Between Now and the Semantic Web

Andrew Crossen, Jay Budzik, and Kristian J. Hammond
Intelligent Information Laboratory
Northwestern University
1890 Maple Ave.
Evanston, IL ISA
{acrossen, budzik, hammond}@infolab.northwestern.edu

Abstract

The Information Source Adapter Platform (or ISA Platform, for short) is a set of practical enabling technologies for developing intelligent information systems. The platform provides an infrastructure for building lightweight interfaces to existing resources that can be composed automatically using means-ends analysis. The ISA Platform enables developers to rapidly create new and innovative applications that leverage existing resources, without requiring the transition to semantic web technologies be complete. Two deployments of the technology are discussed.

Introduction

Many organizations have invested considerable effort in building information systems that support their work processes. These information systems are often built for a specific task, for example, storing project records. Large organizations can have hundreds of such systems that are actively in use. The systems perform well for the tasks they were designed to support, and are frequently tightly integrated into work process. In addition to internal resources, externally-hosted information systems, sometimes publicly available over the Internet (for example, Internet search engines) but also subscription services (for example, Lexis-Nexis) are often used by members of an organization to augment internal resources.

The evolution of these applications has resulted in the development of multiple data silos, often with proprietary interfaces. These data silos often do not provide standards-based access methods, making the development of information aggregation applications a costly, if not entirely prohibitive endeavor. In addition, multiple points of access and strict separation of data mean that significant effort is required to fully utilize existing data resources.

Semantic web technologies [2] provide an infrastructure for building applications that make use of distributed resources by allowing authors to describe those resources using a common ontology (e.g., [5]). Applications can discover, utilize and compose these resources when applicable. Yet these technologies all require information providers publish their data using an agreed-upon ontology. This means that progress in application development has to wait until providers migrate to standards which have yet to fully emerge. Thus applications are limited to simple domains [6] or require significant investment in information infrastructure [10].

We do not have to wait for the widespread adoption of semantic web technologies to start making progress on real applications that use semantics. Using the lightweight adapter platform presented in the following sections, we have built and deployed a variety of such systems using existing information services.

For example, one system supports the task of buying over-the-counter drugs in a brick-and-mortar drugstore by delivering relevant supplemental information to the consumer. The system's input is the barcode of the product. A product information database internal to the store is queried to look up information about the drug. Data such as the drug's type, active ingredients, and intended uses are then used to query an appropriate set of information sources in a second round of information gathering. The resulting aggregate data - retrieved from a variety of sources internally and on the Web - is parsed into a machine-readable format suitable for composition using the ISA Platform. This data is then presented to the user in a coherent fashion as a mini-site that reflects the semantics of the information retrieved and its intended context of use. Moreover, the information is usable given the constraints of the devices on which it will be viewed.

The Information Source Adapter Platform (or ISA Platform, for short) is a set of enabling technologies for developing interfaces to resources, and building intelligent information aggregation systems—virtual systems that leverage existing data silos—without requiring service providers adopt semantic web technologies. We aim to leverage the semantics associated with data contained in these resources, and in the application domain, while the transition to the semantic web is underway.

We have built and deployed a number of such applications, two of which will be described in following sections. The XLibris system [4] automatically aggregates subject-specific information about books and other retail products. The Watson system [3] automatically discovers relevant information in the context of document manipulation tasks. Our experience with the ISA Platform shows that it significantly reduces the time required to develop and deploy intelligent information systems. This application-centric focus allows us to make progress building innovative systems whose functionality has determined the capabilities of the infrastructure described

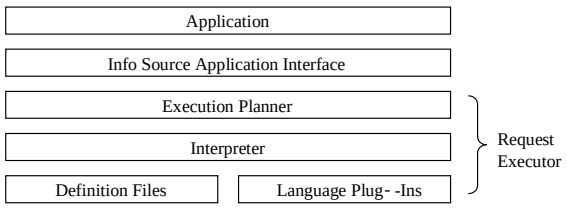


Figure 1: ISA Platform Layer Diagram

herein. Our view is this approach ensures the infrastructure we build is relevant, necessary, and as simple as possible.

Information Adapters

Achieving web service interoperability requires uniform data descriptions. Many existing systems (e.g. Google, Amazon) provide unambiguous data schema, but the use of this data by another application requires transformation into its own schema. This transformation requires a wrapper, or adapter component. The principal problems in adapter construction are source connectivity and parsing, which can be simple if an adapter development environment provides specific scaffolding to support them. The ISA Platform defines a unified access mechanism, and a simple parsing language that, unlike XSLT, works on any data format. Information from databases, Web search engines, and legacy systems can be parsed and used by other systems without the need for the providing source to subscribe to a web services model. Wrappers can expose these legacy services as web services themselves and provide semantic annotations that map directly into existing and evolving ontologies, facilitating the transition to the semantic web.

Information services are encapsulated by information adapters expressed in XML. Information adapters contain metadata descriptions of a source (semantic representations of its input and output characteristics). In addition, they contain descriptions of how to transform information goals generated by query producers (components providing input data for information adapters) into source specific queries, and also for how to parse the results of a query. Information items returned from executing queries can be dealt with in any number of ways by an application leveraging the ISA Platform; they can be immediately displayed to an end-user, dumped to a database, stored in shared memory for use in generating queries to another information source, or transformed via XSLT into any of the existing resource description formats and ontologies.

Information sources and transformations are stored in *definition files*. A *request executor* allows applications to specify information needs based on abstract requirements, without prior knowledge of the source of the information to
be retrieved.

The layer diagram presented in Figure 1 summarizes the architecture for the ISA Platform, which acts as the interface between information aggregation applications and distributed information sources. These applications interact with an API layer by posting a request and providing a sink, which will receive status messages and collect results. The API layer instantiates an instance of the execution planner, which produces an execution plan, consisting of a list of definition files to be executed by the interpreter. The interpreter executes the plan by running definition files, which are backed by language plug-ins.

Representing sources in definition files

Information source definition files contain a declarative representation of an information source or transformation and a procedural definition of a simple parser. The representation consists of the following three parts:

1. A *header*, which contains, in general, metadata concerning the source and a human-readable description.
2. An *executive summary*, which describes the source's input requirements, and what it produces as output, thus defining the contexts in which a source is applicable.
3. A list of *execution steps*, which describe the actions required to produce requisite information given inputs that match the capabilities of the source and the requirements of the step.

Information source headers contain a unique identifier, and the URI at which updates are available. They also contain a description, which defines the type of source the definition represents. For example, the following code extract is from the header of a definition file for the Library of Congress online card catalog:

```
<!DOCTYPE INFOSOURCE SYSTEM "isadapterdef.dtd">
   <INFOSOURCE UID="infolab.infoadapter.loc"
      SRC="http://isapkg.infolab.northwestern.edu/
           isadef/loc.xml"
      VERSION="1.0"
      LASTUPDATE="2004-02-09"
      LOADLIBRARY="BASE.DLL">
        <DESCRIPTION
          TYPE="Card Catalog"
          HREF="http://catalog.loc.gov/"
          NAME="Library of Congress Online Catalog"
          LOGOHREF="http://www.loc.gov/homepage/..."
          Library of Congress online card catalog.
        </DESCRIPTION>
    ...
```

ISA definitions receive syntactic validation through the XML parser by subscribing to an XML Document Type Declaration (DTD). The DTD defines all valid tags and attributes expected in definition files.

The executive summary contains a description of what a source or transformation requires as input and produces as output. For example, the Library of Congress online card catalog is searchable by ISBN (it requires an ISBN as input) and it produces the author and title of the work corresponding to that ISBN:

```
...
<EXECUTIVE_SUMMARY>
   <REQUIRES>
     <FIELD NAME="ISBN"/>
   </REQUIRES>
   <PRODUCES TYPE="SINGLETON" NAME="CATALOG_ENTRY">
     <FIELD NAME="AUTHOR"/>
     <FIELD NAME="TITLE"/>
     ...
```

```
   </PRODUCES>
</EXECUTIVE_SUMMARY>
...
```

Each information source has a set of execution steps, which, when executed with the information they require, produce the information described in the executive summary. Execution steps can be chained together much like information sources, but in the scope of retrieving information from a single source. This allows a single definition file to have multiple execution paths based on intermediate steps (e.g., the success or failure of an authentication process, or retrieval of a session ID from one page to drive search in another).

In the Library of Congress example, the online catalog first requires an HTTP request to retrieve a session ID, which uniquely identifies the request in their catalog system. Once the session ID is retrieved from one page, another HTTP request is issued in order to retrieve the actual record from the card catalog database.

Each execution step contains its own executive summary, as above, which describes its input requirements and what it produces at output. The information a step produces may be used by subsequent steps or may contain information that will end up in the results returned to an application.

The following execution step is typical of Web-based systems. It contains a `REQUEST` tag, which describes the HTTP request to post in order to retrieve the required information. Next, the `RESPONSE` tag contains a description of a lexical analysis routine, which populates a machine-readable schema with elements partitioned from the HTML source code returned by the information source. The parser defines a finite state automaton, which advances through the text stream until it has detected a relevant sequence of characters. Where available, XSLT-based parsing can also be supported using an alternate language plug-in.

```
...
<STEP NAME="PID_REQUEST">
   <PRODUCES NAME="PROCESS_ID">
      <FIELD NAME="PID"/>
   </PRODUCES>
   <REQUEST LOADLIBRARY="HTTPDLL.DLL">
      <ACTION STORE="PID_PAGE"
              METHOD="GET"
              HTTP_VERSION="1.1"
              URL="http://catalog.loc.gov/..."/>
   </REQUEST>

   <RESPONSE>
      <PARSE SOURCE="PID_PAGE" START_STATE="INITIAL"
             END_STATE="THE_END">
           <SKIPUNTIL FROM_STATE="INITIAL"
                      TO_STATE="PRE_PID">
               <![CDATA[&PID=]]>
           </SKIPUNTIL>
           <STOREUNTIL FROM_STATE="PRE_PID"
                       TO_STATE="THE_END"
                       FIELD="PID"
                       RESULT_ON="PROCESS_ID">
                <![CDATA["]]>
           </STOREUNTIL>
      </PARSE>
   </RESPONSE>
</STEP>
...
```

The language is expressive enough to build any parsing program. The structure provided by the primitives allows developers to focus on the important aspects of the parsing program instead of the details of the implementation.

Wrapper Data Verification and Error Handling

The ISA Platform is designed to provide flexible status and error reporting, both during development of wrappers and after the wrapper is in use by a client application. While these error handling techniques may sound like incidental implementation details, they are a requirement for a robust system with real world deployment and are given equal importance and consideration.

Status Reporting

Each component of the Platform propagates messages of various types to the *status reporter*, affording wrapper developers and end-users different levels of debugging and error reporting at different stages of interaction with the Platform. For example, wrapper developers using the Platform need detailed information about the parser at a certain stage of development. When a wrapper is built and fully operable, these detailed messages can be turned off. When the Platform is being used in a production application environment, end users won't care about the operations of the parser, but may want to be informed of errors crucial to system operation, such as a failed network request. Levels of debugging are arranged around the Platform system components, allowing developers to trace wrapper execution down to a set of logical components. To this end, the Platform provides an intuitive interface for managing the status and error reporter settings, allowing the system to be sensitive to users' and developers' needs.

Data Validation

Wrapper authors may not have control over interface changes that occur in the sources being wrapped. For example, if a web search engine changes the structure of their underlying HTML layout, chances are the parser for this source in the ISA Platform may become broken. In contrast, when the URI of a source's query form on the Web changes, the wrapper will be broken in another way. Thus the Platform provides mechanisms for tracking request failures and asserting validation properties of retrieved data, streamlining the (inevitable) wrapper maintenance task.

In practice, two factors can contribute to invalid wrapper operation for all types of data sources: (1) problems stemming from an improperly written wrapper for a properly working and accessible data source; (2) problems stemming from data source unavailability or interface change.

This results in five distinct retrieval error classes, each of which is handled intrinsically by the ISA Platform in a unified manner:

1. Source's retriever fails to locate resource
2. Source's retriever locates resource, returns results for a given input, but results are judged invalid by a set of validation rules.

3. Source's retriever locates resource, but fails to provide results for input due to no matching states in parser (in the case of parsing unstructured data).
4. Source's retriever locates resource, but fails to provide results for input due to improper state transition declaration – one or more states are matched, but state transitions are inappropriate (in the case of parsing unstructured data).
5. Source's retriever locates resource, but fails to provide results for input due to data structure change (in the case of structured data retrieval).

Moreover, the severity of the system's response to an error may differ depending on a number of factors. For some sources, no retrieval errors may be tolerated, and further attempts to use the source should be disallowed. For other sources it may be beneficial to allow some margin of error, especially during wrapper development and for those wrappers where all retrieval states are not known.

Either situation is supported through the use of an "Allowed Failure Percentage" (AFP) value for each ISA. This ratio, which the wrapper writer will typically adjust during the lifecycle of the wrapper, is defined as an attribute in a source's header, and determines the point at which the wrapper will be deemed unusable. The Platform allows for an administrator to be notified of all errors regardless of a source's AFP, or any subset of errors for which they are interested.

Wrapper errors in the class defined by 2 (above) require post-retrieval verification of a source's produced data, based on a set of rules the wrapper writer states about the data. The ISA Platform declares failure as binary, under Kushmerick's entailment assumption (one failed constraint assumes complete failure) equivalence assumption (all constraints are equal measures of failure) [7].

For a given field in an ISA definition, the following verification flags may be specified in the ASSERT attribute of a FIELD tag within a wrapper's executive summary. When the value of a field is available post-retrieval, one or more of these assertions, if stated, are tested against the resultant data item:

- `NUM_ONLY`: The field value should be purely numeric (no alphabetic or punctuation characters allowed).
- `NO_HTML`: There should be no HTML tags or fragments thereof in the field value
- `IS_URL`: The field value is a URL
- `OPTIONAL`: This field is optional and not required as part of a valid response.

Any number of other constraints can be built into the system to validate data items, although empirical evidence [7] suggests that the assertions provided handle the majority of the error cases.

Transformation of Retrieved Results

If the data retrieved via one or more queries to the ISA Platform is intended to be displayed to an end user, it is desirable to have the results be presented in a human-readable format. Moreover, the data may need to be displayed in different formats to best support the user's client, whether that's a standard web browser on a desktop machine, a WAP-enabled cellular phone, or a Palm/PocketPC device.

When it is finished retrieving data, the Platform returns all results bundled in an XML document. This allows the developer to build applications that use XSLT stylesheets to perform any number of transformations on the data. Stylesheets use XPath queries to select nodes out of an XML document, and perform transformations on them to describe how they should be presented to the user. Stylesheets can transform XML into an HTML document for standard browsers, WML for web-enabled cellular phones, into another XML-based format.

Formats such as RSS can be output by another transformation, and further labeled with terms from a standard ontology (e.g., [5]), effectively bringing the information source – or the application aggregating data across multiple sources – into the Semantic Web framework.

Planning and Executing Sequences of Queries

Applications provide the ISA Platform with a request, which consists of the information the application has in hand, and the information it wants the Platform to retrieve. For example, an application could post the following request for Web pages about a book with a given ISBN number:

```
<QUERY>
  <HAVE_NAME="ISBN" VALUE="0805210407"/>
  <WANT NAME="BOOK_PAGE"/>
</QUERY>
```

The planning system, a means-ends analysis engine, automatically constructs an execution plan for retrieving required information by combining information sources and input/output rules defined in the definition files. The execution plan is constructed based on the information an application has in hand, and information it is attempting to retrieve, using a library of information sources and transformations which, when executed in sequence, produce the required output. Figure 2b shows simplified pseudocode for the planning algorithm.

Once an execution plan is created, definition files are processed by an extensible interpreter, which performs the actions required to retrieve the requested information. Given a request, the request executor automatically finds a sequence of information sources and transformations to execute. Using the description of the input requirements and contents of a source's output, contained in the executive summary of each definition file, the system builds a dependency graph by chaining requirements. In cases where multiple sources satisfy a given goal, the system defaults to querying all sources. Optionally, a list of matching sources can be returned to an application for selection refinement.

For example, consider an application that requests Web pages about a book for a given ISBN number. The

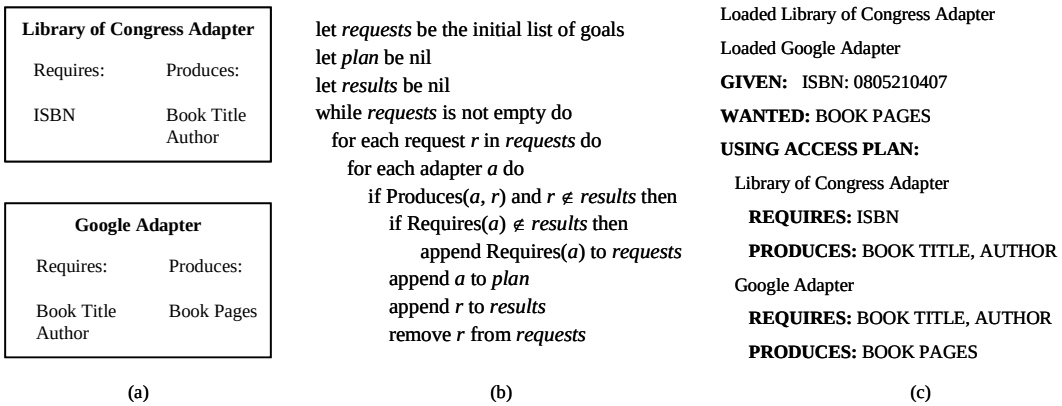


Figure 2: Access planning and source representation. Figure 2a displays two information source adapters (ISAs), Figure 2b shows simplified pseudocode for the ISA Platform planning algorithm, and Figure 2c shows a trace of the plan builder.

executive summary section of the Google definition file shows that it can produce book pages given a book's title and the author's name (see Figure 2a). The Library of Congress definition file shows that it can produce a book title and author's name given an ISBN number (Figure 2a). The application has provided the ISBN number, so the execution plan in Figure 2c is generated.

Once a plan is constructed, the executor instantiates an environment, which is populated with information contained in the application's initial request. Each information source or transformer is run in sequence, populating the environment with information gathered, including information necessary to run the next adapter. The interpreter is backed by *language plug-ins*, allowing developers to extend the functionality of the source definition language. For example, a developer could add support for:

1. legacy systems, by developing a plug-in that defines new language primitives that support access to the system,
2. systems that define a Simple Object Access Protocol (SOAP) interface, an XML-based mechanism for exchanging typed information, using the Web Services Description Language (WSDL), or
3. systems that define a Resource Description Framework (RDF) interface using RDF Site Summary (RSS) technologies.

Three plug-ins have currently been implemented: a base plug-in supporting general interpretation tasks and parsing functionality, an HTTP plug-in providing access to web-based sources, and an ODBC plug-in supporting access to structured data sources like relational databases. We have yet to implement the above protocols because resources that use them were not applicable to the systems built.

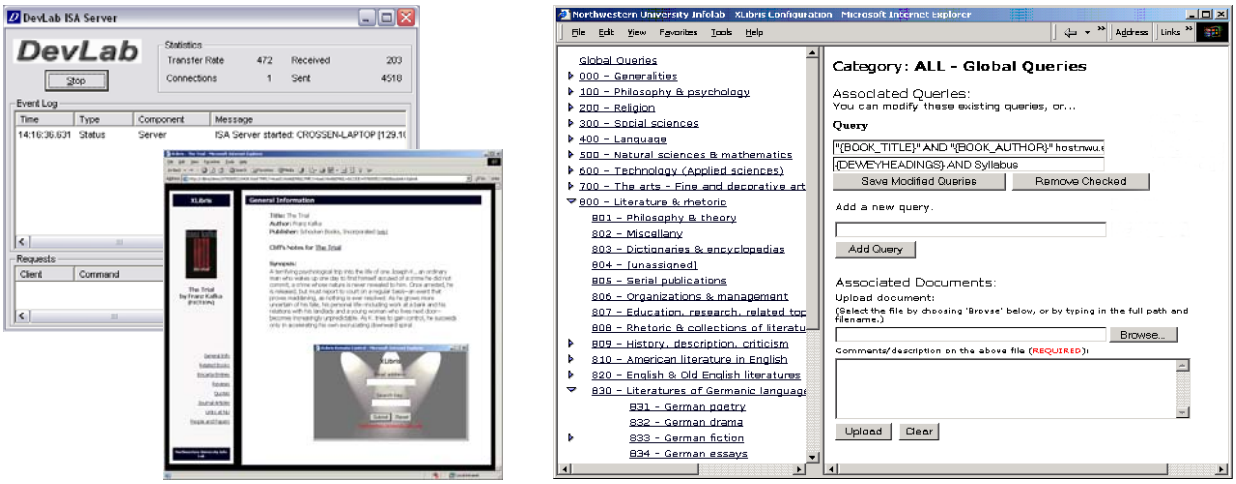
The XML definition file of each source is parsed into a DOM representation by the Xerces XML parser. The XML definition is interpreted by visiting the DOM tree in a depth-first manner. Each node is visited at least twice: once on the way down, and once on the way up. Visiting a node triggers a call to a function contained in a list of loaded language plug-ins. Plug-ins are loaded in-line as an attribute of XML tags, and apply to the children of the node at which they are loaded. During interpretation, a list of applicable plug-ins is kept in memory such that the "nearest" plug-in referenced is searched for the appropriate function; if the function is not contained within that plug-in, the next previously loaded plug-in is searched, and so on. Plug-in functions are passed the current environment, which they can modify. They can also return one of several control values that affect the interpreter's visit behavior, allowing for control flow constructs such as looping and branching.

The information gathered while executing the plan is returned to the executor in the form of XML-based messages, which are forwarded to the application that initiated the request. For example, the above example of a request for Web pages about the author of a book, given an ISBN number, results in the following messages being forwarded to the application:

```
<REQUEST_RESULT QUERYID="1" REQUESTID="2"
        SUMMARY="Encyclopedia Kafka, Franz Kafka,"
        TITLE="Encyclopedia.com - Results for Kafka,
              Franz"
        TYPE="AUTHOR_WEB_PAGE"
        UID="infolab.infoadapter.google"
        URL="http://www.encyclopedia.com/06796.htm"/>
<REQUEST_RESULT QUERYID="1" REQUESTID="2"
              SUMMARY="Metamorphosis: Translation"
        TITLE="Existentialism and Franz Kafka by
              Katharena Eiermann, Franz ..."
        TYPE="AUTHOR_WEB_PAGE"
        UID="infolab.infoadapter.google"
        URL="http://members.aol.com/CazadoraKE/..."/>
     ...
```

Applications

The above infrastructure has been instrumental in rapidly building a variety of intelligent information systems, two of which are described in detail below. In addition to these, we have also built:



(a) (b)

Figure 3: XLibris object information aggregator. Figure 3a displays ISA Platform server instance (background) used in building object information pages (foreground), Figure 3b shows hierarchical association tool driving source selection

- a contextual retrieval agent for broadcast news that supplements the news content with related material gathered from Web sites [8]
- a contextual retrieval agent for live sports shows that automatically gathers background information on teams, and relevant statistics about team members play-by-play [8]
- an art project that amplifies the emotional impact of a movie by automatically retrieving images associated with the dialog that are displayed surrounding the movie in real time, as the dialog occurs [9]
- a price point and product comparison system that gathers specific information helpful in decision-making for a given class of consumer products
- a plug-in for music players that automatically retrieves reviews, listings of concerts, song lyrics, etc., when a user inserts a CD or plays an MP3

XLibris

XLibris is an intelligent information system built using the ISA Platform [3]. XLibris uses a version of the ISA Platform exposed via a high-performance, socket-based server application. Figure 3a shows the dashboard of the server.

Users of the system can scan the barcode of a product into a tablet or Palm device, and the system will present them with a micro-site about that product comprised of information gathered from a variety of distributed resources.

XLibris systems in general use a unique object identifier symbol as a starting point in a multi-round information gathering process. The first pass involves looking up descriptors about that object in an appropriate database, providing the system with meta-information about the object, including the object's class. The object's class is mapped into a hierarchy of objects which activates a set of information goals associated with that type of object, and its parents in the hierarchy. This allows information specific to the object to be retrieved, where available, and ensures general information is always gathered. A handful of information sources capable of satisfying these goals are then automatically selected by the ISA Platform planning component from a wrapper-base of over three dozen wrapper descriptions of proprietary and publicly-available sources.

Hierarchical Source Selection and Plan Generation

How information goals are stored and activated are application-dependent. The XLibris book system seen in Figure 3a leverages the Dewey Decimal hierarchy for intelligent source selection. Figure 3b shows the association tool that knowledge engineers use to bind data sources and queries to appropriate nodes in the Dewey hierarchy. In the case of the XLibris book system, research librarians are the knowledge engineers. They have the domain expertise to populate the hierarchy with appropriate associations. For example, a data source applicable to 20th century German fiction will be bound to Dewey category 833, German fiction.

The Dewey classification of a book, as retrieved from the Library of Congress catalog by XLibris, is used by the ISA Platform to retrieve additional sources, and query templates to execute on them, from the hierarchy. Query templates are filled in by previously retrieved information. For example, one query template retrieves course syllabi that use a given book by querying a general Web search engine, restricted to .edu domains:

```
!{BOOK_TITLE} syllabus site:*.edu
```

The most specific sources are gathered first (those tied to the classification of the book), and further sources are collected by moving upwards (towards more general sources) in the tree. The data items produced by each of these sources become information goals in the Platform planning component.

This type of hierarchical source selection mechanism maintains transparency between the end users of a system and the knowledge engineers who marked up the hierarchy initially. The knowledge engineers can represent their expertise in data source content by making highly granular

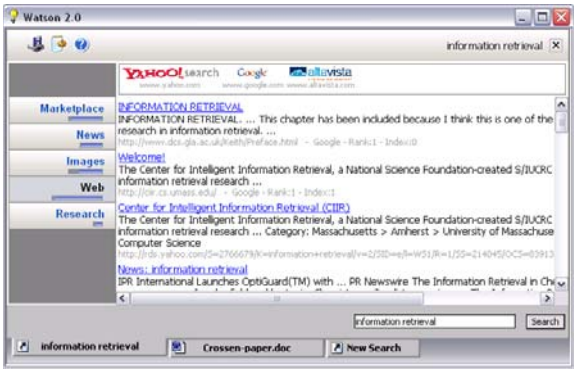


Figure 4: Watson with embedded ISA Platform

associations with equally specific concept nodes in the hierarchy. The end user benefits from this in using XLibris when they've scanned a book that is classified at the same level of granularity. In the cases where very specific sources (and thus very specific information) are not available for a given classification, the worst case scenario is that the information retrieved *is as good as possible*, because the sources hung from the parent node in the hierarchy will be used.

Watson

Watson is an intelligent search assistant [2] that retrieves information related to the document the user is viewing. Watson (see Figure 4), integrates with a variety of everyday applications on a PC and automatically builds a set of words and phrases that represent the document a user is actively working on. Watson then uses these words and phrases to proactively retrieve related, relevant information from a variety of online sources. In addition, Watson can be used as a standard meta-search tool.

Each information source Watson uses falls into one of several functional classes, which map directly to one of the query types that Watson's automated query generation algorithms produce. The ISA Platform allows for new information sources to be rapidly added by selecting from a set of existing query types. For example, one type of query produced by Watson is a phrases query, which contains extracted noun phrases. A new source that requires a phrases query (e.g., an image database, product catalog, or other sparsely-indexed source) can quickly be added to the application by providing a reference to its ISA definition file on the network in Watson's source catalog, and typing the ISA's requirements to be a phrases query. The ISA Platform implementation within Watson will then automatically use this source on the next run by matching the new source's requirements with the already-produced query as input.

Empirical Benefits

The above applications have benefited from the ISA platform in the following key ways:

1. Wrapper construction was accelerated six-fold. On average, skilled developers were spending six days to develop and test a single wrapper using a proprietary framework previously developed in-house using Java. With the ISA Platform, entry-level developers can accomplish the same task in a single day using our highly-optimized XML-based wrapper description language. Development time is further reduced for those sources that expose data in XML format because wrappers can be built using XSLT.
2. Identification of wrapper-related problems has been automated. Instead of determining there is a wrapper-related problem by noticing a system failure or receiving a complaint from a customer, the ISA Platform automatically monitors the performance of wrappers and notifies system administrators of any failure. The Platform further increases reliability by allowing for intelligent routing requests to alternate sources when a failure is detected.
3. Problem diagnosis and resolution has been greatly accelerated. The ISA platform provides a custom set of debugging tools that allow developers to quickly pinpoint problems. Because ISA definitions are interpreted as modules rather than compiled into systems, updates are reflected immediately in production systems.
4. The multithreaded architecture provided by the ISA Platform provides a three- to five-fold performance increase as measured by time to deliver results to the end user. The platform automatically identifies independent chains of retrievals and executes them in parallel, making the system highly scalable and much more responsive than a system employing serial execution.

In short, researchers spent more time focusing on building applications that are valuable to users, instead of the glue that holds them together.

Related Work

Work on agent representation languages and infrastructure for semantic web services [1,5] provide basic infrastructure for building intelligent, distributed systems. While this infrastructure is an important step, applications of this infrastructure are far from being usable in today's environments. Our focus has been to build only the infrastructure necessary to make progress in building intelligent systems.

Efforts like AgentCities [10] move beyond simple problems and demonstrate significant process in the application of semantic web technologies. However, we believe progress can be made using existing services, without such a significant investment in infrastructure and re-representation of existing sources.

Early systems built using the Ariadne framework [1] demonstrated the power of intelligent information

aggregation applications. Yet, still, the focus has remained on complex reasoning infrastructure unnecessary for many applications. The ISA Platform was built with web and high-level systems developers in mind. With our simple XML-based language, the Platform allows developers to rapidly build and deploy wrappers using a dialect of an industry standard language, custom-tailored for this purpose.

The ISA Platform represents a lightweight approach to building intelligent Web applications without requiring the significant initial investment in ontology and representation required by full-fledged semantic web services, nor a complex infrastructure for reasoning about relatively simple goals and plans.

Ongoing and Future Work

The Watson application is currently being deployed to a large user community. In the course of developing Watson and other applications that leverage the ISA Platform, we've assembled a collection of wrappers to over three dozen popular online sources. While wrapper construction time is minimal – on the order of one hour for a completely built and tested wrapper – we would like to automate this process further. The most time-consuming portion of wrapper writing is that of parser construction for sources that produce non well-formed data, such as HTML-based search engines.

To that end, we are working on a suite of tools that guide users in building wrappers and parsers by "watching" them interact with the sources in a natural fashion (i.e., in-line in a web browser). Our goal is to make wrapper construction a trivial, instrumental part of the larger, more important task of developing intelligent information systems.

Our goal is to provide users the capability of customizing existing systems or engineering their own from scratch. This is particularly important in the context of the XLibris system, where special purpose information should be returned for specific classes of books. We intend to allow librarians—experts in classification and the contents of library information sources—to continually refine and improve the system's ability to deliver the most relevant information to library users. This approach represents a departure from the processes typically associated with knowledge-based systems.

Moreover, the ongoing movement towards standards-based exchange of information online using frameworks such as RSS, SOAP and more advanced semantic web technologies is increasing rapidly. Because we intend the ISA Platform to continue to provide a robust, commercially-viable solution for rapid information source integration and aggregation, we are moving forward in providing inline interfaces for these standards-based protocols.

Conclusion

The ISA Platform represents an approach to building intelligent systems that is *application-focused*. By focusing on applications, we learn what reasoning and representation is necessary to build intelligent systems that function in real-world environments. Applications provide functional constraints, which, in turn, lead to tractable domains of semantic analysis, the value of which can be measured by the performance of the application. As we move our applications into full-scale deployments to demonstrate this value, we hope the ISA Platform will enable researchers and developers to build a new wave of intelligent applications that provide dramatically new capabilities to users.

References

1. Ambite, J., Ashish, N., Barish, G, Knoblock, C., Minton, S., Modi, P., Muslea, I., Philpot, A., Tejada, S. ARIADNE: A System for Constructing Mediators for Internet Sources, AAAI 1998.
2. Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. In Scientific American, May 2001.
3. Budzik, J., and Hammond, K. User Interactions with Everyday Applications as Context for Just-in-Time Information Access. IUI 2000.
4. Crossen, A., Budzik, J., Warner, M., Birnbaum, L., and Hammond, K. Xlibris: An Automated Library Research Assistant. IUI 2001.
5. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D. L., & Patel-Schneider, P. F. (2001). OIL: An ontology infrastructure for the semantic web. IEEE Intelligent Systems, 16(2):38--44.
6. Heflin, J. and Hendler, J. Dynamic Ontologies on the Web. AAAI 2000.
7. Kushmerick, N. Regression testing for wrapper maintenance. AAAI 1999.
8. Livingston, K., Dredze, M., Hammond, K., Birnbaum, L. Beyond Broadcast. IUI 2003.
9. Shamma, D., Owsley, S., Hammond, K., and Budzik, J., Network Arts: Exposing Cultural Reality. WWW 2004.
10. Willmott, S., Dale, J., Burg, B., Charlton, P. and O'Brien, P., Agentcities: A Worldwide Open Agent Network. In: AgentLink News, Issue 8, November 2001.