

Modeling Heterogeneous Information

Leon Shklar^{*‡} Kshitij Shah^{*‡} Chumki Basu^{*‡} Vipul Kashyap^{†‡}

shklar@cs.rutgers.edu

Abstract

In our previous work [shk94], we have discussed providing integrated and rapid access to existing heterogeneous information without any restructuring, reformatting or relocation of data. Such access is achieved through the generation of metadata that encapsulates the original information. We are currently working on designing a data modeling language aimed at controlling the metadata generation process. In this paper, we discuss applying this language to modeling heterogeneous information and demonstrate the ease with which the information may be made available for search and browsing.

1.0 Introduction

The InfoHarness[™] system [shk94] has been designed to provide rapid access to huge amounts of heterogeneous information without its relocation, restructuring, or reformatting. Many researchers have investigated the use of metadata to support run-time access to the original information [boh94, fis91, gar93, gro94, hsu91, jai94, sho93], as well as the use of data mining for the automatic extraction of metadata [mat93, sho93]. We have developed and synthesized some of the ideas contained in these efforts to provide advanced search and browsing capabilities without imposing constraints on information suppliers.

We are currently working on defining a data modeling language, which we refer to as the InfoHarness Repository Definition Language (IRDL), to control the metadata generation process. As in [kha94], we treat modeling as an abstraction for the purpose of understanding and utilization. Under this interpretation, related portions of information may be grouped together by imposing logical structures on the physical data. The idea is to provide a semantically-tailored view of data and facilitate search and browsing by imposing the *contains*, *is-contained-in*, and *part-of* relationships. The modeling approach to representing heterogeneous information is supported by IRDL, which aids in logically linking together heterogeneous components.

The InfoHarness prototype is now operational and is being trialed at Bellcore for building software repositories and storing geo-spatial data. It provides access to the original information from Mosaic and other World-Wide Web

(WWW) browsers. An interpreter that converts IRDL statements into metadata entities is under construction. In this paper, we demonstrate the flexibility of the language by discussing the generation of information repositories that contain images, documents, and arbitrarily formatted data.

In section 2, we define the basic concepts employed by InfoHarness. In the following section, we discuss the main features of our modeling language designed to control the metadata generation process. In section 4, we discuss the examples of applying IRDL to modeling heterogeneous information. In the two final sections, we discuss related work and outline our future plans.

2.0 Structure of Metadata

The important advantage of InfoHarness is providing access to information without making any changes in the location and representation of data. This is achieved by generating metadata and associating it with physical information. InfoHarness repositories are composed of metadata entities that are described in this section.

2.1 InfoHarness Objects

Metadata entities, which encapsulate units of physical information of interest to end-users, are called *information units* (IU). An IU may be associated with a file (e.g., a man page), a portion of a file (e.g., a C function), a set of files (e.g., a set of related man pages), or a request for the retrieval of data from an external source (e.g., a database query). For example, a C file and a function that occurs in this file may be encapsulated by separate information units.

An InfoHarness object (IHO) is defined recursively to be one of the following:

- A simple InfoHarness object, composed of a single IU.
- A collection object, composed of a set of references to other InfoHarness objects (*its children*).
- A composite object that combines a simple object and a set of references to other InfoHarness objects.

Each object has a unique identifier that is recognized and maintained by the system. Each simple and composite object stores the location of physical data, the logical address of the encapsulated portion of this data, and typing information that determines the data retrieval method. For example, an object that encapsulates a C function would be of type "C" and would contain the path information for the C file and the name of the function. The type of this object serves as an implicit reference to a retrieval method that would separate out a function from a

* Bell Communications Research, 445 South Street, Morristown, NJ 07960-6438

‡ Computer Science Department, Rutgers University, New Brunswick, NJ 08902

† LSDIS, Department of Computer Science, University of Georgia, Athens, GA 30602-7404

™ InfoHarness is a Bellcore trademark

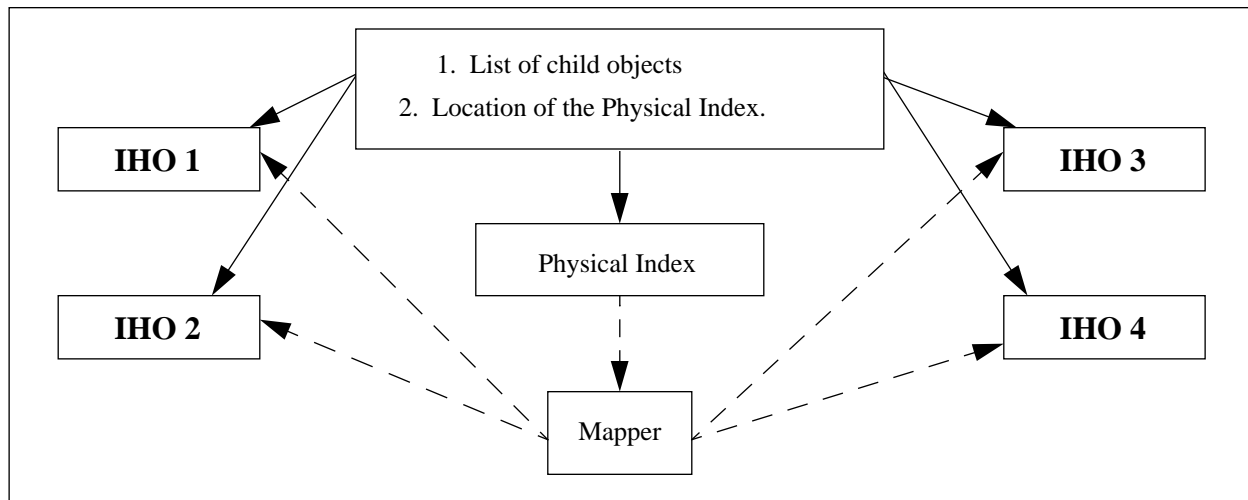


Fig. 1. Indexed Collections.

C file. In addition, each object may contain arbitrary number of attribute-value pairs (e.g., owner, last update, security information, decompression method, etc.).

2.2 Indexed Collections

Collection objects may contain references to independent indices that in turn reference the child objects (Figure 1). By the abuse of notation, we will refer to such collection objects as *indexed collections*, and say that an InfoHarness object belongs to an indexed collection if it is a child of a collection object. Each indexed collection stores the location of the index-related data structures and a reference to the independent query method.

3.0 Defining the Language

The main objective of our modeling language is to provide power and flexibility in building InfoHarness repositories. Each object in an InfoHarness repository may have multiple children, as well as multiple parents. The repository generation amounts to the creation of objects and their parent-child relationships, and to indexing physical information declared to belong to indexed collections.

The two main components of the language are responsible for introducing new types and for defining structures of information repositories. The *type definition* component of the language is responsible for adding support for new data types and new indexing technologies. In our experience, adding support for a new data type is quite simple. Adding support for a new third-party indexing technology is slightly more complicated, primarily because of the need to provide the run-time mapping between the portions of data known to the indexing tool and the InfoHarness objects.

The *structure definition* component allows users to impose desired logical interpretations on physical data. Its syntactic definition may be found in [shk95].

In Sections 3.1 and 3.2, we discuss adding support for new types of data and new indexing technologies. In Sec-

tion 3.3 we briefly summarize the structure definition component of the language.

3.1 Data Types

To support new data types, we must provide methods for encapsulating and browsing data. In the most trivial case of plain text, the encapsulation method associates simple objects with files, and the browsing method transfers the physical data to a WWW browser.

Other types may be more complex in encapsulating and browsing data. In the case of C code, the encapsulation method is responsible for associating simple objects with individual functions, while the browsing method is responsible for retrieving these functions and transferring the retrieved data to the browser. With postscript, the encapsulation is trivial, while the browsing requires a third-party tool.

Adding support for a new data type may be supported by a declaration of the following form:

```
DEF DATA <name> <format> <browser>
```

Here, <format> refers to the data encapsulation format that is either user-defined or is supported by InfoHarness, and <browser> identifies the browsing method.

3.2 Indexing Technologies

To support independent indexing technologies without any modifications to their implementation, we must provide a way of mapping the results of queries to InfoHarness object identifiers (Figure 1). InfoHarness objects must encapsulate those portions of physical data that are referenced by the indices.

For most indexing technologies [dee90,kah91], the results of queries are references to portions of either the pre-processed data or the original information. When generating an indexed collection, cross-references are being created between these portions of data and the InfoHarness objects. When processing the results of a query, these

cross-references may be used to find the encapsulating InfoHarness objects (Figure 1).

Many indexing tools support their own typing to perform filtering and browsing. When used within InfoHarness, the indexing tools are always made to assume that they are dealing with plain text, making their own types irrelevant to both reading and presenting the original information.

Adding support for new indexing technologies may be supported by a definition of the following form:

```
DEF INDEX <name> <build> <query>
      <type1> <type2>
```

Here, <name> is the name of an indexing technology (LSI, WAIS, etc.), <build> and <query> are shell-level commands for building and querying the index structures, and <type1> and <type2> refer to, respectively, the kinds of pre-processing of data and of run-time mapping of the results that are utilized by the indexing technology.

3.3 The Structure Definition Component

The main feature of the structure definition component of the language [shk95] is its non-procedural support for data encapsulation, set operations, and content-based indexing. This support is conditional on the availability of the appropriate type libraries that may be created either independently or by interpreting the type definition statements.

Data encapsulation is supported by specifying expressions in one of the following forms:

```
ENCAPSULATE <data_type> <location>
ENCAPSULATE <data_type> <object_id>
```

The second form should only be used in expressions in which the <data_type> is different from the type of the object defined by the <object_id>. For example, let "12345:/tmp/foo.c" be the identifier of an object of type "TXT" that encapsulates the file "/tmp/foo.c". Consider the following expression:

```
ENCAPSULATE C "12345:/tmp/foo.c"
```

The value of this expression is a set of identifiers of objects that encapsulate individual functions which occur in "/tmp/foo.c".

The creation of the composite objects is supported by the expressions of the form:

```
COMBINE <object_id> <set_expression>
```

The creation of indexed collections is supported by the following expressions:

```
INDEX <index_type> <set_expression>
      <location>
```

In addition, the language provides support for creating and maintaining set expressions, for accessing individual set elements, and for setting and retrieving individual attributes of InfoHarness objects. Examples of applying

IRDL to building repositories of heterogeneous information are discussed in the next section.

4.0 Applications

In this section we discuss using the structure definition component of IRDL to build InfoHarness repositories for a variety of applications. We begin with a very simple example of building an indexed collection of man-pages. Then, in section 4.2, we discuss a slightly more complex example of representing formatted documents. Finally, in section 4.3, we use the example of an environmental library to discuss building InfoHarness repositories from multiple heterogeneous components.

4.1 Man Pages

Consider the example of building a repository of man pages. It is reasonable to suppose that each man page is a unit of interest, and should be associated with an information unit. It is also reasonable to suppose that a single indexed collection should be created for all man pages. Given the location of man pages, their desired run-time representation, and the desired indexing technology, the following steps are required to create the repository:

1. Create simple IHOs to encapsulate individual man pages.
2. Create an indexed collection of simple IHOs created in step 1 using Wide Area Information Service (WAIS) technology [kah91] for indexing physical data.

```
BEGIN
  COLLTYPE  WAIS;
  DATATYPE  MAN;
  VAR IHO:  WAIS_Collection;
  VAR SET IHO: Man_IHO_Set;

  Man_IHO_Set = ENCAP MAN "/usr/man/man*";
  WAIS_Collection =
    INDEX WAIS Man_IHO_Set "/tmp/db/man";
  WRITE Man_IHO_Set, WAIS_Collection;
END
```

Fig. 2. Representing a collection of man pages

An IRDL program consists of declarations that are followed by statements. The first two declarations in Figure 2 are the type declarations, where *WAIS* is declared to be a collection type, and *MAN* is declared to be a data type. A collection type may only be declared if the methods that support corresponding indexing technology are available in the InfoHarness type libraries. Similarly, a data type may only be declared if the corresponding data encapsulation and browsing methods are available.

The following two declarations are the variable declarations that make *WAIS_Collection* an item IHO variable, and *Man_IHO_Set*, - a set IHO variable.

The first statement of the program encapsulates man pages located at "/usr/man/man*" and assigns the set of generated simple IHOs to *Man_IHO_Set*. The encapsulation is always controlled by an appropriate method. For the

type "MAN", this encapsulation method associates each man page with an IU.

The next statement requests that the man pages encapsulated by IHOs in *Man_IHO_Set* are indexed using the WAIS indexing technology, and that a collection object that references these IHOs is created and assigned to the *WAIS_Collection* variable. The final statement results in writing out the collection IHO and IHOs that encapsulate individual man pages.

4.2 Formatted Documents

Documents prepared by a text formatting system usually have an inherent structure. The document itself is a logical unit but it may also be considered to be a composition of sections, sub-sections, paragraphs, sentences, etc. In this example, we view a structured document as a composition of sections. The units of interest are sections that are associated with information units. This differs from the previous example because multiple information units may encapsulate different portions of data within the same file.

Given the location of the documents, their desired runtime representation, and the desired indexing technology, the following steps are required to generate the repository of documents:

1. For each document do the following:
 - 1.1. Create simple IHOs that encapsulate individual sections that occur in this document.
 - 1.2. Create a composite IHO that encapsulates the document and points to IHOs created in step 1.1.
2. Create an indexed collection of the composite IHOs created in step 1, using Latent Semantic Indexing (LSI) [dee90] for indexing physical data.

These steps are implemented by the sample IRDL program in Fig. 3.

```
BEGIN
COLLTYPE LSI;
DATATYPE TEX, TEX_S;
VAR IHO: Doc_IHO, LSI_Collection;
VAR SET IHO: Doc_IHO_SET, Section_IHO_SET;

Doc_IHO_SET = ENCAP TEX "/u/test/docs";
FORALL Doc_IHO IN Doc_IHO_SET
{
  Section_IHO_SET = ENCAP TEX_S Doc_IHO;
  Doc_IHO = COMBINE Doc_IHO
              Section_IHO_SET;
  WRITE Doc_IHO, Section_IHO_SET;
}
LSI_Collection =
  INDEX LSI Doc_IHO_SET "/u/test/db";
WRITE LSI_Collection;
END
```

Fig. 3. Representing a collection of TEX documents

The type declaration statements in this program indicate that *LSI* is a collection type and that *TEX* and *TEX_S* are data types. This assumes that the LSI indexing technology is supported and that the encapsulation and browsing methods for the types *TEX* and *TEX_S* (for documents for-

matted using the TEX program) are available in the Info-Harness type library.

Doc_IHO and *LSI_Collection* are declared to be IHO variables, each representing a single element, while *Doc_IHO_SET* and *Section_IHO_SET* are declared to be variables representing sets of elements.

The first statement of the program encapsulates TEX documents located at "/u/test/docs" and assigns the generated set to *Doc_IHO_SET*. It remains to encapsulate individual sections of all documents in *Doc_IHO_SET*. This is done by the first statement in the FORALL loop. We select the *TEX_S* type to encapsulate individual sections.

Composite IHOs are created by combining the document IHOs and IHOs that encapsulate sections of these documents. The composite IHO and the set of simple IHOs are written out by the last statement in the loop. Documents in *Doc_IHO_SET* are then indexed using LSI technology and the resulting indexed collection is stored in *LSI_Collection*. This collection IHO is finally written out by the last statement of the program.

4.3 Heterogeneous Information

To demonstrate the flexibility and the ease of use of IRDL, we present the example of an electronic library of the environmental data. The library houses a large collection of diverse data about the environment, with a focus on the utilization, evaluation and preparation of environmental impact reports, statements and related documents (i.e., "environmental technical reports"). The desired structure of this library is shown in Figure 4. Whenever any major civil or industrial project is proposed, a report has to be presented on the effects that the project will have on the environment. The report should also conform to the short and long-term plans that have been drawn up for that particular area.

The reports contain frequent references to pertinent data i.e. botanical, ground water, air pollution etc. In addition, other kinds of information may be crucial for the effective utilization of this material, e.g. maps, aerial photos, etc. [hol90, woo94]. In this example, we build a repository that would naturally support such a retrieval scheme. Without any loss of generality we restrict ourselves to plans and maps and assume that these exist for all counties of a particular state. Each county has the above mentioned plans composed of text and images.

The structure of the repository should facilitate the convenient retrieval of information related to individual reports. A logical way to accomplish this is to create complex objects that encapsulate textual contents of the reports and contain references to the related information. The textual contents may then serve as a basis for indexing the reports to simplify search and retrieval. The references may be used to associate the reports with plans and maps for the affected areas (counties).

Implementation of the desired repository of environmental reports requires the following steps:

1. Create a simple IHO for each report.
2. Create simple IHOs for the area plans and maps.

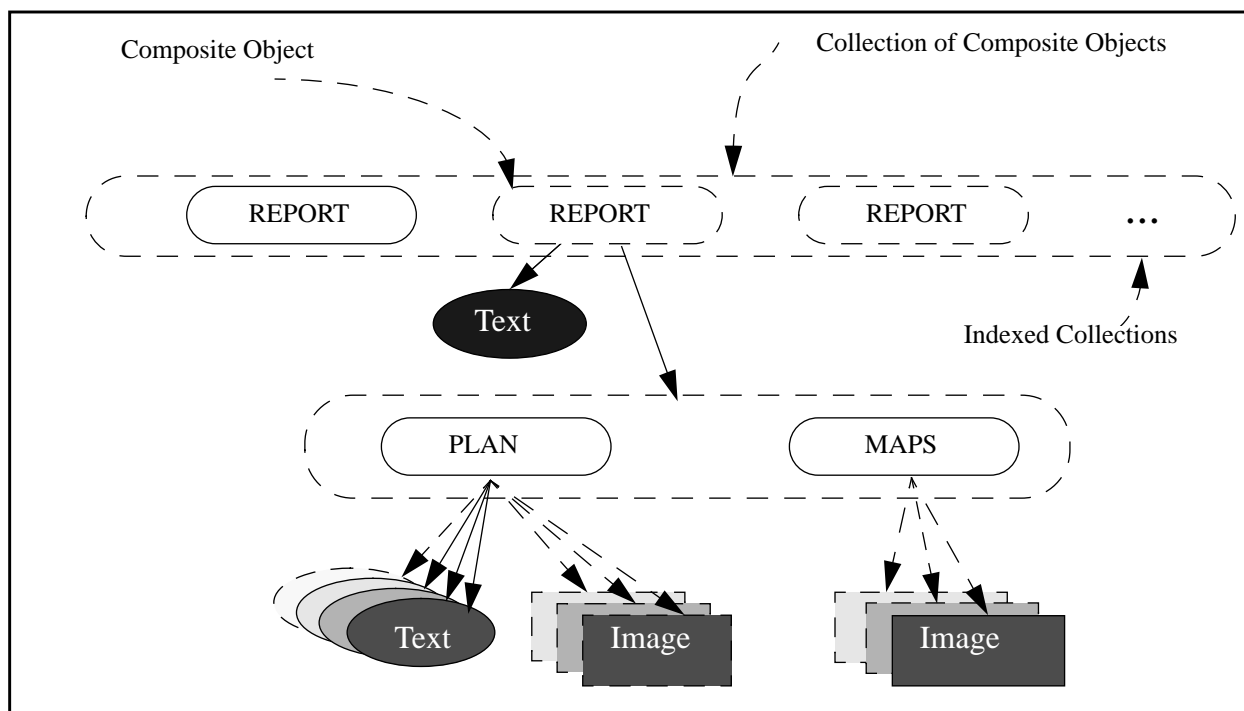


Fig. 4. The desired structure of an Environmental Library.

3. For each report:
 - 3.1 Find the plan associated with that report (based on a file-name attribute).
 - 3.2 Find all maps for each report using the association as in 3.1.
 - 3.3 Create an un-indexed collection of the results of 3.1 - 3.2.
 - 3.5 Combine this collection along with the IHO from step 1 to create a composite IHO for the report.
4. Create an indexed collection of all such composite report IHOs.

Steps 1 to 4 are implemented by the IRDL program in Figure 5. As compared to the examples in sections 3.1 and 3.2, this IRDL program has some additional complexity because of the selective grouping of objects to form heterogeneous sets. Maps and plans are being associated with a report based on the *FileName* attribute of the report's encapsulating object. The file-name prefix is treated as unique for the county for which the report is being prepared. Alternative grouping criteria may utilize other attribute(s) or an inherent language structure within the IRDL code.

The initial statements of the program declare *WAIS* to be a collection type and *TXT* and *MAP* to be data types. The following declarative statements introduce both set and item IHO variables along with a string variable. The first executable statement serves to encapsulate texts of individual reports that are available on the machine *ftp.foo.gov* at the location */u/jan95/refs*. Plans and maps are encapsulated in the similar manner.

For each report, *Join_SET1* is used to accumulate related *plan* and *map* IHOs. Sets of these IHOs are then

combined with *report* IHOs creating composite objects. These composite objects are accumulated in *Join_SET2* and are finally used to create the desired indexed collection of encapsulated reports and assign it to *Comp_Collection*.

5.0 Related Work

The access and retrieval of heterogeneous information has historically centered on different application areas, including software reuse, digital libraries, geo-spatial data, etc. Software reusability now extends beyond code and includes other software assets such as specifications, designs, test cases, plans, data, and documentation [pri88, len87, inc90, bur87,bas91, cro90]. The construction of digital libraries and the modeling of geo-spatial data require assembling a variety of media types, both structured and unstructured, and consequently ensuring ease of access and manipulation. The basic trade-off for these applications lies in balancing the cost of constructing a storage system versus the cost of locating and browsing relevant resources.

One of the most important questions in providing uniform and transparent access to information is data representation. Issues of data representation have long crossed the application boundaries and have called for a uniform solution. Short of a uniform data representation, which has not proved to be practical, the next best thing is a uniform data modeling approach.

Research on modeling data within a dynamic environment has identified two concepts - the set concept and the type concept. These concepts adequately capture invariant properties of data within a database architecture [ter92].

```

BEGIN;
  COLLYPE      WAIS;
  DATATYPE     TXT, MAP;
  VAR IHO:     Comp_IHO, Plans_IHO, Map_IHO, Comp_Collection;
  VAR SET IHO: Comp_IHO_SET, Plans_IHO_SET, Map_IHO_SET, JOIN_SET1, JOIN_SET2;
  STRING       Prefix;
  Comp_IHO_SET = ENCAP TXT "ftp.foo.gov:/pub/jan95/rep";
  Plans_IHO_SET = ENCAP TXT "/pub/doc/plans";
  Map_IHO_SET  = ENCAP MAP "ftp.fake.org:/pub/maps/nj";
  FORALL Comp_IHO IN Comp_IHO_SET
  {
    FilePrefix = ATTR Comp_IHO FileName;
    FilePrefix = SUBSTR(FilePrefix,0,index(Prefix,","));
    JOIN_SET1 = {};
    FORALL Plans_IHO IN Plans_IHO_SET SUCH THAT ((ATTR Plans_IHO FileName) =~ Prefix*);
    {
      Join_SET1 = {Plans_IHO, Join_SET1};
    }
    FORALL Map_IHO IN Map_IHO_SET SUCH THAT ((ATTR Map_IHO FileName) =~ Prefix*);
    {
      Join_SET1 = {Map_IHO, Join_SET1};
    }
    Comp_IHO = COMBINE Comp_IHO Join_SET1;
    Join_SET2 = {Comp_IHO, Join_SET2};
  }
  Comp_Collection = INDEX WAIS Join_SET2 "/u/design/index/ind2";
  WRITE Comp_Collection;
END;

```

Fig. 5. Sample IRDL program for representing an Environmental Library.

Our objective is to develop a simple, yet powerful, language that would support modeling heterogeneous information based on the underlying notions of sets and types. The structure definition component, which is currently the most well-developed component of the language, focuses on defining sets. The type definition component includes declarative support for new data types, as well as new indexing technologies.

Other methodologies exist for capturing the internal structure of heterogeneous data [con87, hal87, hal88, nie90, tom91]. Representations within hypertext systems capture basic notions of objects (nodes) and relationships (links) which are used to build complex structures (e.g., hierarchies, arbitrary networks). Various implementations have enhanced this representation with additional features that include:

- nodes for heterogeneous and user-defined data types,
- nodes with varying functionality (content-based nodes for capturing text versus structure-based nodes for building hierarchies),
- singular links encapsulating multiple relationships,
- nodes containing version histories,
- nodes and links with unlimited number of attribute-value pairs,
- graphical encapsulation of low-level data.

Despite these adaptations, a common usability complaint is the relative incongruity between the users' own conceptual representation scheme and that imposed by a system.

This often leads users to organize and structure their information outside the context of the system, hence, defeating its intended purpose.

Within data modeling, the process of imposing structure on data has traditionally stemmed from the observation of instances [ter92]. In this respect, set theory is a natural purveyor of data structuring's requisite tools. From a programming language perspective, set theory provides the operations necessary for the study of structured types [set89]. In IRDL, we emphasize the role of sets in structuring data.

The concept of a simple, declarative language to support modeling is not new. Although modeling languages borrow from the classical hierarchical, relational and network approaches, a number of them incorporate and extend the relational model. We categorize the languages examined below as algebraic model formulation generators (AMPL [fou87], GAMS [ken87], GEML[neu94], LINGO [lin91], LPL [hur89]), graphical model generators (GOOD [gys94], GYNGEN [for94]), and hybrid/compositional model generators with an underlying representation based on mathematical and symbolic properties (CML [fal94], SHSML [tay93]). In the following discussion, we examine these languages along the structure and the type definition dimensions. In all cases, a language is reflective of the modeling paradigm from which it evolved.

GOOD attempts to provide ease of high-level conceptualizing and manipulation of data. Sharing similarities with GOOD, GYNGEN focuses on process modeling by capturing the semantics underlying planning problems. CML and SHSML facilitate the modeling of dynamic processes. Although GEML and LINGO belong to the same class, LINGO is tailored for optimization problems while GEML abstracts the generic semantics of algebraic formulations. IRDL is being designed for data modeling, but it has common features with the process modeling languages discussed below.

GEML is a language based on sets. Structure is defined through a sub-language, derived from the principles of the Executable Modeling Language (EML), which evaluates numeric-valued, logical-valued, and index-set valued expressions in connection with solver interfaces, model debuggers, etc. GEML has both primitive and derived data types. Primitive types may be defined by the user or built-in scalars. Derived types are recursive applications of operations such as the Cartesian product or subtyping. Like GEML, IRDL incorporates the ideas of indexing expressions and set-valued expressions during object formation. In addition, GEML allows built-in, primitive, derived, and user-defined functions.

GOOD, GYNGEN, SHSML, and CML all employ graphs for defining structures. For the individual languages, variations arise when determining the role of nodes/edges as representations of the “underlying” concepts and composing and interconnecting them to produce meaningful representations. GOOD relies on the operations of node addition/deletion, edge addition/deletion, and abstraction to build directed graphs. If we think of objects as nodes and edges as relationships, IRDL supports node and edge addition but not deletions. In GOOD, node and edge labels may be associated with types. In IRDL, though we may assign types to nodes, we do not currently assign them to edges.

GYNGEN uses similar principles to build arbitrary networks from the graph structure. Nodes are characterized or “typed” by one or more attributes, capturing such properties as time. An attribute may be a primitive set or a simple index set. An attribute in this context goes beyond a “one-dimensional” characterization: a set of nodes may be viewed as a subset of the product space of its attributes. Furthermore, arcs can be defined between pairs of nodes subject to conditional expressions, a capability which is not provided by IRDL at this time.

Although GYNGEN addresses the process modeling issues, SHSML and CML are designed specifically to handle data dependencies arising from dynamic processes with time-varying properties. SHSML may be used to define hierarchies of interconnected components of different types. The lowest level components have a type which is “pure” continuous-time, discrete-time, or symbolic. Similarly, in IRDL, simple and composite objects store types that are representative of the encapsulated data. Finally, SHSML builds hierarchical architectures by arbitrarily assembling these components. IRDL goes a step further by modeling directed-acyclic graphs as well.

CML broadens the scope of these languages by adding extensibility to the process modeling representation. Structure in CML is domain-theory dependent, defined by a set of top-level forms. Domain theories are composed from components, processes, interaction phenomena, logical relations, etc. Since the syntax of CML is derived from Common Lisp, the types in this language include symbols, lists, terms composed of lists, sequences, and sets of sequences. An example of a “derived” type, *quantity*, captures properties such as dimension, time, and measure. The language promotes reuse of existing domain theories to model processes under a variety of conditions. This mirrors the InfoHarness class hierarchy where new types may easily be defined as subclasses of abstract classes. The abstract class hierarchy is stable and is designed to support method sharing between types.

6.0 Future Work

Our future work is directed towards the type definition component of our modeling language, and towards defining an action language for the run-time maintenance of changes in active repositories. We are also performing investigations aimed at the scalability of search by combining results of queries against independent indices. The initial public release of InfoHarness on the Internet is planned for the first half of 1995.

7.0 Acknowledgments

Our special thanks go to Amit Sheth of the University of Georgia for his insightful comments on the draft of the paper.

References

- [bas91] V.R. Basili, “Support for comprehensive reuse”, *Software Eng. Journal*, pp.303-316, 1991.
- [boh94] K. Bohm and T. Rakow, “Metadata for Multimedia Documents”, *SIGMOD Record, special issue on Metadata for Digital Media, December 1994*.
- [bur87] B.A. Burton, R.W. Aragon, S.A. Bailey, K.D. Koehler and L.A. Mayes, “The reusable software library”, *IEEE Software*, pp. 25-33, July 1987.
- [che94] F. Chen, M. Hearst, J. Kupiec, J. Pederson and L. Wilcox, “Metadata for Mixed-Media Access”, *SIGMOD Record, special issue on Metadata for Digital Media, December 1994*.
- [con87] E.J. Conklin, “Hypertext: an introduction and survey”, *IEEE Computer*, Vol 20(9), pp 17-41, 1987.
- [cro90] C.J. Crouch, “An approach to the automatic construction of global thesauri”, *Inform. Processing and Management*, vol. 26(5), pp. 629-640, 1990.
- [dee90] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer and R. Hashman, “Indexing by Latent Semantic Indexing”, *Journal of the American Society for Information Science*, 41(6), 1990.

- [fal94] B. Falkenhainer, et al. "CML: A Compositional Modeling Language", DRAFT.
- [fis91] G. Fischer and C. Stevens, "Information access in complex, poorly structured information spaces", *Proceedings of the CHI Conference '91, 1991*.
- [for94] M. Forster nad P. Mevert, "A tool for network modeling", *European Journal of Operational Research* 72, 287-299, 1994.
- [fou87] R. Fourer, D.M. Gay and B.W. Kernighan, "AMPL: A Mathematical Programming Language", *Technical Report 87-03 Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL*,
- [gar93] F. Garzotto. P. Paolini, and D. Schwabe. "HDM - A Model-Based Approach to Hypertext Application Design", *ACM Transactions on Information Systems*, 11(1), 1993.
- [gro94] W. Grosky, F. Fotouhi and I. Sethi, "Content-Based Hypermedia - Intelligent Browsing of Structured Media Objects", *SIGMOD Record, special issue on Metadata for Digital Media, December 1994*.
- [gys94] M. Gyssens et al."A Graph-Oriented Object Database Model", *IEEE Transactions on Knowledge and Data Engineering, Vol 6, No. 4,1994*.
- [hal87] F. Halasz, T. Moran and R. Trigg, "Notecards in a nutshell", *Proc. ACM CHI'87 (1987) pp 45-52*.
- [hal88] F. Halasz, "Reflection on Notecards: seven issues for the next generation of hypermedia systems", *Comm. ACM Vol 31 No 7 (1988) pp 836-852*.
- [hsu91] C. Hsu, "The Meta-database Project at Renesse-laer", *SIGMOD Record, special issue on Semantic Issues in Multidatabases, 20(4), 1991*.
- [hur89] T. Hurlimann, "Reference manual for the LPL modeling language (Version 3.1)", *Institute for Automation and Operations Research, university of Fribourg, 1989*.
- [inc90] A.J. Incorvaia and R.E. Davis, "Case studies in software reuse", *IEEE Soft.*, 1990, pp.301-306.
- [jai94] R. Jain and A. Hampapur, "Representations for Video Databases", *SIGMOD Record, special issue on Metadata for Digital Media, December 1994*.
- [kah91] B. Kahle and A. Medlar, "An Information System for Corporate Users: Wide Area Information Service", *Connexions - The Interoperability Report, 5(11), November 1991*
- [ken87] D. Kendrick and A. Meeraus, "GAMS. An introduction", *Development Research Department. The World Bank, 1987*.
- [kha94] S. Khajenoori, D.G. Linton and C.A. Morris, "Enhancing software reuse through effective use of the essential modeling approach", *Information and Software Technology, vol. 36, no. 8, pp. 495-501, 1994*.
- [len87] M. Lenz, H. Albrecht Schmid and P.F. Wolf, "Software reuse through building blocks", *IEEE Software, pp. 34-42, July 1987*.
- [lin91] LINDO systems, Inc., "LINGO optimization modeling language", *Chicago, IL, 1991*.
- [mat93] C.J. Matheus. P.K. Chan, and G. Piatetsky-Shapiro, "Systems for Knowledge Discovery in Databases", *IEEE Transactions on Knowledge and Data Engineering, December 1993*.
- [neu94] L. Neustadter. "A Formalization of Expression Semantics for an Executable Modeling Language", *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences, 1994*.
- [nie90] J. Nielsen, "Hypertext and Hypermedia", *Academic Press (1990)*.
- [pri88] R. Prieto-Diaz, "Domain Analysis for reusability", *Software Reuse: Emerging Technologies, IEEE Computer Society Press, 1988*.
- [ram94] M. Ramesh and H.R. Rao, "Software reuse: issues and an example", *Decision Support Systems vol.12, no. 1, pp. 57-77, August 1994*.
- [set89] R. Sethi. "Programming Languages: concepts and constructs", *Addison-Wesley, 1989*.
- [shk94] L. Shklar, S. Thatte, H. Marcus, and A. Sheth, "The InfoHarness Information Integration Platform", *Advance Proceedings of the Second International WWW Conference '94, October 1994, Chicago, IL*.¹
- [shk95] L. Shklar, K. Shah, and C. Basu, "Putting Legacy Data on the Web: A Repository Definition Language", *Special Issue of "Computer Networks and ISDN Systems" on the Third International WWW Conference '95, Vol. 27(6), 1995*.²
- [sho93] K. Shoens, A. Luniewski, P. Shwartz, J. Stamos, and J. Thomas, "The Rufus System: Information Organization for Semi-Structured Data", *Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993*.
- [tay93] J.H. Taylor. "Toward a Modeling Language Standard for Hybrid Dynamical Systems", *Proceedings of the 32nd Conference on Decision and Control, Texas, 1993*.
- [ter92] J.H. Ter-Bakke. "Semantic Data Modeling", *Prentice Hall, 1992*.
- [tom91] I. Tomek, S. Khan, T. Muldner, M. Nassar, G. Novak and P. Proszynski, "Hypermedia - Introduction and Survey", *J. of MCA Vol 14 No 2 (April 1991) pp 63 - 103*.

1. <http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/shklar/shklar.html>

2. <http://www.igd.fhg.de/www/www95/papers/78/irdl.html>