

# An Architecture to Support Distributed Data Retrieval in Specialized Formats

M. Brian Blake

Lead Software System Engineer Consultant  
Center for Advanced Aviation System Development  
The MITRE Corporation (CAASD)  
7515 Colshire Drive N420  
McLean, VA 22102-7508  
(703) 883-7084 (Voice) 883-1917 (Fax)  
bblake@mitre.org

Assistant Professor  
Department of Computer Science  
Georgetown University  
234 Reiss Science Building  
Washington, DC 20057  
(202) 687-3084 (Voice) 687-1835 (Fax)  
blakeb@cs.georgetown.edu

## ABSTRACT

Raw data and processed information are essential to the development of software simulation systems used in the analysis of various application domains. In such domains, the dissemination and management of this information is a daunting task. This paper discusses software architectural and specification-driven approaches to data retrieval in custom formats. Simulation developers use XML-based specification to request database results in any format.

## Keywords

Distributed and Web-Based Software Engineering, Software Architecture

## 1. INTRODUCTION

In simulation development domains, the dissemination and management of this information is a daunting task. Not only must this data support a heterogeneous array of researchers, but also the requirements on this data are constantly changing. With the acceptance of the Internet, the use of web-based technologies has been essential in providing information to a diverse set of stakeholders. Consequently, the CAASD Repository System (CRS), an application framework and architecture, was developed to provide this rapid web to database development. The technology behind CRS integrates meta-information with the mark-up language in standard WWW pages. In this way, CRS allows the autonomous evolution of the database and interface while disseminating the requested data in various formats such as delimited text, Extensible Markup Language (XML), Microsoft Excel, and HTML tables. However, a further problem discovered in this domain, is that existing and long-standing software simulation systems accept specialized entry formats as opposed to these standard formats. This paper discusses the architecture of CRS system and particularly the enhancements that use an XML-based specification to allow simulation developers to request database results in any format.

@ 2003 Association for Computing Machinery. ACM acknowledges that this contribution was authored or co-authored by a contractor of affiliate of the [U.S.] Government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

SAC '03, Melbourne, Florida USA

@2003 ACM 1-58113-624-2/03/2003...\$5.00.

The main thrust of this paper is to present new enhancements to the existing architecture work in the area of distributed data dissemination. There has been extensive research reported in context of this existing architecture [2][3][4][5][9]. The scope of this paper is not to reiterate that work, however, the understanding of the current work would be greatly benefited by an overview. The following sections summarize technical description and prior contributions of the initial CRS architecture work. Subsequent sections give the technical descriptions and motivation of the new research and results toward distributed data formatting. Finally, there is a case where we use the new distributed data formatting architecture.

## 2. BACKGROUND OF THE CRS

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, software engineers in these individual groups design and develop simulations that require the data in different formats (i.e. specialized delimited text files, database format, XML [15], etc.) Moreover, each group looks at different subsets of data that may cross multiple data sources. Initially, researchers were provided with data from outside sources that was gathered and distributed by a data librarian. This data was distributed in the same media and format in which it was obtained. Two major problems discovered in this domain were:

1. *Data sources were stored independently without any connection to other possibly similar data sources.*
2. *Groups, that use the same information, duplicated their efforts in parsing from the stored formats to more acceptable formats.*

The obvious solution was the incorporation of a database management system (DBMS). However, there were several caveats. The software engineers/researchers that developed these simulations were not regular database users and had no interest in becoming experts in the latest DBMS technologies. In addition, the software engineer/researchers were dispersed across decentralized sites using heterogeneous operating environments. Based on these caveats, the CAASD Repository System (CRS) team was formed to explore architectures that would provide a solution for this domain.

The CRS was devised architecture to support this diverse set of analyst and software engineers. These engineers 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, delivering the data to the customers, and integrating the sharing of their tools. 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 Management Module. The Client Interface Module is supported by Internet browsers where the customers (internal software engineers) use the browsers to connect to the system.

The Specification Module and the Presentation and Query Module 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. This can be a standard interface developed by the CRS team, that group, or an interface developed by another group. This module packages the information that will later be used in the Data Management Module. The Data Management Module 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).

The CRS architecture is developed with the latest in Internet technologies. Figure 1.0 presents the implementation that supports the architectural details. 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 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 combined with the user preference information in the Interface Specification Module. Consequently, this XML is 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, have the ability to use the Presentation and Query module for data retrieval. The Java 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.

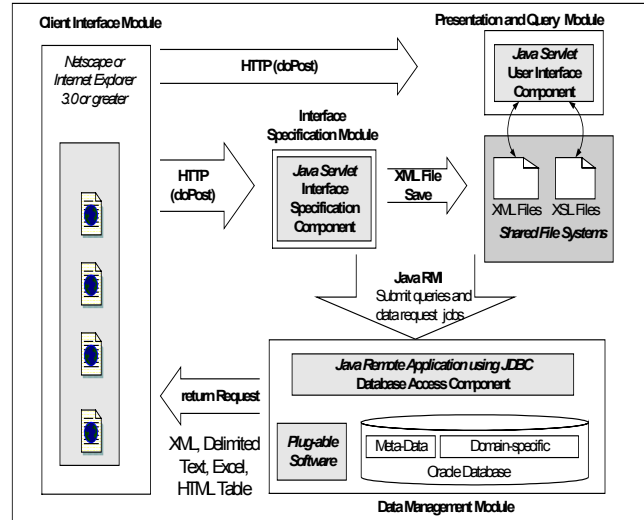


Figure 1.0. Architecture of the CRS System.

## 2.1 Contributions of the CRS Architecture

The major contribution of the CRS architecture is the run-time evolvability of the system when incorporating new data sources. Forms are specified with XML, thus they are not tightly coupled to the underlying software. Also a contribution in this work is that the Query Component produces the actual query string dynamically. This approach is unlike many other approaches that hard-code the query string in the query form [5]. The Query Component has intelligent functionality that dynamically accesses the database schema at run-time [2], so as the database changes the code does not have to change and queries that correspond to these changes are built dynamically. Currently the generic capabilities of the CRS architecture have been abstracted into a patent-pending framework named MRALD [9] (MITRE Resource For Accessing Loaded Data-sources).

## 2.2 Latest Areas for Improvement in CRS

The CRS has gained tremendous acceptance throughout MITRE-CAASD. The CRS implementation currently contains over 100 staged and user-generated forms that support approximately 20 research teams within the first year of operation. Currently, the architecture has been ported to other divisions in the MITRE Corporation outside of the air traffic control domain. This utilization has led to a number of excellent comments, thus leading to current enhancement projects.

The two most prevalent problems with the CRS is that pre-existing software simulations almost never use common formats like delimited text, XML, or Microsoft Excel as input formats. The pre-existing format requirements are always more cryptic and less standardized. The reason for this is because earlier software engineers tried to minimize parsing effort by creating input formats to their software simulations that were closer to the cryptic nature of the data as it is received. Ironically, in fact, the

data returned from the CRS is too clean to be accepted by most of the pre-existing simulations.

Another problem is that researchers, in addition, have their own pre-processing modules. These pre-processing modules further process the initial data into more of a “story” that software simulations can electronically enact.

CRS gave researchers the ability to access multiple data sources from one centralized area. Almost immediately researchers requested the ability to connect multiple data sources and perform several layers of queries where the result set of one query could feed the input of other queries. Specialized functionality was not initially possible in the first CRS architecture in order to maintain the evolvability. The three aforementioned problems can be summarized in the following list of requirements:

1. A need for specification-based database return formatting
2. A need for the ability allows modules to be plugged in
3. A need for the ability to handle multiple interconnected queries while returning the data as in (1)

The subsequent section discusses the CRS re-architecture that provides a solution to the aforementioned requirements.

### 3. SPECIALIZED FORMAT GENERATION

The Specialized Format Generation architecture (SFG) and supporting specification language, Specialized Format Markup Language (SFML), were devised to handle the requirements described in Section 2.2. The SFG architecture was designed in the same paradigm as the CRS architecture. The goal in the SFG as in CRS is to promote the separation of the interface and the software implementation. In order to do this we promote a specification-based approach. SFML is an XML-based specification that allows the users to specify the appearance of their database returns. Building on the current CRS architecture, this specification file can be automatically transformed into a web-based user interface. This interface has input fields to allow the specification of more database filters.

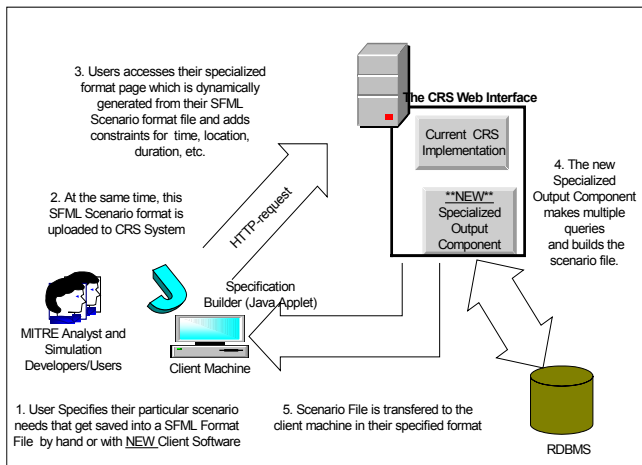


Figure 3.1. Specialized Format Generation Operation Flow.

### 3.1 SFG Overview

The operational flow of the SFG architecture follows the same operations as the CRS. The first step in the process to complete a The operational flow of the SFG architecture in relation to the

existing CRS is illustrated in Figure 3.1. This section describes the operational flow of the SFG architecture. Also there is a description of the semantics of SFML. Finally, there is the object-oriented design of the enhanced SFG component and how this component interacts with the SFML-based output format specification. Specialized output is the generation of a user-specific SFML file. Expert users should be able to construct this file directly. However, most of the CRS researchers would need the use of a tool that handles the database specific aspects. In this case, the user would access a local SFML builder tool that connects to the database and allows the user to build their file based on the domain-specific data and the database schema and fields. The existing CRS system provides an interface where XML files can be uploaded to a common repository. At this point, a user can access this file and the CRS system uses an Extensible Stylesheet Language Transformations (XSLT) component to convert the file into a HTML web interface. Using this interface, the user can now enter additional information that can be used to filter the database results. Using the new SFG component combined with the existing CRS intelligent query building components, the architecture iteratively queries the database to produce the specialized output. Finally, the specialized output is returned to users client or to a pre-specified file location.

### 3.2 SFML

The purpose of the SFML file format is for users to specify how they want their database returns to look. In the case of the MITRE-CAASD environment, these files typically resemble the formats that are acceptable as input to software simulations used in analysis. SFML uses the concept that users will specify their format using a list of unique lines. These unique lines can be described in terms of database row returns. In this way, the SFML files connect output formatting to database returns. In all database queries, there is some “base” filter. In SFML, this base case represents the main line. This main line is the foundation for the output file. Other lines can be derivatives of this main line. Derivative lines can develop new queries based on data returns from the query represented in the main line. In addition, derivative lines can get information directly from main line’s information. There can also be multiple main lines. In each line, there is a specification of where the output will go. Lines can output back to the users’ screen, to a file, or to multiple files as specified in the SFML. This is helpful in building multiple format files from one query.

Lines can be represented as a set of line elements. Line elements can be mapped to a specific column that is returned from a row at the completion of a query. Line elements contain special formatting. For example, a line element can specify if there is a trailing space. Also, line elements can specify the case of the characters. An example of line and line elements is shown in Figure 3.2. This example shows how a unique line of data such as “Airport Information” can be separated into four data fields. These data fields are sequenced as line elements.

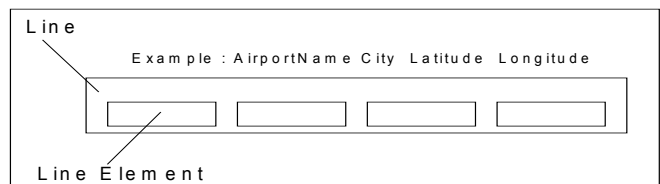


Figure 3.2. Line and Line Elements in SFML

### 3.2.1 The SFML Schema

A pseudo-XML schema based of the SFML file is depicted in Table 3.2. The top-level XML element is the SpecFormatFile tag. This element has a name attribute that represents the name of the specialize format that is being generated which is typically similar to the name of the data source. Another attribute is the name of the plug-able post processor module, if applicable. The SpecFormatFile has essentially one element that can have multiple occurrences. This element is represented by the Line tag. The Line element has four attributes. The Id and Number attributes give unique identification and sequencing for the line. The Type attribute defines rather this line is a parent line or a child line. The Type is represented by integer values 1 and 2, respectively for parent and child. If this line is a derivative line, then the ConstraintOnId attribute contains the line id that this line is depending on for information. We will discuss how this information will be used in the QueryConstraint element in later text. Finally the Filename attribute identifies where the output for the line will be printed. This tag usually contains a file name, however if “toScreen” is printed as the value then the SFG architecture knows to send the output to the screen.

The Line element contains three sub-elements. Those sub-elements are Query, NumberOfElements, and Element. The Query element represents the actual SQL query string in the QueryString element. The QueryConstraint element describes how this query can be further filtered by another query. Typically a query is constrained in the where clause by some column being equated to a particular literal value. This element describes what local database column to filter (Table\_name and Local\_name attributes) and from what line and column the information comes from (QueryConstraint element value). In addition, the Format attribute describes if this literal value is a string or number. Finally, the Type attribute tells if this filter/constraint should be attached by an AND or by an OR.

Another sub-element of the Line element is the NumberOfElements element. This element just gives the number of elements that will be included for the line. The final sub-element of Line is the Element or “LineElement” element. This element describes each data string contained in the line and maps the value back to information that should have been received from the aforementioned query. Typically, fields specified in the Select clause of the query are used to satisfy the information constraint on these “Line Elements”. This information is captured in the DBTable and DBField subelements. The Type specifies if the returned data needs to be formatted as a string or number. The StaticValue element allows researchers to put in a value that stays the same and not connected to any database returns. In building the architecture, we recognized the need to tie in special methods and algorithms. The SpecialTransform element is used to specify special formatting features such as changing time or date formats.

```

<SpecFormatFile name="DataSFGName"
  postProcessor = "PostProcessorCodeName">
  <Line id="1A" number="1" type="1" constraintOnID="None"
    fileName="OutputFileName">
    <Query>
      <QueryString> Query String </QueryString>
      <QueryConstraint format="number" type="and"
        table_name="DBTableName"
        local_name="FieldName">
        Constrained FieldName
      </QueryConstraint>
    </Query>
    <NumberOfElements>TotalElements</NumberOfElements>
    <Element number="Sequence_Number">
      <DBTable/>
      <DBField/>
      <StaticValue/>
      <Type/>
      <SpecialTransform/>
    </Element>
  </Line>
</SpecFormatFile>

```

Table 3.2 SFML in Pseudo-XML Schema Format

### 3.3 Object-Oriented Design of the SFG Component

The SFG component is the centerpiece of the new innovation to the CRS. The SFG component contains the functionality to parse the XML file and build a representation of the specialized format into the memory of the application. It contains the functionality to handle multiple inter-leaving queries. It also has the capability of converting the database returns into a format as specified in the SFML. The design of the SFG component as a UML class diagram [6] is illustrated in Figure 3.3.

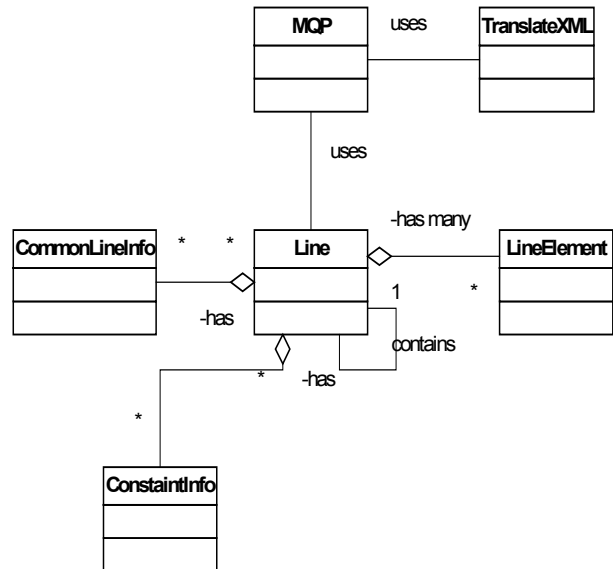


Figure 3.3. Class Diagram of SFG Component.

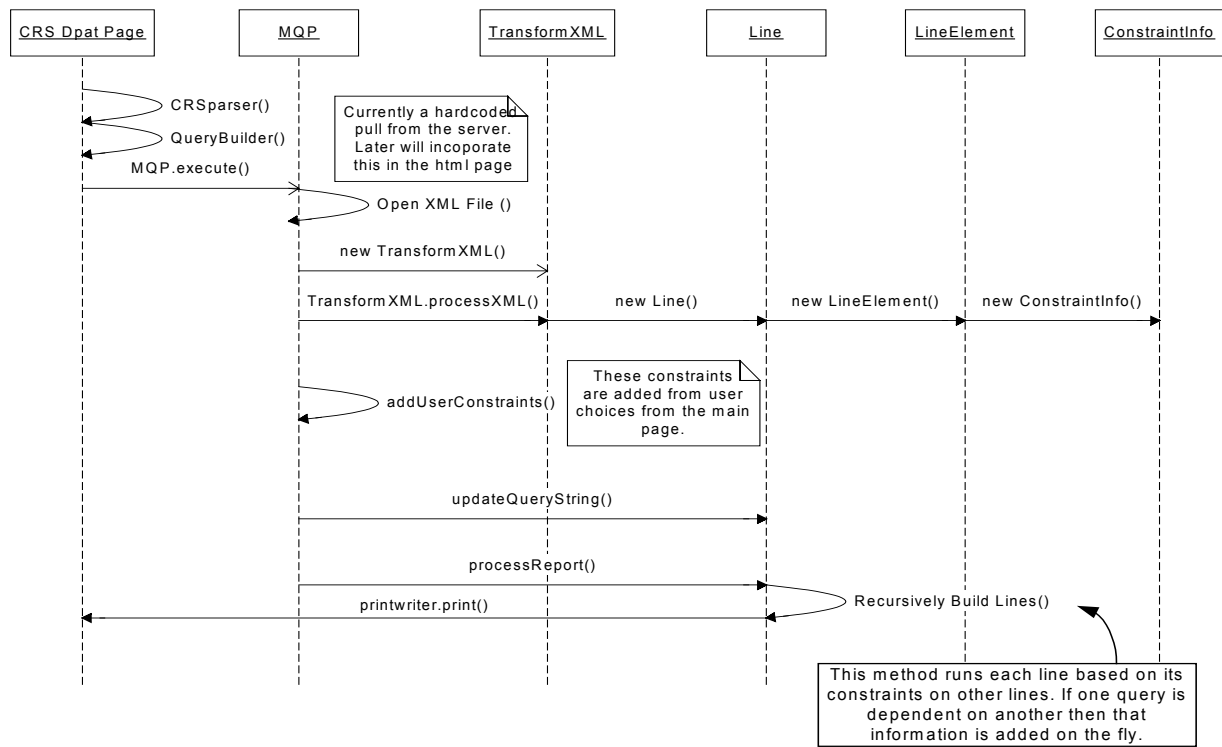


Figure 3.4. Sequence Diagram of SFG Component.

The object-oriented design of the SFG component consists of six objects. These are the MQP (Multi-query processor), TranslateXML, Line, LineElement, ConstraintInfo, and CommonLineInfo objects. The MQP contains the main control for the component. The TranslateXML object is used to interpret the format information from the SFML file at runtime. Essentially, the representation of the output format is represented in a tree of Line objects as shown by the reflexive association in Figure 3.3. The TranslateXML builds this tree and returns one Line object (i.e. the base object of the tree) to the MQP. LineElement, ConstraintInfo, and CommonLineInfo are aggregates of the line objects and hold the remainder of the information both for database processing and results processing. The MQP starts the recursive invocation of the Line objects. Each line knows how to process itself. In addition, each Line element knows how to format and print its value in a buffer prior to either printing to the screen or to a file. Since the LineElements process their data independent of other LineElements it was necessary to create a CommonLineInfo object to handle the case when one line element builds on another. The UML sequence diagram [6] in Figure 3.4 illustrates the interaction among the SFG objects when a specialized format is initiated and processed. The first action is not captured in the sequence diagram. The first thing that happens is a user specifies a particular XML file, in this case the DPAT file that will be discussed in later sections. This XML file is processed to produce a HTML-based web interface. The user then makes selections based on the quantity of information that is needed. The first two steps in the sequence diagram illustrate functionality that is currently in the CRS system. The CRSParser gathers the user information from the web interface and the

QueryBuilder method intelligently builds a query string based on that information. The next step is the invocation of the execute method in MQP. This is the main control of the SFG component. Initially, the TranslateXML object is passed a copy of the XML file that initiated the web interface. The Translate XML builds a Line object that encapsulates the entire tree for the formatting. This object contains all sub-Lines, LineElement objects, ConstraintInfo objects, and CommonLineInfo objects. Next, the MQP must gather the user information. In basic terms, the MQP strips the where clauses from the CRS generated query string and appends them to the query string included in the SFML file (i.e. updateQueryString). The final step is rather simple. Each Line object has a process method that prints its internal line of information. Each Line object first prints its line information and then recursively traverses the tree to lower elements. The architecture can support as many tree levels as specified by the user and supported by the CPU. The output is either printed back to screen or printed back to a file specified by the user in the SFML file. If there is post-processor code specified in the SFML file, then this code is executed in the MQP after the file has been printed to its destination. Post-processing functionality requires the code be printed back to a location on the server accessible to the specified post-processing module.

#### 4. SFG FOR THE AVIATION DOMAIN

The Detailed Policy Assessment Tool (DPAT) [16] is a software simulation at MITRE-CAASD that builds a queuing model that emulates the operation of the National Airspace System (NAS). The DPAT team is a good example of a case where though helpful in disseminating raw data, the CRS was only a couple steps from

benefiting their efforts greatly. Relevant to this case, the CRS provides the OAG or flight guide information. In its raw form, this information contains information about the origin and destination cities, flight numbers, and airline information for independent flights. The DPAT team easily could obtain a list of atomic flights, but, in fact, the inputs to their simulation are the itineraries. The itineraries are generated by taking all of the flights for a given timeframe and grouping them in such a way that you can see every stop a particular plane makes in that given timeframe. For example, in CRS, an analysis would be able to list all the flights for a particular airline based on the specific equipment number airplane. This would give each individual flight. The DPAT teams need that type of information in a specific format as input to DPAT. At first this seemed possible in the original CRS by changing how the query was built, but several problems were encountered as below:

1. *Sometimes airplanes were changed (mid-itinerary), so the equipment Ids were not as expected.*
2. *The output still needed to be in a specialized output that contained some static information necessary for the simulation*

Realistically, neither the CRS nor SFG could solve the first problem with just data alone. The DPAT team actually has an intermediate software module that can determine how to link atomic flights by other means in addition to the equipment Ids. This software used taxi-in/taxi-out times and flight numbers as alternatives. This intermediate software also takes a specialized format with specific static values.

The SFG was an obvious solution for this problem. In addition to the ability to format the output as input to the intermediate software, the SFG components could also run the post-processor module to generate the itineraries. A portion of the SFML file for the DPAT Itinerary Generation is illustrated in Table 4.0. It is not in the scope of this paper to discuss this file in detail, however some points are worth discussion. This SFML files shows how LineElements are populated based on the airline carrier information, flight number information, departure/arrival airport, departure/arrival times, and equipment identification information. The DPAT format required that the departure time (Line Element 4) be a function of the arrival time (Line Element 6). This is one case where the CommonLineInfo was necessary to pass information among the LineElements. In addition, the DPAT software takes a particular formatting style for the arrival/departure times. The LineElements know that these fields need special formatting by the specification of MPM or “Minutes past Midnight” in the SpecialTransform element. Also, the post-processing code is specified as “itinGen” in the top most element. At this point in development, SFG can only send the specialized output as standard input to the post-processing modules. The SFML file in Table 4.0 omits the static elements that were needed in processing this information in the consideration of space limits in this paper. The DPAT Itinerary Web Interface is illustrated in Figure 4.0.

The DPAT example does not show the true strength of the multiple query processing and formatting supported by the SFG architecture. Another research group, not discussed in the context of this paper, uses flight tracking information to build complex flight route outputs based on geographic points and latitudinal/longitudinal information. In this case, there is no post-processing, but some very complex data formatting.

```
<SpecFormatFile name = "DPAT" postProcessor = "itinGen">
  <Line id = "1A" number = "1" type = "1" fileName = "selftfile">
    <Query>
      <QueryString>
        Select OagCommon.OagDataID, OagCommon.flightNo,
        OagCommon.deptAirport, OagCommon.arrAirport,
        OagCommon.ataEquip, OagDerivedTime.arrGMTTime,
        OagDerivedTime.deptGMTTime, Carrier.ataCarr, Carrier.oagCarr,
        ROWNUM From Carrier, OagCommon, OagDerivedTime WHERE
        OagCommon.oagCarr = Carrier.oagCarr AND
        OagDerivedTime.oagdataid=OagCommon.oagdataid </QueryString>
      </Query>
      <NumberOfElements>10</NumberOfElements>
      <Element number = "1">
        <DBTable>Carrier</DBTable>
        <DBField>oagCarr</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>Regular</SpecialTransform>
      </Element>
      <Element number = "2">
        <DBTable>OagCommon</DBTable>
        <DBField>FlightNo</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>Regular</SpecialTransform>
      </Element>
      <Element number = "3">
        <DBTable>OagCommon</DBTable>
        <DBField>deptAirport</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>Regular</SpecialTransform>
      </Element>
      <Element number = "4">
        <DBTable>OagDerivedTime</DBTable>
        <DBField>deptGMTTime</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>MPM</SpecialTransform>
      </Element>
      <Element number = "5">
        <DBTable>OagCommon</DBTable>
        <DBField>arrAirport</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>Regular</SpecialTransform>
      </Element>
      <Element number = "6">
        <DBTable>OagDerivedTime</DBTable>
        <DBField>arrGMTTime</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>MPM</SpecialTransform>
      </Element>
      <Element number = "7">
        <DBTable>OagCommon</DBTable>
        <DBField>ataEquip</DBField>
        <StaticValue>Empty</StaticValue>
        <Type>String</Type>
        <SpecialTransform>LINEBREAK</SpecialTransform>
      </Element>
      *** MANY OTHER STATIC FIELDS ***
    </Line>
  </SpecFormatFile>
```

**Table 4.0** SFML File for DPAT Itinerary Generation

**CAASD Repository System  
DPAT Itinerary Generation Form**

*Time Selections*

**Start Date & Time**  
 Month/Day/Year  Hours:Minutes

**End by:**  
 Date & Time: Month/Day/Year  Hours:Minutes   
 Duration: Months  Days  Hours  Minutes  Seconds

**Options**  
 Time entered based on event   
 Start time entered in

*Country Codes*

**Originating Airports:**  OR  OR

**Destination Airports:**  OR  OR

**Figure 4.0.** Web Interface for the DPAT Itinerary Generation using SFG Architecture.

## 5. RELATED PROJECTS AND TOOLS

Though there is a great deal of existing work in software architecture [8] and reconfiguration of software architectures [1][12], as discovered by this author, there is not much research towards web-to-database architectures. The Zelig project introduces a schema that can be coupled with HTML to control various CGI-based database executables [13]. The main difference is that the Zelig project appears to tightly-couple the interface with the database by hard-coding entire query strings in the HTML documents. The CRS architecture extends this approach by incorporating intelligent query building knowledge in the middle-level components. The CRS interface has aspects of the query, while the middle-level components encapsulate the generic intelligence that builds query strings dynamically. The SFG architecture, indeed, hard-codes portions of the query, but these queries can be generated with tool support. Also, SFG queries are dynamically extended with filter information using the CRS intelligence. In addition, the CRS/SFG architecture has the flexibility to allow additional business-specific components to be plugged in to further format and process the database results. Moreover, the CRS implementation makes use of the later, more flexible technologies of XML (for SFML) and Java Servlets. Another project that is even closer to the CRS architecture is the WebInTool [10][14]. In the WebInTool, Hu specifies a web to database interface building tool. Hu takes a similar approach as the CRS/SFG architecture in promoting a separation of interface and source code. Hu uses several CGI-based modules but similar

to the ZELIG project, there is no intelligent query building knowledge in these modules. In the absence of this feature, the WebInTool does not have the same ability as the CRS implementation to dynamically create a large range of queries. In addition, the SFG architecture supports a larger range of output choices with respect to formatting.

There are also industry tools that have rapid web-to-database development tools. Such tools as Oracle's WebDB [11] and Crystal Reports[7] allow developers to build graphical user interfaces that construct formatted database reports. These tools are similar to SFG, but they do not use universally accepted specification methods. Each of these corporations have more complex proprietary methods to specify formats. These tools also do not allow the ability for other modules to be seamlessly plugged in.

## 6. DISCUSSION AND FUTURE WORK

In this work, we introduce a new paradigm and architecture for distributed data formatting, particular that data that is retrieved from a relational database. We discussed an implementation that would support this architecture using web-based technologies and the use of XML-based language that we conceived called SFML for format specification. This implementation has been highly successful in supporting software simulation input files that can be derived from database information but have cryptic formats.

There are many areas of future work that were discovered. The major area is for performance. Performance results were not provided in context of this paper, because the performance was

more than effective for the CAASD domain. Most files running these software simulations are only about 3 megabytes, in size. The SFG easily produces these files in less than 10 minutes for up to 5 levels of chained queries. However, if this approach is used to build files 100 times this size with more chained queries, performance can indeed become an issue. Since each returned row is formatted independently and sequentially, future work may consist of a more parallel processing approach. In addition to performance issues, we are constantly extending the architecture for additional formatting features (*SpecialTransforms*). In addition, SFML has on-going work that extends the schema to address newly discovered file format constraints.

## 7. ACKNOWLEDGEMENTS

There was a great deal of support given by members of both the MITRE-CRS team and the MITRE-DPAT team consisting of Fred Wieland, Gail Hamilton, Jeffrey Hoyt Ali Obaidi, Tho Nguyen, Ted Cochrane, Jay Cheng, Ali Obaidi, Dave Milner, Len Wojcik, and Lisa Schaefer.

This is the copyright work of the MITRE Corporation and was produced for the U.S. Government under Contract Number DTFA01-93-C-00001 and is subject to Federal Acquisition Regulation Clause 52.227-14, Rights in Data-General, Alt. III (JUN 1987) and Alt. IV (JUN 1987). The contents of this document reflect the views of the authors and The MITRE Corporation. Neither the Federal Aviation Administration nor the Department of Transportation makes any warranty or guarantee, expressed or implied, concerning the content or accuracy of these views.

M. BRIAN BLAKE is a faculty member in the department of computer science at Georgetown University and a lead software system engineering consultant with The MITRE Corporation. He received his Ph.D. in Information and Software Engineering from George Mason University with an emphasis on systems and software engineering. His research interests encompass the domain where software engineering approaches can be used for the development and realization of information systems. Dr. Blake has authored more than 30 papers in journals and refereed conference proceedings in the general area of agents and information management systems.

## 8. 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 for the Consolidation of Enterprise-Wide Data-Oriented Simulations" Workshop on Agents In Industry at the 4th International Conference on Autonomous Agents (AGENTS2000), Barcelona, Spain, June 2000
- [3] Blake, M.B. and Liguori, P. "An Automated Client-Driven Approach to Data Extraction using an Autonomous Decentralized Architecture" 5th International Symposium of Autonomous Decentralized Systems (ISADS2001)/IEEE Computer Society Press, 311-319, Dallas, TX March 2001
- [4] Blake, M.B. and Liguori, P. "An Autonomous Decentralized Architecture for Distributed Data Management and Dissemination" *IEEE/IEICE Transactions on Information and Systems Joint Special Issue on Autonomous Decentralized Systems*, Vol. E84-D, No.10, pp 1394-1398, October 2001
- [5] Blake, M.B., Hamilton, G., and Hoyt, J. "Using Component-Based Development and Web Technologies to Support a Distributed Data Management System", *Annals of Software Engineering* , Vol. 13, No. 1, April 2002, Kluwer Academic Press (in press)
- [6] Booch, G., Rumbaugh, J., Jacobsen, I., "The Unified Modeling Language User Guide", Addison Wesley, Reading MA, 1998
- [7] Crystal Reports (2002) <http://www.crystaldecisions.com/products/crystalreports/>
- [8] Garlan, D. and Shaw, M., *Software Architectures, Perspectives on an Emerging Discipline*, Prentice Hall, 1992
- [9] Hoyt, J.C., "MRALD:Delivering Data – Any Data, Any Way, Anywhere" *Foundations of Software Engineering (ACM SIGSOFT 2002)*
- [10] Hu, J., D. Nicholson, C. Mungall, A.L. Hillyard, A.L. Archibald, " WebInTool: A Generic Web to Database Interface Building Tool," In *Proceedings of the 7th International Conference and Workshop on Database and Expert System Applications (DEXA96)*, IEEE Computer Society Press, Zurich, Switzerland, 1996
- [11] Oracle Corporation (2002), *WebDB Application 3.0* <http://oradoc.photo.net/ora816/webdb.816/a77075/basics.htm>
- [12] Shrivastava, S. and Wheeler, 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
- [13] Varela, C.A. and C.C. Hayes, " Zelig: Schema-Based Generation of Soft WWW Database Applications", In *Proceedings of the 1st International Conference of the World Wide Web (WWW94)* Elsevier Science, Geneva, Switzerland 1994.
- [14] WebInTool(1997), <http://www.ri.bbsrc.ac.uk/webintool.html>
- [15] Weiss, A. "XML gets down to Business," *Networker*3,3 pp 36-37, September 1999
- [16] Wieland, F., "Limits to Growth:Results from the Detailed Policy Assessment Tool" , *Proceedings of the 1<sup>st</sup> USA/Europe Air Traffic Management Research and Development Seminar*, Sarclay, France, June 1997