IEICE TRANS. INF. & SYST., VOL.E84–D, NO.10 OCTOBER 2001

LETTER *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems and Systems' Assurance*

An Autonomous Decentralized Architecture for Distributed Data Management and Dissemination

Malworsth Brian BLAKE[†] *and* **Patricia LIGUORI**[††]**,** ***Nonmembers***

SUMMARY 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. In enterprise-level data management domains, both the schema of the data repository and the individual query needs of the users evolve over time. 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. 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. 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), Java Servlets, Extensible Stylesheets (XSL), and a relational database management system (RDBMS).

key words: *software architecture, autonomous decentralized systems, XML, database management*

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 takes 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.

The next section this paper provides an overview of the CRS architecture and its autonomous modules. Section 3 provides a description of the actual technologies used to implement the architecture. In Sects. 4 and 5, we discuss its autonomy and current usage.

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 paradigm. 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 Fig. 1.

The Interface layer is the layer by which users can connect to the system. This layer consists of the Client

Manuscript received April 3, 2001.
Manuscript revised May 20, 2001.

[†]The author is with the MITRE Corporation Center for Advanced Aviation System Development (CAASD), 7515 Colshire Drive N420 McLean, VA 22102–7508, USA.
[††]The author is with the MITRE Corporation Center for Advanced Aviation System Development (CAASD), 500 Scarborough Drive Egg Harbor Township, NJ 08234, USA.

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).

Each module can further be decomposed into individual autonomous components. The decomposition of the modules is illustrated in Fig. 3. 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. 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.

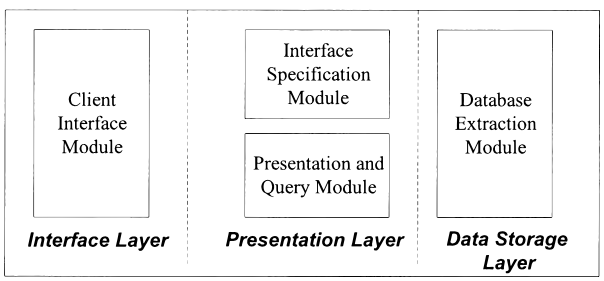

Fig. 1 CRS architecture by layers.

3. An Implementation of CRS System

At MITRE-CAASD, the CRS team has implemented the CRS architecture using various Internet technologies. Figure 2 presents the implementation that supports the details in the original architecture diagram. The CRS implementation mainly uses Java-based technologies. 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. These Servlets are integrated with

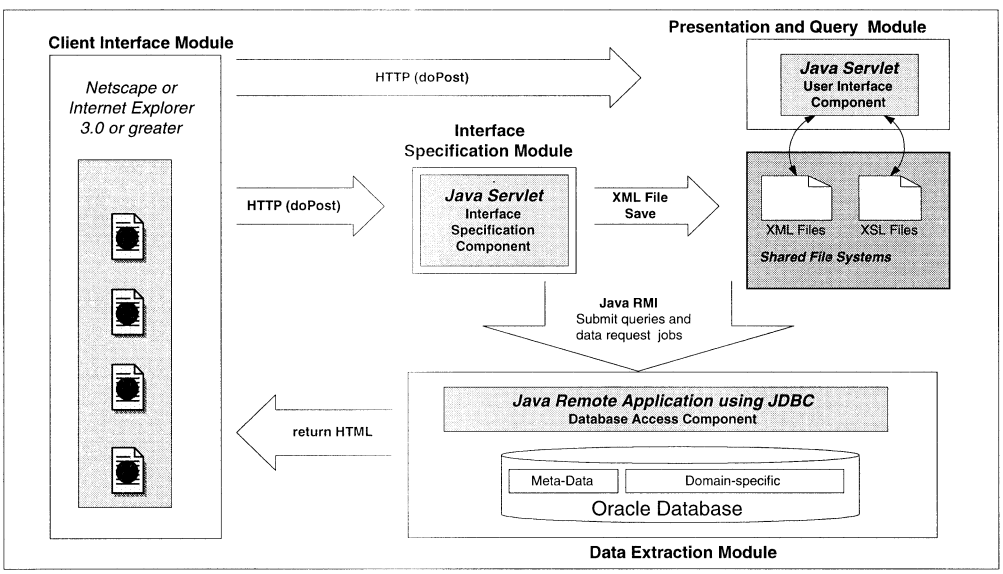

Fig. 2 CRS system implementation.

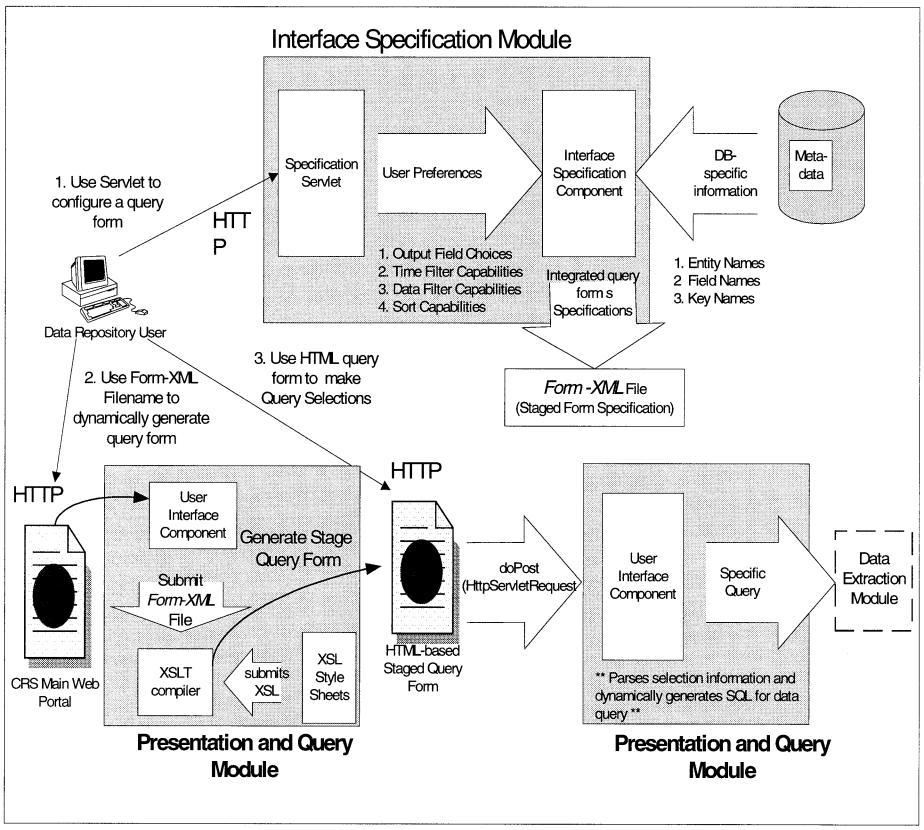

Fig. 3 Operation of the CRS implementation.

Java classes that fulfill the underlying query services. The Servlet in the Interface Specification module accepts information from the browsers in the Client Interface module in the form of an 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 a shared file system location. In building this XML file, the module gathers database specific information using the Data Extraction Module. This information is coupled with the user preference information. Subsequently, this XML can be processed with a generic XSL file to dynamically generate the HTML-based query form. This XML file is the centerpiece of the architecture as it is the basis for the execution of the system. This file contains 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. This personalized query form will be specific to each user.

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. In Fig. 3, the three main functions of the system are shown in relation to the modules. The following section discusses the autonomy of the modules comprising the CRS Architecture.

4. Autonomy in the CRS Implementation

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 respect to the evolution of the data as well as exceptions in the system.

The Interface Specification module and the Presen-

tation and Query module are completely autonomous. They are represented by independent Servlets. The Query Specification module creates XML files that are later used by the Presentation and Query module. Both modules share a file system location, but the operation or failure of one module has no effect on the operation of the other. The Data Extraction module is represented by remote components. There is a dependency on this module by both the Interface Specification module and the Presentation and Query module. However, by implementing the Data Extraction module as remote objects, the other modules can easily invoke new Data Extraction modules in event of a failure.

The autonomy of the Interface Specification Module and the Presentation and Query module could have been assured by incorporating the Data Extraction module within each. However, in doing this, both modules would be tightly coupled with the database. In creating a Data Extraction module, we further extend the autonomy of the other modules with respect to database location and connection protocol changes. The other modules can easily re-initialize the Data Extraction modules when failures occur. Moreover, the modularity of these Data Extraction modules allows them to be reused across MITRE-CAASD for this database or for any other.

5. Current CRS Usage

MITRE-CAASD has approximately 400 researchers that work to perform analysis and simulations for problems surrounding air traffic management and control. These researchers are split into approximately 50–60 different research teams. The CRS implementation currently has 34 staged and user-generated forms that support approximately 15 research teams. Currently all forms are connected to a single 5 gigabyte database. The systems performance varies on the size of the database tables. The ETMS dataset has information about the messages sent between aircrafts and air traffic controllers. Subsequently some tables are loaded with approximately 2.5 million rows per day. In returning 2 megabytes of filtered data from this dataset, we have experienced an average of 35 seconds delay. This performance is excellent considering the alternative is a human-supported data librarian. The Jeppesen dataset contains a finite amount of information that describes map coordinate information. This dataset only contains a total of approximately 10 megabytes of data. Consequently, filtered data queries only experience about 7 seconds of delay regardless of the size or complexity of the query.

6. Conclusion

The CRS Architecture is a distributed approach to data dissemination [2]–[4]. Autonomous modules within the system can reside on different machines. There is a great deal research in autonomous decentralized systems showing how autonomy and modularity can increase system utilization [1], [5]. The autonomous nature of the CRS system 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.

Acknowledgements

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

References

[1] T. Aizono, K. Kawano, H. Wataya, and K. Mori, "Autonomous decentralized software structure for integration of information and control systems," Proc. 21st International Computer Software and Applications Conference, Washington DC, Aug. 1997.

[2] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," Proc. 1998 Conference on Fundamental Approaches to Software Engineering (FASE98), March 1998.

[3] J. Ambite, Y. Arens, E. Hovy, A. Philpot, L. Gravano, V. Hatzirassiloglou, and J. Klavans, "Simplifying data access: The energy data collection project," IEEE Computer, vol.34, no.2, pp.47–54, Feb. 2001.

[4] M.B. Blake, "SABLE: Agent support to consolidate enterprise-wide data oriented simulations," Proc. 4th International Conference on Autonomous Agents (AGENTS2000), Agents in Industry, Barcelona, Spain, June 2000.

[5] B. Thuraisingham, "Object technology for building adaptable and evolvable autonomous decentralized systems," 4th IEEE Symposium Autonomous Decentralized Systems (ISADS99), Tokyo, Japan, 1999.