An Automated Client-Driven Approach to Data Extraction using an Autonomous Decentralized Architecture[1]

M. Brian Blake
Sr. Software Engineer
The MITRE Corporation
Center for Advanced Aviation System Development
1820 Dolley Madison Blvd
McLean, VA 22030
bblake@mitre.org

Patricia Liguori
Sr. Software Engineer
The MITRE Corporation
Center for Advanced Aviation System Development
500 Scarborough Drive
Egg Harbor Township, NJ 08234
pshearn@mitre.org

Abstract. Over recent years, "Internet-able" applications and architectures have been used to support domains where there are multiple interconnected systems that are both decentralized and autonomous. One such domain is the support of data management at the enterprise level. Typically, in these domains, both the schema of the data repository and the individual query needs of the users evolve over time. These query needs can include both the type of data and the required format and structure. To handle this evolution, the resulting architecture must enforce the autonomy in systems that support the client needs and constraints, in addition to maintaining the autonomy in systems that support the actual data schema and extraction mechanisms [1][5][9]. At the MITRE Corporation, this domain has been identified in the development of a composite data repository for the Center for Advanced Aviation System Development (CAASD). In the development of such a repository, the supporting architecture includes specialized mechanisms to disseminate the data to a diverse evolving set of researchers [2]. This paper presents the motivation and design of such an architecture to support these autonomous data extraction environments. This run-time configurable architecture is implemented using web-based technologies such as the Extensible Markup Language (XML) [6][11], Java Servlets [10], Extensible Stylesheets (XSL), and a relational database management system (RDBMS).

Keywords. Heterogeneous Distributed Information Systems, Novel Applications in information service systems

---

1. INTRODUCTION AND MOTIVATION

At the MITRE Corporation-Center for Advanced Aviation System Development (CAASD), researchers develop simulations for both design-time and real-time analysis. This research constitutes a wealth of knowledge in the area of air traffic management and control. This division of MITRE is split into a large number of individual groups that investigate various problems comprising the air traffic domain. Although the groups analyze different problems, the data to support the investigations are typically the same. Also, these individual groups develop simulations that require the data in different formats (i.e. specialized text files with delimited data, database format, XML, etc.) Moreover, each group looks at different subsets of data that may cross multiple data sources. Researchers are currently provided with data from outside sources that is gathered and distributed by a data librarian. This data is usually distributed in the same media and format in which it is obtained. The CAASD Repository System (CRS) team at MITRE has identified the need for obtaining the desired raw data from outside sources and building a composite data repository that serves the need of this diverse environment.

This paper presents the architecture that will allow this data extraction not only for the air traffic domain but also for other domains throughout MITRE such as army strategic motions and telemetry. The goal of the CRS team is to develop a dynamic architecture that will facilitate this data extraction for any data schema with the inclusion of some specific meta-data. In order to ensure the dynamic nature of this architecture, the CRS team take a distributed web-based approach that separates the major components of the architecture into various autonomous modules. Though the goal of the team is toward an architecture that will not change, this autonomy will ensure the reusability of each component in the architecture.

This paper will proceed in the next section with an overview of the CRS architecture and the autonomous modules. Section 3 provides a description of the actual technologies used to implement the architecture. In Sections 4-7, each of the modules will be discussed in detail. In the final sections, there is a comparison of this system to related third-party software tools and the relevance of the CRS architecture.

2. THE CRS ARCHITECTURE

The CRS architecture was devised to support a diverse set of customers/users. These customers internal to MITRE-CAASD use numerous technologies, programming languages, and interfaces. Web access is the one technology common to all groups. Therefore, the direction in designing the architecture was to use Internet technology as much as possible in gathering data request information and delivering the data to the customers. The CRS architecture is composed of four autonomous modules that fit seamlessly into the Internet module. These modules are the Client Interface Module, the Interface Specification Module, the Presentation and Query Module, and the Database Extraction Module. These modules can be split across three layers, the Interface Layer, the Presentation Layer, and the Data Storage Layer. These three layers and the underlying modules are illustrated in Figure 1.

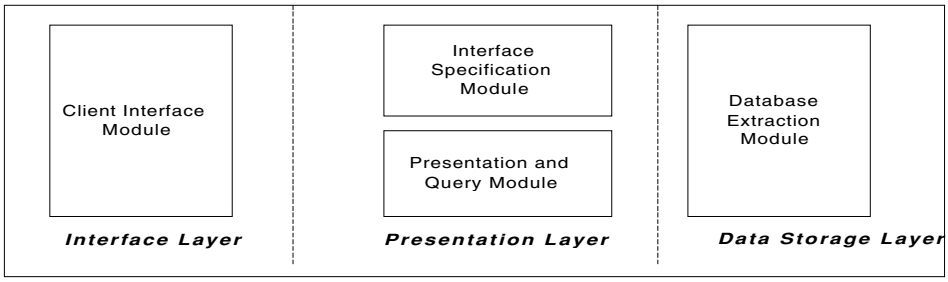

Figure 1 CRS Architecture by Layers

The Interface layer is the layer by which users can connect to the system. This layer consists of the Client Interface Module. Currently, Internet browsers implement the Client Interface Module. The customers use the browsers to connect to the system. In the future, this module might also include some stand-alone applications, which support data streaming. The Presentation Layer contains the Interface Specification Module and the Presentation and Query Module. Both of these modules include software services that provide a graphical user interface. The Interface Specification module allows the customers to customize their user interface to meet their specific needs. This is important considering the diverse data needs. The Presentation and Query module allows the customer to choose a standard or specialized interface in order to request data. Later, this module will need to be enhanced to explicitly allow the specification of business and domain logic. This module packages the information that will later be used in the Data Storage Layer. The Data Storage Layer contains functionality to maintain and extract data from some data repository. This layer consists of software services for extracting data from the relational database management system (RDBMS) [7].

Each module can further be decomposed into individual autonomous components. The decomposition of the modules is illustrated in Figure 2. As previously mentioned, the Client Interface Module currently contains Internet browsers that connect to the CRS system. The system provides two main functions for the users/customers.

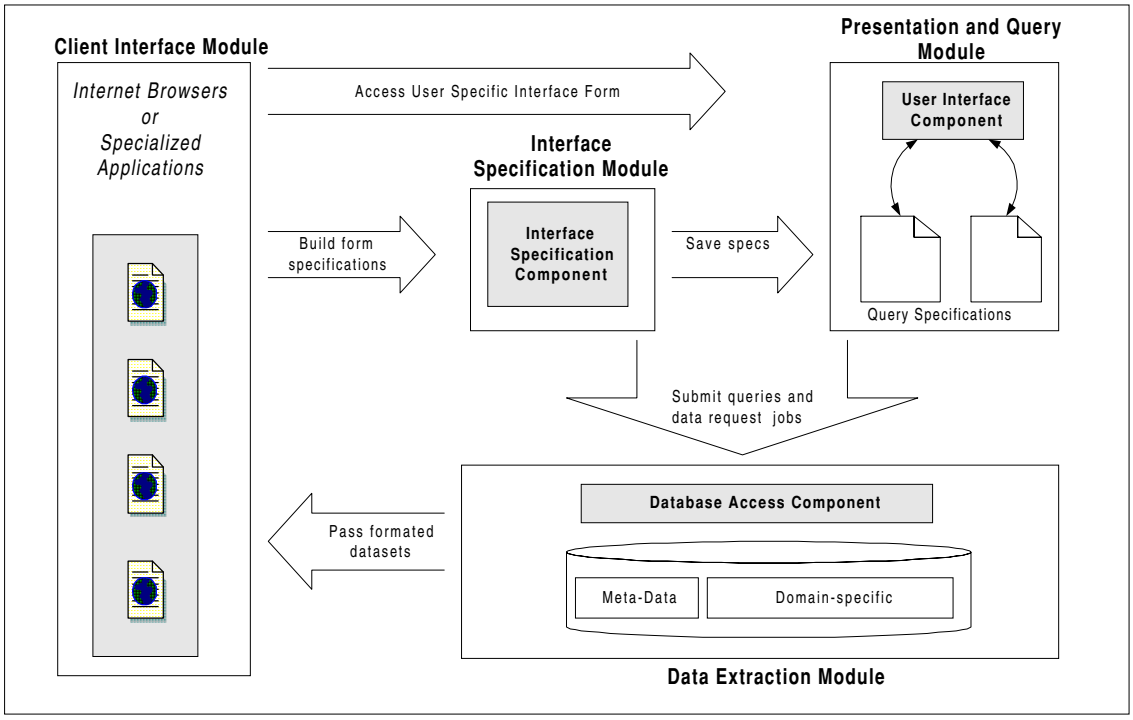

Figure 2 CRS Architecture by Module

In the first main function, a user can access the Interface Specification Module and design a personalized query form. This functionality is designed for users that need to execute repetitious personalized queries. The Interface Specification Component saves this personalized query form in a shared file system. The other function allows the user to access a personalized or standard query form within the Presentation and Query module and execute a query on the data repository. The User Interface component has access to the shared file system that stores the personalized and standard forms. The Data Extraction module is a service to both the Specification module and the Presentation and Query Module. The Data Access component accepts connections from the User Interface or Interface Specification component to satisfy internal or external services.

3 An Implementation of CRS System

At MITRE-CAASD, the CRS team has implemented the CRS architecture using various Internet technologies. Figure 3 presents the implementation that supports the details in the original architecture diagram. The CRS implementation mainly uses Java-based technologies.

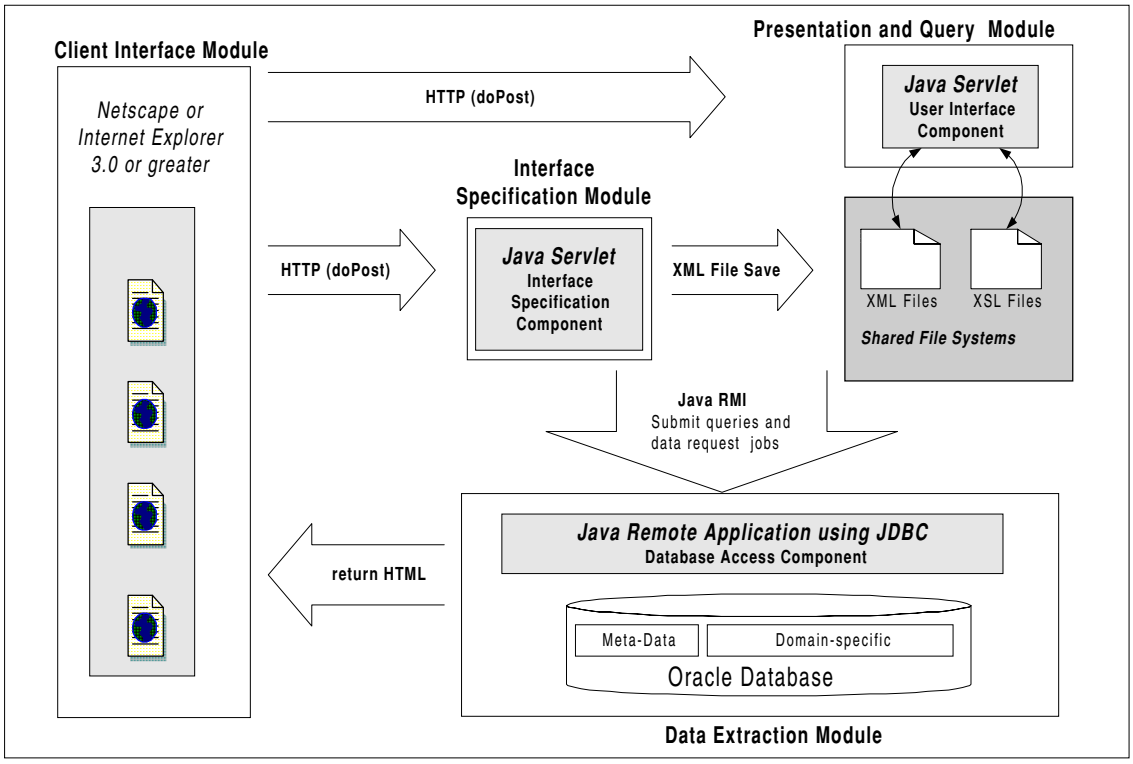

Figure 3 CRS System Implementation

The browsers in the Client Interface Modules connect to the Java Servlet-based components in both the Specification Module and the Presentation and Query Module. Both the User Interface component and Interface Specification component are implemented with Java Servlets integrated with Java classes that fulfill the underlying services.

The servlet in the Interface Specification module accepts information from the browsers in the Client Interface module in the form of an HttpServletRequest. The customers' query page specifications are encapsulated in the HttpServletRequest. This information can be parsed and used to generate the specifications for the HTML-based query form. This module stores this user preference information as an XML file in share file system location. In building this XML file, the module may have to gather database specific information. The intent is to use the XML file with a generic XSL file to automatically create the HTML-based query form. This XML file is the centerpiece of the architecture. This file contains a great deal of detailed information that allows the Presentation and Query Module to be generic. The benefit here is to allow outside sources to use the same XML format, and without the Interface Specification module, to have the ability to use the Presentation and Query module for data retrieval.

The servlet in the Presentation and Query module also receives an HttpServletRequest from the browsers. This module receives two independent messages. The first HttpServletRequest designates the

particular standard or user-personalized query form to display. Once the user is presented with the query form and submits it, the second HttpServletRequest includes information that will be used to create a generic query on the database. The components in this module make use of remote registry-based services from the Data Access Components to fulfill their database needs. Each of the modules has static interfaces in some cases asynchronous. Each module has autonomous functionality that can be changed or updated independently. The following sections discuss the low-level design associated with each module. Since the Client Interface module consists only of Internet browsers there is no need for an independent section discussing it.

4 The Specification Module

The Interface Specification Module is a stand-alone module that allows the user to specify the parameters that will be used in his/her data query. This module can be accessed via the Client Interface Module through the Internet. This module uses a servlet-based implementation. When a user accesses this module, a servlet is used to present various options for creating a staged query form. These options are used to create an XML file for this specific user. The main functionality in this module is to gather this information, combine the information with database specific information and create this XML file. Later in the Presentation and Query Module, this XML file will be combined with an XSL file to create the actual HTML-based query form.

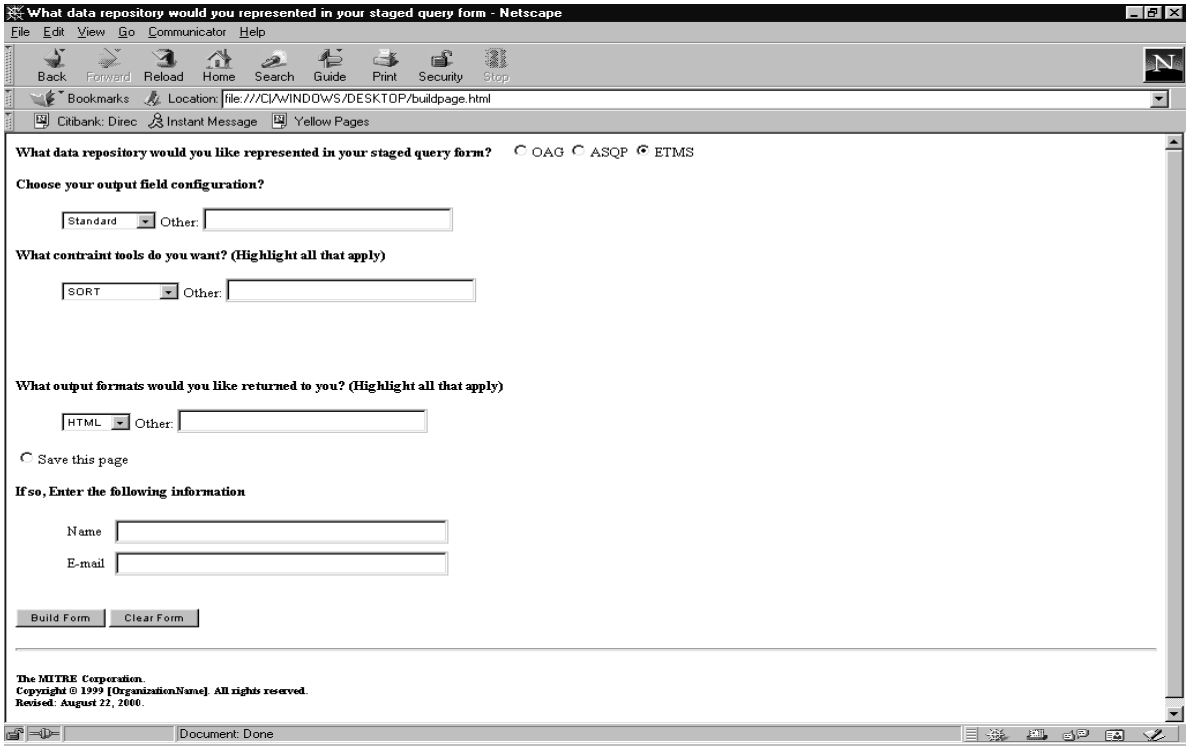

Figure 4 HTML Page used for User Specification

This process can be illustrated by showing the different components. The first component of the process is presenting the user with a form for specifying his desires on a specific data repository. Suppose a user desires to continuously query information from an airline-based data repository. This user would be presented with a form similar to that in Figure 4.

Perhaps the user chooses the options as illustrated in Figure 4. From the user selections it is obvious that the user wants data extractions from the ETMS data repository (an orientation specific to airline schedule information). The user also wants to use a pre-defined set of output fields that may be "standard" for this data repository. Other choices show that the user wants to be able to constrain the returned data by specific data values. In addition, the user chooses information to be returned as HTML. Finally, to record and monitor the use of this system, specific user information is recorded for each transaction.

At this point, the Interface Specification Component would gather these user preferences. The CRS system has numerous data entities in the database that serve as meta-data for the actual domain-specific data. The Interface Specification Component would use this meta-data to generate an XML file that will incorporate both database-specific knowledge and the knowledge of the user preferences. The process is illustrated in Figure 5.

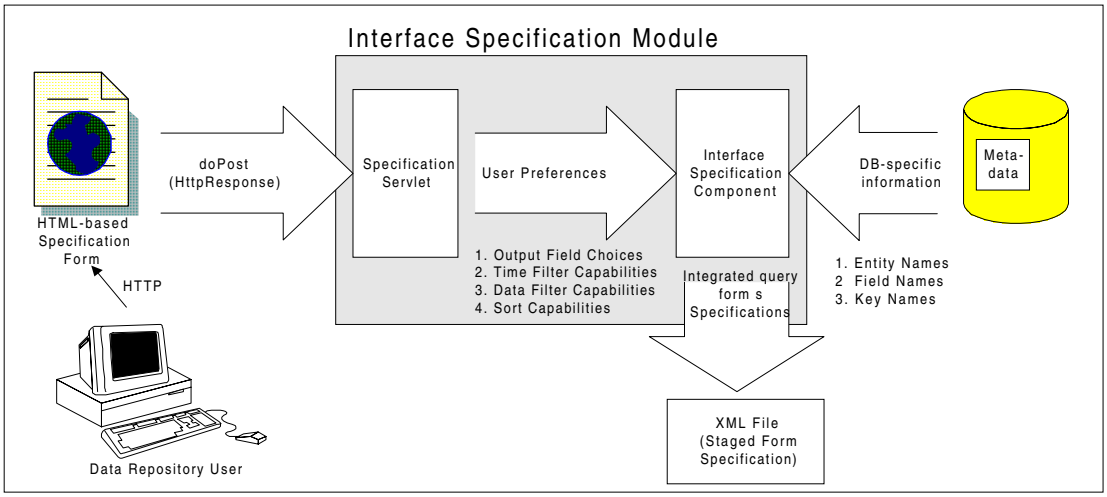

Figure 5 Process for building Stage Query Forms

By making the Interface Specification Module an autonomous entity, this will allow the addition of other user preferences in the future without affecting the other modules. The only connection between this module and the Presentation and Query Module is the XML file. As long as this XML file and HTML-based specification form maintains the same schema and set-up, any changes to the Interface Specification Module will be invisible to the operation of either the Presentation and Query Module or Client Interface Module, respectively.

5 The Presentation and Query Module

The Presentation and Query Module also is implemented with Java Servlets and additional components. When accessing this module, a user is presented with a list of standard and staged query forms. This user or other users may have built the staged forms previously. The standard forms are created by the same method but contains general query preferences. Once the user picks a particular form to use, this module selects the underlying XML form that represents these preferences. This XML file and a XSL file are submitted to an XSLT[2] compiler and a HTML-based staged query form is returned to user. This staged query form has database entity and field specific information. The Presentation and Query Module contains a User Interface Component that parses the information from this staged form. This information is in the form of an HttpServletRequest. The User Interface Component has generic functionality for parsing the selection information in the HttpServletRequest and building a database query. The process performed by the Presentation and Query Module is illustrated in Figure 6.

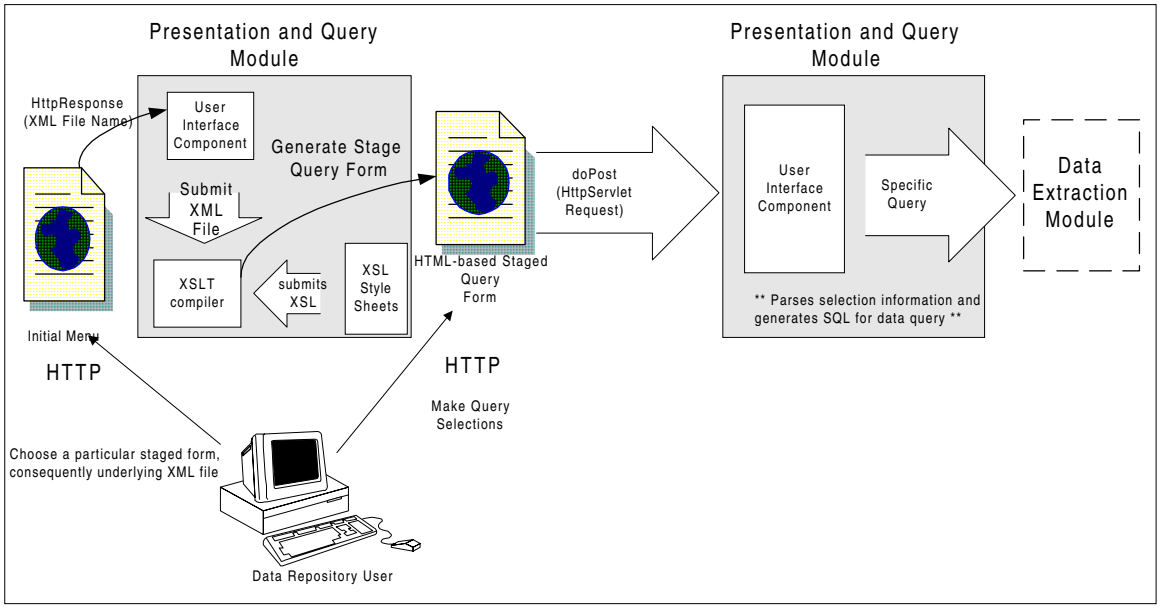

Figure 6 Processes in the Presentation and Query Module

The design of the User Interface Component has a strict object orientation [3]. The component is split into four main objects: ControlServlet, StagedPageParser, QueryElements, and QueryBuilder. The ControlServlet displays the graphical display to the user and captures the users' requests. The StagePageParser parses the users' requests and populates a list of the information. This list is used to create

[2] An XSLT Compiler takes XML and an XSL style sheet as inputs. Based on information from the XML file and the designated output format in the XSL, a file based on the specified format is returned as output.

a list of QueryElements. The QueryElements object is a parent class to several objects that represent different portions of the database query. These objects are the OutputElement, TimeRangeElement, TimeLengthElement, DataFilterElement, and SortElement. This is not a complete list of specializations of database query elements but it represents the general query needs for the CRS users. Additional query elements will be defined. This list of specialized QueryElement objects are passed to the QueryBuilder object. This object makes polymorphic calls on the QueryElements to populate the entire query string (The QueryElement has a QueryBuildable interface with the common methods, buildSelect, buildFrom, buildWhere, and buildSort). Consequently, the resulting query string is used as a parameter in requests submitted to the Data Extraction Module. The static view of the low-level design of the User Interface Component is displayed in Figure 7.

In Figure 8, a collaboration diagram is used to show the sequence of method calls in creating the query string. Initially, the doPost event is captured by the ControlServlet. The ControlServlet parses the resulting HttpRequest, and invokes the buildElementList method. This method instantiates a list of QueryElements based on the individual elements. This list of QueryElements is passed to the QueryBuilder object and each of the interface methods is invoked systematically.

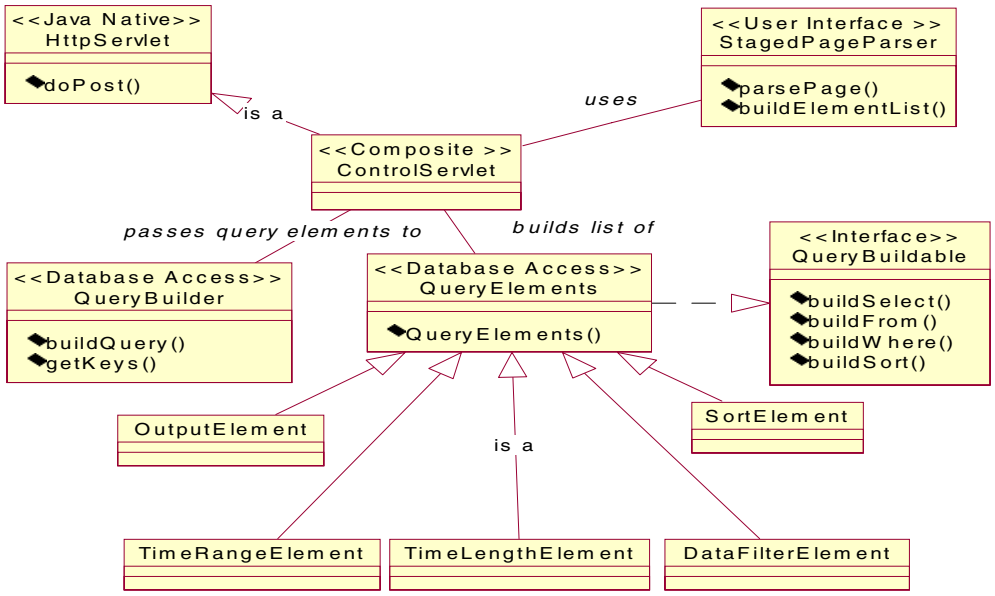

Figure 7 Low-level static view of User Interface Component[3]

[3] This diagram and subsequent diagrams were generated in Rational Rose Enterprise Edition.[8]

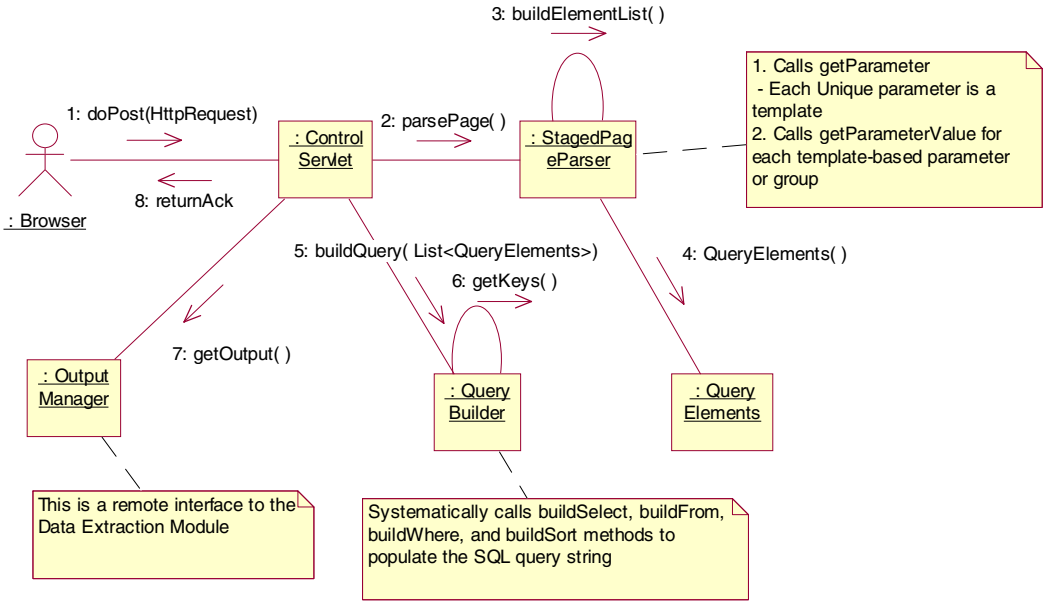

Figure 8 Collaboration Diagram of the Query Building Process

It is important to maintain autonomy in the Presentation and Query Module. In order for this module to be used for many data sources, it must not be bound to any particular database schema or query form. This module should have the ability to operate using a generic naming convention from the stage form and create a query in the universal SQL language.

6 The Data Extraction Module

The Data Extraction Module is basically an customized interface to a DBMS. This is not a new technology as many vendors have constructed APIs for DBMSs. Here, we specialize one of those APIs for use within the CRS architecture. In this implementation, we use Sun Microsystems' JDBC methods to connect to the Oracle database. The Data Extraction Module runs as its own process to help performance. This module registers methods that can be used by any other modules. These methods are registered as RMI remote objects. The main remote object is the OutputManager object. This object can be specialized to return HTML, ASCII, XML, etc (as in the HTMLGenerator, TextGenerator, and SpecializedFormatGenerator). A representation of this design is illustrated in Figure 9.

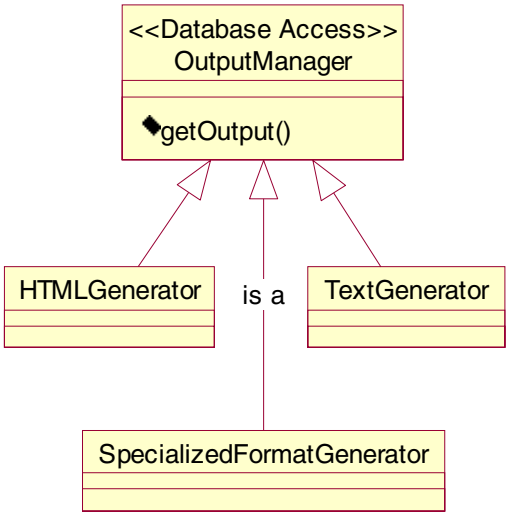

Figure 9 Low-level static diagram of OutputManager

This Database Extraction Module needs to be independent so that neither a change in the database nor changes in the query forms will affect its functionality. This approach allows the Data Extraction Module to reside on the same machine as the DBMS. By registering general services on the registry, other modules can access data from any networked machine.

7. SUMMARY

The CRS Architecture is a client driven approach to data extraction. This autonomous architecture allows a user to specify general types of queries and then provide the proper input to execute the queries as necessary. This architecture uses four modules for client interfacing, query specification, presentation and querying, and database extraction. By maintaining the autonomous nature of these modules, the architecture can adapt in the future with minimal lapses of services. In addition, the modules can be reused in other systems without being coupled to a particular domain. Moreover, as different business needs change, the system can adapt to the corresponding changes of data orientation.

8. RELATED APPLICATIONS

There are numerous applications on the market that allow the dynamic generation of query forms for the relational databases. Moreover, there are applications that allow this same dynamic generation for web based parameter forms such as Oracle's WebDB [7]. These approaches tend to be tightly coupled with the database. They neglect the need for distribution of services especially in enterprise solutions. By taking this centralized approach to data extraction, there tends be a centralized point of failure. Potentially, this

can be a terminal problem that causes a lapse in service. This centralized approach also makes it difficult to significantly increase performance because the main extraction tools reside on the same machine.

The CRS Architecture is a distributed approach. Individual functions within the system can reside on different machines. The independent nature allows these modules to be upgraded individually. Also, this architecture can make use of the processing time on multiple machines as opposed to one centralized location. Another advantage to this architecture is that it abstracts database specific information from the user. The users of CRS have merely to understand the output data, formats, and types of filters to operate this implementation.

9. ACKNOWLEDMENTS

There was a great deal of support given by members of the MITRE-CRS team consisting of Rob Tarakan, John Mack, Tho Nguyen, Gail Hamilton, Ted Cochrane, Jay Cheng, Dennis Sandlin, and Ali Obaidi.

REFERENCES

[1] Allen, R.J., Douence, R. and Garlan, D., "Specifying and Analyzing Dynamic Software Architectures," *Proceedings of the 1998 Conference on Fundamental Approaches to Software Engineering (FASE98),* March 1998

[2] Blake, M.B. "SABLE: Agent Support to Consolidate Enterprise-Wide Data Oriented Simulations," ", *Proceedings of the 4th International Conference on Autonomous Agents (AGENTS2000), Workshop on Agents in Industry,* Barcelona, Spain June 2000

[3] Booch, G., Rumbaugh, J., Jacobsen, I., "The Unified Modeling Language User Guide", Addison Wesley, Reading MA, 1998

[4] Bronsard, F, et al "Toward Software Plug-and-Play" *Proceedings of the 1997 Symposium on software reusability*, 1997 Pages 19-29

[5] Garlan, D. and Shaw, M., Software Architectures, Perspectives on an Emerging Discipline, Prentice Hall, 1992

[6] Glushko, R. et al, "An XML Framework for Agent-based E-commerce," Communications of the ACM 42, 3 pp 106-107 March 1999

[7] Oracle Corporation, WebDB Application 3.0 http://oradoc.photo.net/ora816/webdb.816/a77075/basics.htm

[8] Rational Corporation, Rational Rose Enterprise Edition, http://www.rational.com

[9] Shrivastava, S. and Wheater, S., "Architectural Support for Dynamic Reconfiguration of Large Scale Distributed Applications" *The 4th International Conference on Configurable Distributed Systems (CDS'98),* Annapolis, Maryland, USA, May 4-6 1998.

[10] Sun Microsystems Inc., Java Language Specification and the Distributed Event Model Specification, http://java.sun.com.

[11] Weiss, A. "XML gets down to Business," *Networker3,3* pp 36-37, September 1999