file formats allow arbitrary access to data directly.
- **Concurrently accessible**: Most scientific file formats are designed in a thread-safe manner so that multiple processes can read and modify contents of the file simultaneously. This concurrency is essential for high performance computing for increased performance in accessing content.
- **Archivable**: Scientific file formats are designed to support a high volume of data and to maintain complex and heterogeneous data. Most scientific data file formats provide their own archiving scheme to perform backup and restore capabilities.

In addition, to encourage the use of a specific scientific file format, software packages are often freely available. Some tools assist in manipulating and displaying the contents of a file and other tools help to compute complex mathematical data. However, such tools also present a new set of challenges:

- **Representing the organization of the file structure**: Even though software exists to manage and support each scientific file format, most tools provide a textual user interface or execute under a command-line environment. In addition, users need to have a somewhat deep understanding of the file specification to use these tools efficiently. Moreover, users need to know how the API works in order to design and implement their own applications using the format.
- **Managing the evolution of APIs**: Most scientific file formats provide an API that supports several programming languages (e.g., C/C++, FORTRAN, and Java). However, changes to an API are inevitable, because APIs need to accommodate new requirements in the improved understanding of the constraints of data, or to interface with new types of data gathering devices that provide high performance and better resolution. If these changes are introduced to the file specification, some APIs need to be redesigned and re-implemented in accordance with the revisions of the file specification. To automate API evolution, some researchers introduced control-flow analysis and temporal logic-based matching techniques [2][5][13]. But these approaches work based on source code or test cases rather than the specification of a file format. Thus, a systematic method is needed to verify which APIs are modified with the changes of the file specification, because the file specification and APIs are managed under different languages and tools.
- **Maintain stability of existing applications**: Because APIs often evolve, each user application may also need to be changed. This change leads to the need to test the correctness of a user application repeatedly to guarantee that the application is compatible over new APIs and libraries. Thus, maintaining the stability of a user application amid evolution of the file format API requires an investment of additional effort.
- **Limited support for data integration**: Because each scientific file format has different characteristics and uses different technologies, exchanging data between different scientific file formats may not be possible by mapping contents one-by-one. Users may need to develop a complex data conversion tool, which requires understanding of both source and target file specifications and APIs. Moreover, users need to develop new applications or modify existing applications to integrate data with emerging data gathering devices.

In the following section, we describe the advantage of using a DSML and how a DSML can resolve the issues just mentioned.

3. A DSML for File Format Interoperability

DSMLs provide a concise set of visual notations that denote a specific set of domain abstractions, which fits well to the specific problem space while also assisting with the communication between stakeholders. In addition, DSMLs can help with the implementation of a new application by adopting generative programming concepts (i.e., describing the essence of the problem in the DSML and then generating lower level executable code from a model). Using a DSML requires the definition of several models that are inter-related across three different layers: the model, metamodel, and meta-metamodel [11]. Each model defines a representation structure and a global typing system that is used by the layer beneath it. For example, each model should conform to the structure and types of the metamodel. These different model layers provide several important key advantages of using a DSML over a GPML (e.g., UML), such that domain experts can define and build a modeling tool for their own purpose that fits a specific domain of expertise.

3.1 Building Domain-Specific Models

A key advantage of a DSML over a GPML is that DSMLs can have flexibility to define notations that satisfy a specific problem space. This advantage helps engineers shorten their learning time to use the tool and helps to prevent miscommunications between stakeholders.

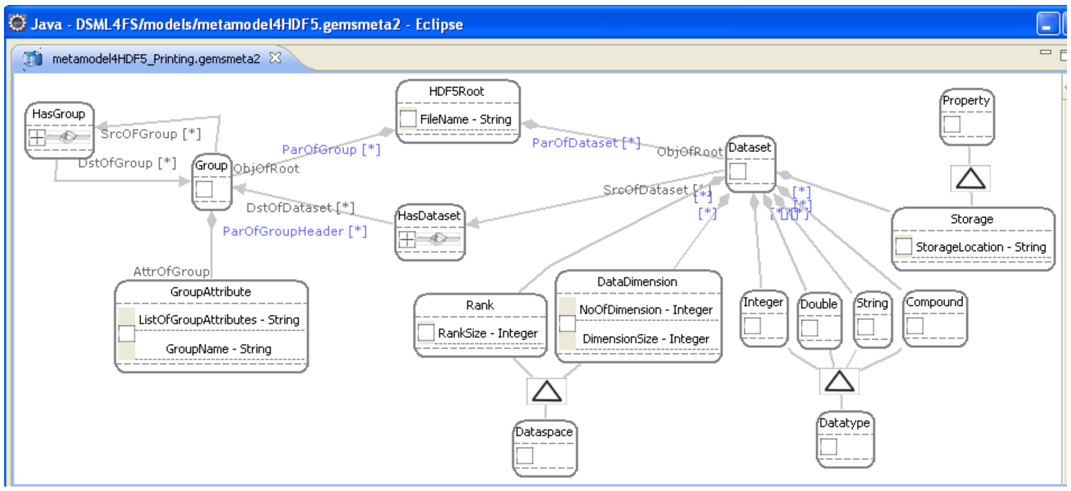

Figure 1. Metamodel for HDF

A DSML for enabling interoperability across scientific data files can be built with the following three steps: domain analysis, metamodel definition, and DSML tool construction. Domain analysis is the very first step and compulsory activity to understand the requirements needed by end-users. To build a DSML for scientific data file management, domain analysis needs to focus on analyzing the data structures (e.g., what type of data will be maintained, and how is the data organized). In metamodel definition, constructs identified during the domain analysis are modeled as a metamodel, which is then used to define the language and tool support. The metamodel shown in Figure 1 models the fact that HDF5 supports primitive data types (e.g., integer, double, and string) as well as compound data types that are similar to a `struct` in C and composed from primitive data types. Each data type in HDF5 is stored in a Dataset and each Dataset can exist either in an independent set or under the nested groups. Properties of each Dataset and Group are specified using an attribute and property. From this metamodel, a modeling environment for this DSML can be generated automatically. The idea has been implemented using an Eclipse plug-in, the Generic Eclipse Modeling System (GEMS) [1], which supports the automatic generation of a visual DSML through a meta-programmable modeling environment.

3.2 Scientific Data Interoperability

Figure 2 presents an overview of our framework for scientific data composition and integration. The framework has been designed with a layered architecture and is divided into three layers; Physical layer, API layer, and DSML layer.

The Physical layer maintains physical scientific data files and scientific devices that gather or present actual scientific data. The API layer interconnects the DSML layer with the Physical layer using APIs and libraries. The API layer provides APIs that can be used to develop new applications, which can create, modify, view, and share scientific data. Additionally, the API layer includes an API abstraction layer, which represents the collection of abstracted APIs that are defined from a generalization of all common scientific data APIs. For example, in netCDF a call to a specific API function can be used to create and open a file (e.g., the function `int nc_create (const char* path, int cmode, int *ncidp)`). Similarly, HDF5 provides a function for this same purpose, but as a constructor (e.g., `H5File (const char *name, unsigned int flags)`). Because both file specifications have evolved with different background and technologies, APIs cannot be matched to each other for the same functionality. Therefore, the API abstraction layer provides the same functionality independently from the specific scientific file API. As a result, file creation and opening are redefined as `createFile(const char *path, FileCreationProperty fileCreationProperty)` in the API abstraction layer.

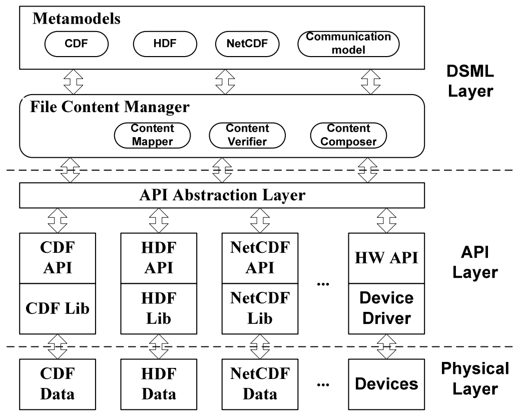

Figure 2. Framework for Scientific Data File Management

The API abstraction layer can help to evolve a DSML and user applications from the evolution of APIs. The DSML layer, which plays an important role in the framework, consists of two parts: Metamodel and File Content Manager (FCM). The FCM maintains the metamodel of each scientific file. As mentioned in Section 3.1, each metamodel defines constructs for the scientific file specification. Unlike other metamodels, the Communication metamodel is used to define a DSML that informs the communication method used in the data gathering devices. The Communication metamodel may include information, such as category of device, data gathering or representation method, and communication protocols. FCM is mainly responsible for creating the graphical model for composing and integrating data. FCM is comprised of three sub-modules: Content Composer (CC), Content Verifier (CV), and Content Mapper (CM). CC and CM provide integrated graphical data composition and an integration environment. The main functionality of CC is to design the structure of the scientific data file using the graphical notations that are defined in the metamodel. CM will also provide a graphical user interface so that users can specify how to integrate data either with another scientific data file or with data gathering devices. After the file structure is composed with CC, the CV verifies the correctness of the file structure. First, CV verifies that the model conforms to its metamodel and then it verifies the model against the API to determine whether the model can be mapped with appropriate APIs. If the model passes the verification process, CV can generate code that can access the content directly. CV is also useful for checking the mismatched evolution between a metamodel and its APIs. For example, if APIs are not evolved in accordance with the evolution of a metamodel, the CV can return exceptions because there is no APIs to match the newly evolved metamodel. CV can be used to regenerate the file structure model by reading the metadata of the physical scientific data file. This function is especially helpful when composing data integration rules. Finally, CM builds the content mapping rules when transforming contents from one file format to another.

4. Results and Future Works

Currently, the framework for data composition and integration is designed and the feasibility of our approach has been verified with primitive data types of each file specification. As shown in Figure 2, the framework has been designed with the layered architecture, which consists of the DSML layer, API layer, and Physical layer. From our experience with this project, we found that:

- The abilities of a DSML, especially abstraction and graphical representation, can help to build a graphical tool to compose and support interoperability across scientific file structures.
- Adoption of the layered architecture in the framework can help to keep the independence of each layer separate from the other layers.
- Both the API abstraction layer and the layered architecture are essential to develop and maintain user applications independently from the evolution of specific scientific file APIs.

We are currently working on the creation of metamodels that include full specification of each scientific file. We also are categorizing APIs in accordance to their intended use (e.g., file creation and data read/write) for the API abstraction layer. Additional future work includes more support for managing the evolution of APIs and integrating data obtained from data gathering devices.

ACKNOWLEDGEMENT

This work was supported by an NSF CAREER award (0643725).

REFERENCES

[1] Arnold, B.R.T., van Deursen, A., and Res, M. 1995. “An algebraic specification of a language describing financial products,” *In ICSE Workshop on Formal Methods Application in Software Engineering*, pp. 6-13, Seattle, WA, April 1995.

[2] Bartolomei, T. T., Czarnecki, K., Lämmel, R., and van der Storm, T. 2009. “Study of an API migration for two XML APIs,” In *Proceedings of Software Language Engineering*, pp. 46-65, Denver, CO, October 2009.

[3] Barstow, D. R. 1985. “Domain-specific automatic programming,” *IEEE Transactions on Software Engineering* vol. 11, issue 11, pp.1321-1336, November 1985.

[4] Bjarnason, E., “Applab: a laboratory for application languages,” *In Nordic Workshop on Programming Environment Research*, pp. 99-104, Aalborg, Denmark, May 1996.

[5] Brunel, J., Doligez, D., Hansen, R. R., Lawall, J. L., and Muller, G. 2009. “A foundation for flow-based program matching: using temporal logic and model checking,” *In POPL '09: Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages*, pp. 114-126, Savannah, GA, January 2009.

[6] Chandra, S. and Larus, J., “Experience with a language for writing coherence protocols,” *In Proceedings of the Conference on Domain-Specific Languages*, pp. 51-66, Santa Barbara, CA, October 1997.

[7] Gray, J., Tolvanen, J-P., Kelly, S., Gokhale, A., Neema, S., and Sprinkle, J., “Domain-Specific Modeling,” *Handbook of Dynamic System Modeling*, (Paul Fishwick, ed.), CRC Press, ISBN: 1584885653, 2007, Chapter 7, pp. 7-1 through 7-20.

[8] Elliott, C. 1997. “Modeling interactive 3D and multimedia animation with an embedded language,” *In Proceedings of the Conference on Domain-Specific Languages*, pp. 285-296, Santa Barbara, CA, October 1997.

[9] Gupta, N. K., Jagadeesan, L. J., Koutsofios, E. E., and Weiss, D. M., 1997. “Auditdraw: Generating audits the fast way,” *In Proceedings of the Third IEEE Symposium on Requirements Engineering*, pp. 188-197, Annapolis, MD, January 1997.

[10] Kamin, S. N. and Hyatt, D. 1997. “A special-purpose language for picture-drawing,” *In Proceedings of the Conference on Domain-Specific Languages*, pp. 297-312, Santa Barbara, CA, October 1997.

[11] Kurtev, I., Bézivin, J., Jouault, F., and Valduriez, P. 2006. “Model-based DSL frameworks,” *In Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications*, pp. 602-616, Portland, OR, October 2006.

[12] Ladd, D. A. and Ramming, J. C. 1994. “Two Application languages in software production,” *In Proceedings of the USENIX 1994 Very High Level Languages Symposium*, Santa Fe, NM, October 1994.

[13] Padioleau, Y., Lawall, J., Hansen, R. R., and Muller, G. 2008. “Documenting and automating collateral evolutions in linux device drivers,” *SIGOPS Operating System Review*, 42(4):247-260.

[14] Thibault, S., Consel, C., Muller, G. 1998. “Safe and efficient active network programming,” *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, pp. 135-143, West Lafayette. IN, October 1998.

[15] C. Pu, A. Black, C. Cowan, J. Walpole, and C. Consel. 1997. “Microlanguages for operating system specialization,” *In Proceedings of the Workshop on Domain-Specific Languages*, pp. 49-57, Paris, France, January 1997.

[16] http://www.iucr.org/resources/cif/spec

[17] http://www.unidata.ucar.edu/software/netcdf/

[18] http://www.hdfgroup.org/

[19] Generic Eclipse Modeling System (GEMS), http://www.eclipse.org/gmt/gems/