LexHarness: Providing Flexible Access to Legal Information on the World-Wide Web

Leon Shklar*  L. Thorne McCarty

Computer Science Department, Rutgers University, New Brunswick, NJ 08902
{shklar, mccarty}@cs.rutgers.edu

Abstract

In our previous work [shk951,shk952], we have discussed providing integrated and rapid access to existing heterogeneous information through the World-Wide Web (WWW) browsers without any restructuring, reformatting or relocating data. Such access is achieved by generating metadata entities that encapsulate the original information and using them at run-time to present this information. We have also proposed a data modelling language aimed at controlling the metadata generation process. In this paper, we discuss applying this language to modelling heterogeneous legal information that is already available on the Internet in some form, and demonstrate the ease with which we can achieve a big improvement in both presentation and search support without any duplication of data or intrusion with the existing service providers.

1.0 Introduction

The InfoHarness™ system [shk952] has been designed to provide rapid access to large amounts of heterogeneous information without its relocation, restructuring, or reformatting. Its prototype extension, called GeoHarness, which is a public system being designed to provide access to NASA’s remote sensing data, served as a foundation for this work. GeoHarness provides support for search and presentation based not only on the data content but also complex content-descriptive metadata.

We have also designed a data modelling language, which we refer to as the Information Repository Definition Language (IRDL), to control the metadata generation process [shk952]. As in [kha94], we treat modelling 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 modelling approach to representing heterogeneous information is supported by IRDL, which aids in logically linking together distributed heterogeneous components.

In this paper, we discuss how IRDL may be used to define information repositories for judicial opinions from the United States Supreme Court and the Circuit Courts of Appeal. Sample repositories of legal information are now operational and available for access from any Web browser through GeoHarness1. As we add more features specifically tailored for legal applications, we plan to create a new member in the Harness family to be known as LexHarness.

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

2.0 System Architecture

The current system architecture provides a platform for integrating information in a distributed environment by encapsulating new and existing information and meta-information, without converting, restructuring or relocating data.

* and Bell Communications Research, 445 South Street, Morristown, NJ 07960-6438
™ InfoHarness is a Bellcore trademark
1. http://www.cs.rutgers.edu/cgi-bin/nph-gh.cgi
Through object-oriented encapsulation, the system provides an integrated view and access to diverse and heterogeneous information. The system supports the use of independent tools for accessing, retrieving, and browsing data and content-descriptive metadata encapsulated by objects in the repositories.

As shown in fig. 1, the main components of the implementation of the current architecture are:

1. The **Harness Server**, which uses metadata to traverse, search, and retrieve the original information.

2. The **Hypertext Transfer Protocol (HTTP) Gateway**, which is used to pass requests from HTTP clients to the Harness Server (via an HTTP server) and responses back to the clients.

3. The **Metadata Generator** [shk951], which is not shown on this picture because it is used off-line to generate metadata that represents the desirable view on the structure and organization of the original information. This metadata is used by the Harness Server for run-time search and retrieval.

At run-time, HTTP clients may issue query, traversal, or retrieval requests that are passed on to the gateway, which performs the following operations:

1. Parses the request, and reads input information.

2. Establishes a socket connection with the Harness Server, generates and sends out a request, and waits for a response.

3. Parses the response, converts it to a combination of HTML forms and hyperlinks, adds the HTTP header, and passes the transformed response directly to an HTTP browser.

In the current implementation, users may combine simple attribute-based queries with content-based queries. The system architecture is open, modular, extensible and scalable. The Harness server implements an abstract class hierarchy that does not have to be modified to support a new data type, or a new indexing technology. Methods associated with the abstract classes are general enough because they are data-driven and may invoke independent programs. The definitions of terminal classes are also data-driven and are not part of the implementation, which makes the system capable of supporting arbitrary information access and management tools (e.g., browsers, indexing technologies, access methods).

With the emergence of Java [gos95] technology, it becomes feasible to significantly streamline the architecture to achieve better performance and reliability without sacrificing functionality. Moreover, one of the most exciting oppor-
The important advantage of InfoHarness and GeoHarness 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. The generated repositories are composed of metadata entities that are described in this section.

### 3.1 Information 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 dissenting opinion), a portion of a file (e.g., a summary), a set of files (e.g., a court case), or a request for the retrieval of data from an external source (e.g., a database query). For example, an opinion and its summary may be encapsulated by separate information units.

An information object (IO) is defined recursively to be one of the following:

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

Each object has a unique identifier that is recognized and maintained by the system. Each simple and composite object stores the possibly remote address 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 an opinion summary would contain the path information for the file containing the opinion and summary identification. The type of this object serves as an implicit reference to a retrieval method that would create a summary from an opinion file. In addition, each object may contain arbitrary number of attribute-value pairs (e.g., docket number, last update, security information, decompression method, etc.).
Collection objects may contain references to independent indices that in turn reference child objects (fig. 2). By the abuse of notation, we will refer to such collection objects as indexed collections, and say that an 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 an independent query method.

3.2 Metadata Generation

The generation of Harness repositories is controlled by IRDL. The IRDL Interpreter converts a high level IRDL program to a sequence of Generator calls (section 2.0). Each object in a Harness 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 building full-text indices from children’s contents and (or) their descriptive attributes.

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, which is currently being implemented, 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 run-time mapping between the portions of data known to the indexing tool and the Harness objects.

The structure definition component allows users to impose desired logical interpretations on physical data. Its syntactic definition may be found in [shk95]. The examples of its use for creating the repositories of court cases are discussed in section 4.0.

In section 3.2.1, we discuss adding support for new types of data and new indexing technologies. In Section 3.2.2, we briefly summarize the structure definition component of the language.

3.2.1 Data Types and Indexing Technologies

To add support for a new data type, it is necessary to provide methods for both encapsulating and presenting the data. At this time, such methods are generated by customizing one of the template programs. In the future, we intend to provide declarative support through the type definition statements of the language. To support independent indexing technologies without any modifications to their implementation, we must provide a way of mapping the results of queries to object identifiers (fig. 2). Of course, Harness objects must encapsulate the same portions of physical data that are referenced by the indices.

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. With postscript, the encapsulation is trivial, but browsing requires a third-party tool. In the case of court cases, the encapsulation method is responsible for associating simple objects with individual opinions and summaries, while the browsing method is responsible for grouping together portions of data related to a single case, generating HTML including both the internal hyperlinks to individual information units and external hyperlinks to potentially helpful information, and transferring the result to the browser.

For most indexing technologies [dee90,kah91], the results of queries are references to portions of either the preprocessed data or the original information. When generating an indexed collection, cross-references are being created between these portions of data and the Harness objects. When processing the results of a query, these cross-references may be used to find the encapsulating Harness objects.

Many indexing tools support their own typing to perform filtering and browsing. When used within Harness, 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. To add support for a new indexing technology, it is necessary to provide shell-level commands for building and querying the index structures and choose one of the pre-defined methods for the run-time mapping of query results into Harness objects.

3.2.2 Structure Definition Component

The main feature of the structure definition component of the language [shk951] 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 (section 3.2.1).
Data encapsulation is supported by specifying expressions in one of the following forms:

ENCAPSULATE <encapsulation_type> [presentation_type] <location>
ENCAPSULATE <encapsulation_type> [presentation_type] <object_id>

The encapsulation type determines what method to use to analyze the data and to generate metadata entities. The presentation type, which determines the run-time presentation method, is either specified explicitly or assumed identical to the encapsulation type. Another role of the presentation type is to control data pre-processing when building full-text indices. For example, if the presentation type is set to postscript, all control statements will be removed from the content prior to passing it on to an index-building program.

The second form of the ENCAPSULATE statement only makes sense if the <encapsulation_type> is different from that of the object defined by <object_id>. For example, let 12345 be the identifier for an object that encapsulates a file at ftp://ftp.law.vill.edu/pub/law/Fed-Ct/Circuit/3d/00index.txt that contains an index of cases for the Third Circuit Appeals Court at ftp.law.vill.edu. Consider the following expression:

ENCAPSULATE TEXT Court "12345"

The value of this expression is a set of identifiers for objects that encapsulate index information for individual cases which are referenced in ftp://ftp.law.vill.edu/pub/law/Fed-Ct/Circuit/3d/00index.txt.

The creation of 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> [presentation_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 information objects. Examples of applying IRDL to building repositories of heterogeneous legal information are discussed in the next section.

4.0 Applications

In this section we discuss using the structure definition component of IRDL for building Harness repositories of court cases. We begin with a very simple example of building an indexed collection of cases for a Federal Appeals Court. Then, in section 4.2, we discuss a more complex example of representing the Supreme Court cases where each case may be assembled from different sources.

4.1 Federal Appeals Courts

Consider the example of building a simple repository of court cases, where information about each case is assembled in a single file. It is reasonable to suppose that each case 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 court cases. Given the location of data, their desired run-time representation, and the desired indexing technology, the following steps are required to create the repository:

1. Create simple IOs to encapsulate individual court cases.
2. Create an indexed collection of simple IOs created in step 1 using the Latent Semantic Indexing (LSI) technology [dee90] for indexing physical data.

An IRDL program consists of declarations that are followed by statements. The first three declarations in fig. 3 are the type declarations, where LSI is declared as a collection type, and Court is declared as both an encapsulation and presentation type. A collection type should only be declared if the methods that support the corresponding indexing technology are available in the Harness type libraries. Similarly, presentation and encapsulation types should only be declared if the corresponding encapsulation and presentation methods are available. The two following declarations introduce LSI_Collection as an item IO variable and Case_IO_Set as a set IO variable.
BEGIN;

COLLTYPE LSI;
ETYPE: Court;
PTYPE: Court;
VAR IHO: LSI_Collection;
VAR SET IO: Court_IO_SET;

Court_SET =
LSI_Collection =
   INDEX LSI Court_IO_SET "file://localhost/tmp/db/Apppeals3c/";
WRITE Court_IO_Set, LSI_Collection;
END;

Fig. 3. Representing a collection of simple court cases at ftp.law.vill.edu.

Fig. 4. Query interface for the collection in fig. 3.

Fig. 5. Run-time presentation for objects in fig. 3.
The first statement of the program encapsulates individual case elements located at ftp://ftp.law.vill.edu/pub/law/Fed-Ct/Circuit/3d/ and assigns the set of generated simple IOs to Case_IO_Set. The encapsulation is always controlled by an appropriate method. For the type Court, this method associates each file with an IU. When the generated metadata entities are later used to provide run-time access to information, the original plain text will be formatted using the presentation method of the type Court before being sent to the browser (fig. 5).

The next statement requests that the man pages encapsulated by IOs in Case_IO_Set are indexed using the LSI indexing technology, and that a collection object that references these IOs is created and assigned to the LSI_Collection variable. The run-time presentation of this collection object is illustrated in fig. 4. The final statement results in writing out the collection IO and IOs that encapsulate individual court cases.

### 4.2 The United States Supreme Court

In this section we discuss how the collection of the U.S. Supreme Court cases at ftp.cwru.edu may be represented in a more meaningful way than in the previous section. Here, more than one file may be related to a particular case and some additional information is also available.

Given the location of the original information, the desired run-time presentation of individual cases, and the desired indexing technology, the following steps are required to generate the repository of the Supreme Court cases:

1. Create simple IOs that encapsulate individual elements of the court cases (one per file). The encapsulation method should locate the case numbers and associate them with every object.

2. For each object created in step one (unless it had already been excluded from the further consideration), find other objects related to the same case, join them together in a LIST with the presentation type Court, and exclude them from any further consideration. The presentation method for this type should be responsible for the generation of the internal hyperlinks to case elements and the external hyperlinks to related information (bios of the judges, etc.).

3. Create an indexed collection of the LIST objects created in step 2 using the LSI technology, as in section 4.1.

   These steps are implemented by the IRDL program in Fig. 6. The type declaration statements in this program indicate that LSI is a collection type and that Court is both an encapsulation and a presentation type. This assumes that the LSI indexing technology is supported and that the encapsulation and presentation methods for the type Court are available in the Harness type library.

   Obj and LSI_Collection are declared as item IO variables, each representing a single element, while Obj_Set, Case_Set, Join_Set and Join_Set1 are declared as set IO variables.

   The first statement of the program encapsulates individual case elements located at ftp://ftp.cwru.edu/hermes/ascii/ and assigns the generated set to Obj_SET. The next two statements initialize Join_Set and Case_Set variables. Join_Set is then used to accumulate objects from Obj_SET that should be excluded from the further consideration, while Case_Set is used to group together cumulative case objects. Next, the FORALL statement is used to go over the objects in Obj_SET using Join_Set to avoid going through the same case twice.

   Objects that are related to the same case are located using the CaseID attribute the value of which is set by the encapsulation method for the type Court in the ENCAP statement. All objects related to the same case are then grouped together in a LIST using the INDEX statement. Note, that the presentation type for these lists is set as Court to ensure proper formatted presentation of a single case (fig. 8). Finally, when all objects related to the same cases have been grouped together, an indexed collection is created for objects in Case_Set and the collection object is assigned to the LSI_Collection variable.

   Notice, that in this example each indexed object is itself a non-indexed collection. That is why the results of a query in fig. 7 look different than those in fig. 4: for each selected object we see not only a hyperlink for its content but also hyperlinks for the elements of the case. Same hyperlinks to individual elements (opinions, etc.) are available when the whole case is selected (fig. 8).
BEGIN;
  VAR IO: Obj, LSI_Collection;
  VAR SET IO: Obj_SET, Case_Set, Join_SET, Join_Set_1;

  Obj_SET = ENCAP Court "ftp://ftp.cwru.edu/hermes/ascii/";
  Join_Set = {}; Case_Set = {};
  FORALL Obj IN Obj_SET SUCH THAT (Obj NOT IN Join_Set)
  {
    Join_Set_1 = {};
    FORALL Obj1 IN Obj_SET SUCH THAT (ATTR Obj "CaseID" == ATTR Obj1 "CaseID")
    {
      Join_Set_1 = {Obj1, Join_Set_1};
    }
    Join_Set = {Join_Set_1, Join_Set};
    Case_Set = {INDEX LIST Court Join_Set1, Case_Set};
  }

  LSI_Collection = INDEX LSI Case_Set "file:/tmp/db/SupremeCourt";
  WRITE Obj_SET, Case_Set;
END;

Fig. 6. Representing the collection of the U.S. Supreme Court court cases at ftp.cwru.edu.

Fig. 7. Query interface for the collection in fig. 6.

Fig. 8. Run-time presentation for objects in fig. 6.
5.0 Related Work

Work on access and retrieval of heterogeneous information has historically centered on different application areas, including software reuse, digital libraries, geospatial 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 modelling of geospatial 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.

In the legal domain, very interesting work on maintaining a Web presentation standard for legal data and providing centralized access to various sources of information is being performed at Cornell’s Legal Information Institute. We believe our work to be complementary to these efforts. Our main objective is to provide a framework in which presentation standards (possibly multiple) may be easily imposed on heterogeneous distributed information. 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 modelling approach.

Research on modelling 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]. Our objective is to develop a simple, yet powerful, language that would support modelling 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 modelling, 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 modelling languages borrow from the classical hierarchical, relational and network approaches, a number of them incorporate and extend the relational model. We categorize these languages as algebraic model formulation generators (AMPL [fou87], GAMS [ken87], GEML [neu94], LINGO [lin91], LPL [bur89]), 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]). A detailed discussion of these languages and how they are related to our IRDL work may be found in [shk95-1].

2. http://www.law.cornell.edu
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.

7.0 References


