Enterprise Data Management in Research Organizations:
Data the Way You Want It

M. Brian Blake

Center for Advanced Aviation System Development
The MITRE Corporation (CAASD)
7515 Colshire Drive N420
McLean, VA 22102-7508
bblake@mitre.org

Department of Computer Science
Georgetown University
234 Reiss Science Building
Washington, DC 20057
blakeb@cs.georgetown.edu

ABSTRACT
Raw data and processed information are essential to organizations that perform operational analysis and build simulation systems. In such 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. To achieve maximum utility, data of this sort must be made available in distributed locations and offered in various custom formats. Such approaches as relational-to-XML, XML-XSL-based custom formats, and web-accessible database reporting tools offer some solutions for this domain. However, there are some requirements that the current state of the art do not fulfill. In this paper, there is a characterization of the state of the art for this distributed data management domain and a discussion of the current short-comings.

Categories & Subject Descriptors
H. Information Systems, H.3 Information Storage and Retrieval, H3.4. System and Software

General Terms
Design, Languages

Keywords
Semi-structured Data, XML, XSLT, web-accessible databases

1. INTRODUCTION
In some enterprise organizations, it is common that data is shared across multiple underlying groups and teams. This is especially true in organizations that perform analysis on specialized domains, such as Air Traffic Management (ATM), Command, Control and Communication (C3), Business Process Modeling, and Neuroinformatics. At times, the analysis efforts of multiple teams are interrelated and the same sets of data are used for different computational tasks.

A benefit to enterprises with respect to saving staff time is the reuse of these common data sets. This reuse can be particularly challenging when data is captured from domains in heterogeneous formats. Compounding this problem is the fact that internal analysis teams may adopt differing formats.

An obvious solution for this problem is the insertion of some universal data schemas and representations. Many technologists think that the Extensible Markup Language (XML) is the solution to the aforementioned challenges [19][20]. Other technologies using XML for formatting and translation are the Extensible Stylesheet Language (XSL) [23] and the Extensible Stylesheet Language Transformations (XSLT). Also, many relational database management systems (RDBMS) provide reporting tools that can process data in formats acceptable to a set of heterogeneous users. However, we have found that in some simulation development domains, the current technologies and tools for producing *specialized formats* are inadequate to meet the current needs.

The next section specifies the needs identified in organizations, where tools are needed for advanced data management and dissemination. The following section discusses the underlying aspects of these needs and using a survey of current technologies, including the previous work of the author, shows what needs are currently not met. In Section 4, we discuss are current architecture work that supports this domain and alleviates the aforementioned limitations.

2. DEFINING DATA MANAGEMENT AND DISSEMINATION
Researchers typically use simulation systems for both design-time and real-time analysis. Both the raw data and processed data must be shared at an enterprise level. In this section, the requirements surrounding data management and dissemination in this domain are discussed.

2.1 Data Management Requirements
Although the underlying groups in these types of organizations analyze different problems, the data to support the investigations are typically the same. Also, software engineers in these individual groups design and develop software simulations that require the data in different formats (i.e. specialized delimited text files, database format, XML [19], etc.) Moreover, each group looks at different subsets of data that may cross multiple data sources. The need for enterprise data management is exemplified in the identification of two

major problems. One problem discovered is that typically data sources are acquired from external sources thus stored independently without any connection to other possibly similar data sources. A second problem is that groups, that use the same information, duplicated their efforts in parsing data out of stored formats to generate more acceptable formats.

The obvious solution is the incorporation of a relational database management system (RDBMS). However, there are several caveats. In some organizations, software engineers/researchers that developed these simulations are not regular database users and have no interest in becoming experts in the latest RDBMS technologies. These researchers tend to be domain specialists. In addition, the software engineer/researchers are dispersed across decentralized sites using heterogeneous operating environments. In these cases, there is the need for a data management architecture that offers distributed, user-friendly connectivity.

2.2 Data Dissemination Requirements

Another set of problems surrounds the fact that software simulations almost never use common formats like delimited text, XML, or Microsoft Excel as input formats. The pre-existing format requirements tend to be 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. Such domains as financial management and aviation information systems acquire data from legacy mainframe systems. Ironically, in fact, the data returned from conventional technologies can be *too clean* to be accepted by most of the pre-existing simulations. Moreover, researchers tend to have their own pre-processing modules. These pre-processing modules further process the initial raw data into more of a *story* that software simulations can electronically enact. Finally, these simulations need initial data that is the result of the integration of multiple data sources. Data sets need to be generated where several layers of queries are specified and the result set of one query could feed the input of other queries.

2.3 Requirements for a Distributed Data Management and Dissemination Domain

In this work, the effort is toward an architecture (Specialized Format Generation Architecture (SFG)) that alleviates both needs for data management and for distributed data dissemination. The requirements discussed in the previous sections can be summarized in Table 1.

Table 1. Requirements of Distributed Data Management and Dissemination Domains

1. Architecture must support the distributed dissemination of data.
2. Data sources need to be consistently stored and accessible.
3. Architecture support for specification-based data return formatting (at times, from RDBMS).
4. Architecture must allow modules to be plugged in
5. Architecture must handle multiple interconnected queries while returning the data as in (3)

The CAASD Repository System (CRS) [3] was previous work that handled the first two requirements listed in Table 1. The CRS is a framework developed by The MITRE Corporation to store multiple sources of aviation data in a common relational database. This framework is comprised of a relational database management combined with a web-based user interface. The CRS supports both the loading of data in the database in addition to the ability to deliver data in certain pre-defined formats. As it was initially designed, the requirements alleviated by the CRS lie more in the data management aspects of the architecture. The short-comings presented is this paper has a focus on the specification-based dissemination requirements not supported in the CRS and listed as 3,4, and 5 in Table 1. The requirements for this type of dissemination can best be illustrated in a use case diagram [4] showing the high-level functionality that such an architecture should be able to achieve.
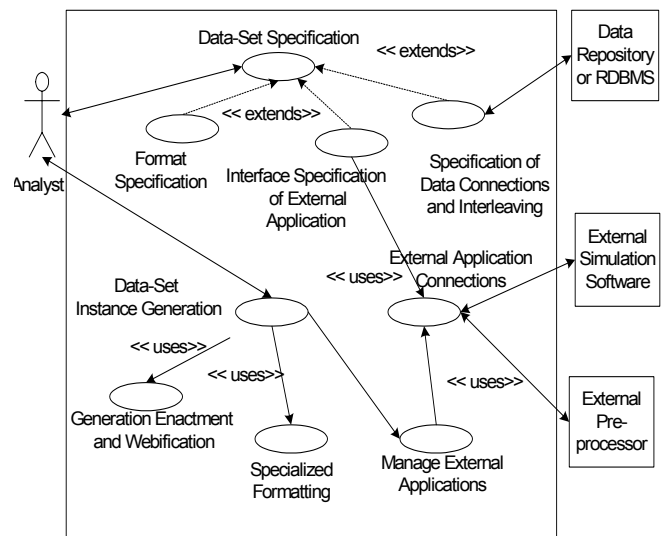

Figure 1. Use Case Diagram of Distributed Data Management and Dissemination

The two major functionality requirements detailed in Figure 1 are the need for specifying data-set format and the actual generation of data instances. The data specification is separated into the specification of the format, how data-sets are connected, and the relation of the data-set to external applications. The generation functionality uses the aforementioned specification information for producing and formatting web-accessible data-sets. Each of the major functions must interact with some data repository and external applications (i.e. pre-processors and simulations). The work presented in this paper is toward an architecture to fulfill these major requirements.

3. Related Work

There is no shortage of research projects and industry tools that provide approaches to meet the needs in this domain. In this section, we scope the plethora of tools and techniques that are available in this domain. It would be impractical to attempt to present all the available tools in the scope of this paper, however many representative tools and approaches are discussed here. The two major areas related to this domain are web-accessible database technologies and automated architectures using XML and XSLT. Finally, based on

represented tools and techniques, we discuss the group of requirements not met in these current relevant technologies.

3.1 Characterizing the State of the Art

With respect to all technologies surveyed, there is a natural composite path that shows how all approaches fit into the scheme of distributed database management and dissemination. This composite path, with sub-paths that represent the various existing technologies (as in Figure 2) shows the excessive number of approaches that are considered part of this domain.

In Path 1, data retrieved from a relational database is represented back to the user in XML. Extracting XML from relational formats is not new. Most RDBMSs currently support the return of relational data in XML formats. One requirement, however, is the creation of an XML schema (also referred to as Document Type Definition (DTD) and more recently XML Schema Definition language (XSD)) that conforms with information that will be returned from the database. In addition, there has to be a mapping from the database to this XML schema. There are a number of other tools and research projects that specialize in relational to XML mapping. One such tool, XML Lightweight Extractor (XLE) [21], has a graphical user interface allowing users to make mappings from an XML document to a relational format using a mapping file called DTDSA. XML-DBMS [22] is a middleware for transferring data between XML documents and a RDBMS. Again, an XML-based mapping language is used similar to that of DTDSA. In Path 2, using XSL and applications for XSLT can be used to translate XML documents into any format. This combination of XML and XSL is a commonly accepted approach to the area of data dissemination and specialized formatting generation.

Technologies in Path 3 are toward web-accessible database tools that allow for results reporting. One project, the SilkRoute [8] of Fernandez [9], achieves both the web-accessible database capabilities while delivering data in XML format. Other projects do not particularly use XML/XSLT approaches. The Zelig project introduces a schema that can be coupled with HTML to control various CGI-based database executables [26]. In the WebInTool [11][18], Hu specifies a web to database interface building tool. This approach promotes the separation of interface and source code. Hu uses several CGI-based modules but similar to the ZELIG project, there is no automated query building knowledge in these modules. The original work of the authors in the CRS [3] is similar to the Zelig and WebInTool projects. The major difference is that the architecture uses a middleware of reconfigurable objects that can autonomously build the relational query from user input. This approach supports prior work toward the reconfiguration of software architectures [1]. Similarly, Cooper [5] devises a middle-tier architecture for data management called eXtensible Data Management (XDM). This approach uses XML requests to a component-based middle-tier to access multiple databases. The main issue covered in this work is toward connecting multiple databases.

There are also industry tools that have rapid publishing of relational databases on the web. Such tools as Oracle's WebDB [16] and Crystal Reports [6] allow developers to build graphical user interfaces that construct formatted database reports. Queries are built dynamically at design time, so forms must be rebuilt when the schema changes. In Path 4 and 5a, there are several projects that automate the integration of heterogeneous formats, or *any-to-any* translations. This area is pertinent to automating the connectivity of distributed applications as in translating messages among on-line businesses (business-to-business (B2B) interoperability). One product, the IBM WebSphere Data Interchange [12], is an example of Path 4. The TSIMMIS project [10] and Enosys Software [7] both integrate applications with databases.

Other related research projects, not represented in the Figure 2, explore how XML files can be queried. Research in this area is definitely valid as more and more data is being represented in XML. Leslie [15] explores technologies for querying and the transformation of XML document types, and Petropoulos [17] has interfaces for XML query enactment. However, they do not connect to relational data models and have fairly specific approaches to data transformations that may not be flexible enough for the complex simulation formats.
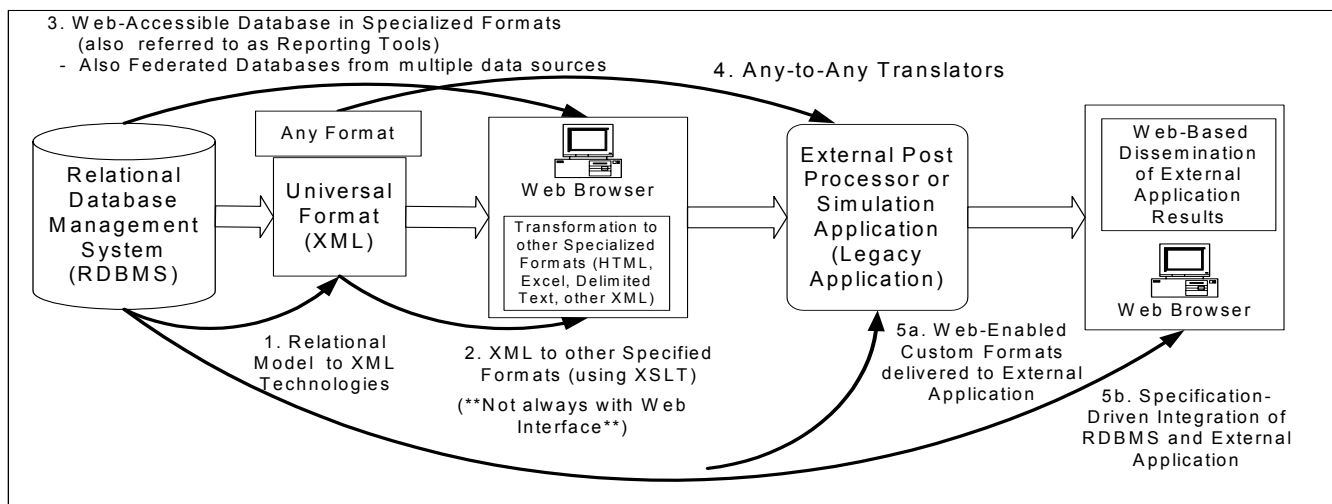

Figure 2.. Main Distributed Data Management and Dissemination Path and Subpaths

3.2 Limitations in Leading Related Work

One path not discussed in the earlier section is Path 5b. There is not one tool or technology that manages all three main requirements as illustrated in Path 5b. Those requirements are listed below.

1. Specification of relational data (formatting) and automated web interfaces for the entry user constraints

2. Managing the transfer of resultant data into external applications

3. Presenting the resulting database information or external application output within a web form.

Either a combination of Path 1 and 2 or solely Path 3 can be extended to handle this functionality. However, there are problems that make these approaches inadequate. The combination of Path 1 and Path 2 is the extensible solution, because the XML resulting from Path 1 can be translated into various future formats easily using XSL (Path 2). However, there would be greater performance overhead in the XML/XSL solution than the single technologies of Path 3, particularly for large files. This is mainly because the information must be generated twice, first in XML format, then in the final format. In addition, this combination solution does not inherently support web-published results. Reporting tools, as in Path 3, have the functionality to query the database and generate the destination files simultaneously. However, a problem with web-based reporting tools are their inadequacy in producing extremely cryptic formats with complex relational query interleaving. The focus of these applications are to present table information and statistics from relational data. In addition, the tools use proprietary languages that limit their extensibility. Also, technologies in Path 4 do not directly support the needs defined in this domain. These tools more generally operate on multiple databases or federated data sources. These technologies are excellent for integrating data, but do not particularly support the requirements in this domain to present resulting data to the user. Other problems with the above approaches are the need for in-depth XSL knowledge, which is not the reality in traditional research organizations where non-computer specialists are focused on other more operational knowledge.

4. SPECIALIZED FORMAT GENERATION

The Specialized Format Generation architecture (SFG) and supporting specification language, Specialized Format Markup Language (SFML), were devised to handle the requirements illustrated in Figure 1 and needs discussed in Section 3.2. The SFG architecture extends technologies used in the CRS approach. Consequently, a major goal is to promote the separation of the interface and the software implementation. In order to support this separation, we promote a specification-driven approach. The two major *contributions* of this approach are the SFML language and the underlying SFG component. We created one language, SFML, to specify all four concerns of interface specification, query specification, results formatting, and external application connectivity. The SFG component uses a recursive, object-oriented design to manage the enactment of the operations specified in SFML.

This section continues with an overview of the SFG architecture. In following sub-sections, there is a description of the semantics of SFML. We show how these semantics can be used for the four aforementioned concerns. In the concluding sub-sections, we show a screen-shot of the SFG graphical user interface and how that interface is used to create SFML.

4.1 SFG Overview

The first step in the operational flow the SFG approach is the generation of a user-specific SFML file. Expert users should be able to construct this file manually. However, most of the users (non-developers) will use a tool that handles the database specific aspects which will be discussed in Section 4.4. In this case, the user would access a local SFML builder tool that connects to the database and allows the user to build a file based on the user's domain-specific data and the database schema and fields. The system provides an interface where SFML files can be uploaded to a common repository. Users can then access this file. The new system incorporates an XSLT component that converts the file into a HTML web interface. During this conversion process, the web interface that is dynamically generated includes specific retrieval templates. These templates allow the user to enter additional information that can be used to constrain the database results. The SFG architecture allows for the interleaving of database queries to produce the specialized output. Moreover, the output can be provided to the standard input of the user-specified external applications or post-processors. The final data is returned to the browser. If no external application is specified, the specialized output is directly written to users' client or to a user-specified file location.

4.2 SFML Overview

The main purpose of the SFML file format, from a user's perspective, is to specify how they want their database query results to appear. 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, similar to the return of rows when querying a relational database. As such, these unique lines can be described in terms of database row returns. Therefore, the SFML files connect output formatting instructions to database returns. In most database queries supporting simulation software, there is some *base* filter that constrains the query by such qualifiers (or dimensions) as quantity, time, or location. 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. However, multiple main (peer-level) lines can be specified. In each line, there is a specification of where the output will go. Line information can be presented in the users' browser, to a file, or to multiple files as specified in the SFML. This is helpful in building multiple output files from one SFML file and web interface.

Lines are represented as a set of line elements. A Line element is mapped to a specific column that is returned at the completion of a query. Special formatting can be applied to line elements. 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 4. This example shows how a unique line of data such as "Employee Information" can be separated into four data fields. These data fields are sequenced as line elements.
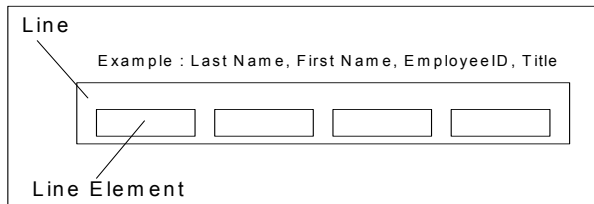

Figure 4. Line and Line Elements.

4.3 SFML Schema

A pseudo-XML schema (an XML file with descriptive schema information) based on the SFML file is depicted in Table 2. The top-level XML element is the *SpecFormatFile* tag (Line 1). The *SpecFormatFile* element has a *name* attribute that represents the name of the specialized format that is being generated. At times, this name is similar to that of the data source. Another attribute, *PostProcessorCodeName*, is the name of the external application or post processor module, if applicable. The *SpecFormatFile* has one element that can have multiple occurrences. This element is represented by the *Line* tag (Line 2). The *Line* element has five attributes, *id*, *number*, *constraintOnId,* and *filename*. The *id* and *number* attributes give unique identification and sequencing for the line (Line 3). The *type* attribute specifies whether this line is a main line or a derivative line. The *type* is represented by integer values 1 and 2, respectively for main and derivative (Line 3) lines. If this line is a derivative line, then the *constraintOnId* attribute contains the *id* of its main line. 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 user's browser.

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 (Line 5). The *QueryConstraint* element describes how this query can be further constrained 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 the local database column to constrain (*table_name* and *local_name* attributes) (Lines 7 and 8) 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* (Line 6) 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 (Line 10). This element gives the number of elements that will be included for the line. The final sub-element of Line is the *Element* or "Line Element" element (Line 11). 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* sub-elements (Line 12 and 13). 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.

Table 2. SFML File with Inserted Schema Information

```
1.   <SpecFormatFile  name="DataSFGName"      postProcessor  =
     "PostProcessorCodeName">
2.     <Line id="1A" number="1" type="1" constraintOnID="None"
3.   filename=" OutputFileName">
4.        <Query>
5.           <QueryString>  Query String </QueryString>
6.           <QueryConstraint format="number"   type = "and"
7.                           table_name="DBTableName"
8.                           local_name="FieldName">
     ConstrainedFieldName </QueryConstraint>
9.        </Query>
10.      <NumberOfElements>TotalElements</NumberOfElements>
11.      <Element number="Sequence_Number">
12.         <DBTable/>
13.         <DBField/>
14.         <StaticValue/>
15.         <Type/>
16.         <SpecialTransform/>
17.      </Element>
18.    </Line>
19.  </SpecFormatFile>
```

4.4 How SFML Incorporates Four Concerns into One Language

As stated earlier, an innovation of SFML is the ability to combine several concerns into one language. Other related approaches separate these concerns into multiple specifications. These related approaches are extensible but impractical in domains where users are not familiar with these languages. In such domains as the MITRE domain, it is more practical to limit the number of technologies that users must adopt. One commonly-adopted approach, using an XML/XSLT, is illustrated in Figure 5. This approach requires users to develop technologies and learn languages in 4 distinct areas. For the first concern (1), there must be a method by which a

user-specific parameter form must be created for a particular scenario. There are a number of technologies that offer programmatic support, such as Java Servlets, Java Server Pages (JSP), Active Server Pages (ASP) [13][14]. that allow the creation of such forms. However, these approaches are not evolvable solutions. It is impractical to use these approaches to develop a new form each time a new scenario is conceptualized. Using SFML and XSLT, we have created an evolvable approach, which will be discussed in greater detail in the following section.
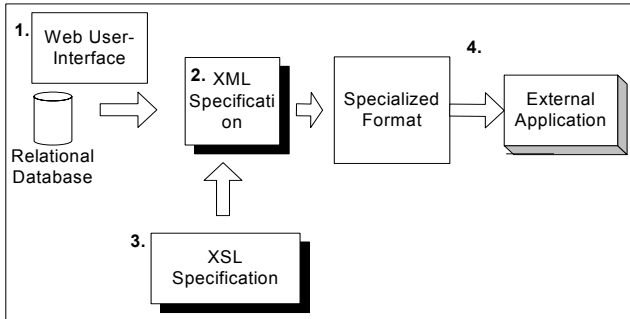

Figure 5. Four Concerns with SFML.

The second and third concerns (2 and 3) require users to develop a XML schema (DTD or XSD) that is general enough support enterprise-wide relational database retrieval. In addition, a number of specific XSL specifications must be created to support multiple scenario-specific formatting. Consequently, there is some difficulty in creating an XML schema that is general enough to support the differing needs at an enterprise level. Also, as discovered in the MITRE-CAASD domain, XSL was determined to be a difficult language for the users to learn. Moreover, there are no XSL tools with graphical support for specifying the transformation, as known by the author. The final concern (4) is a specification for the manipulation of the results so that they can be provided to external applications or presented back to the user in their browser. Concerns 2, 3, and 4 can be supported inherently with the semantics of the SFML language. Query specification, interleaving, results formatting, and external application integration can all be specified in the language. The SFG component combined with the initial CRS architecture handles the *webification* of these processes.
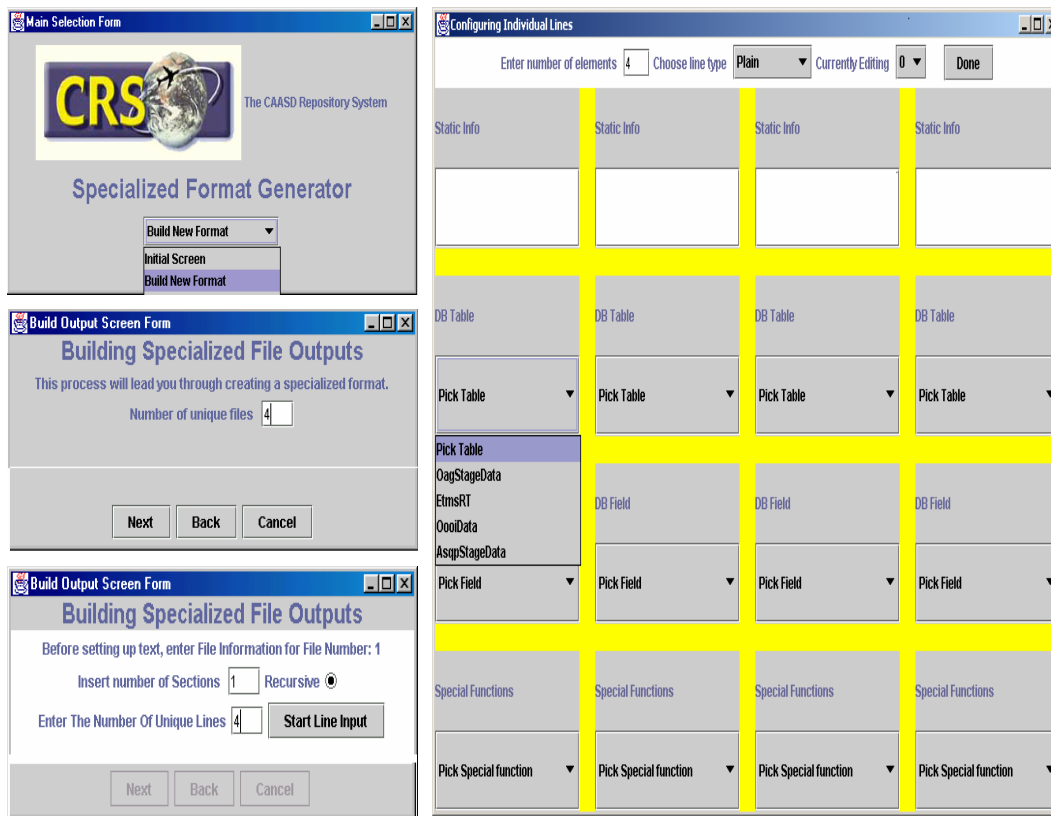

Figure 6. SFML Generation Tool.

4.5 The SFML Graphical User Interface

For experienced users, the SFML format can be generated manually and uploaded to the system. Since the SFML language is relatively involved, it can be generated using a graphical tool. This client-side application, as shown in Figure 6, was developed to prevent human error by automating the construction and upload of the SFML files. The first screen (top left) allows users to choose to create a new form, edit an existing form or run a test report. The test report feature is subset of the functionality available on the server-side. This sample report shows a sample expected output format. The next two screens directly below the aforementioned screen allow the user to specify the number of unique files, sections, and lines. The final screen on the right is where the user connects formatting instructions with the database tables and columns. Since this application connects directly to the database, the user is dynamically provided with the available columns. An additional benefit is that administrators of the database can limit the users by restricting table access. This will help assure users access the correct tables. The applet uses internal query-building components as created by the initial CRS application. In this way, the user specifies the desired columns, and the system automatically creates the query constraints required for table joins. The final SFML file can be tested, edited, and uploaded to the system.

5. AN APPLICATION OF SFG/SFML

To demonstrate and validate the effectiveness of the SFG approach, we show an operational usage of the system. The Total Airspace and Airport Modeler (TAAM) is a simulation modeling tool for airspace and airport environments. One input to the TAAM simulation is the *flight traffic file*. When airplanes fly from one airport to another, air traffic management messages and radar data is stored in a set of database tables produced from the Enhanced Traffic Management System (ETMS). The information is stored on the basis of individual flights. The database model for the ETMS data relevant to the TAAM simulation is illustrated in Figure 7a. For each flight record, there is ETMS-specific information about the messages passed to that flight. TAAM traffic files must first query the flight table then use the flight id to access information in other tables. This information must be returned in the specialized TAAM format (as represented in Figure 7b). A small subset of the TAAM SFML file is shown in Figure 8.

In the consideration of space, a full description of the mapping from SFML file (Table 3) to the output file (Figure 7b) is not possible. However, the subset of these files should show the ability of the SFG to handle complex simulation input files with multiple interleaved queries. One example, Line 2A receives flight information from the query in Line 1A as designated by the *constraintOnID* tag. Therefore, for every unique flight determined in Line 1A (Flight Table), there will be a new tracking data record for that flight as specified in Line 2A (ETMSCommon Table). Similarly, for each of the tracking data records generated in 2A, Line 3A is used to determine the current altitude (ETMSFZ). Another action from this SFML File (not depicted in Table 3) is that for each tracking record, a routing information record is extracted (ETMSRT). Finally for each routing record (ETMSRT), there is a list of waypoints (latitude/longitude points) records generated related to points through which the plane has flown (RTWayPoint).
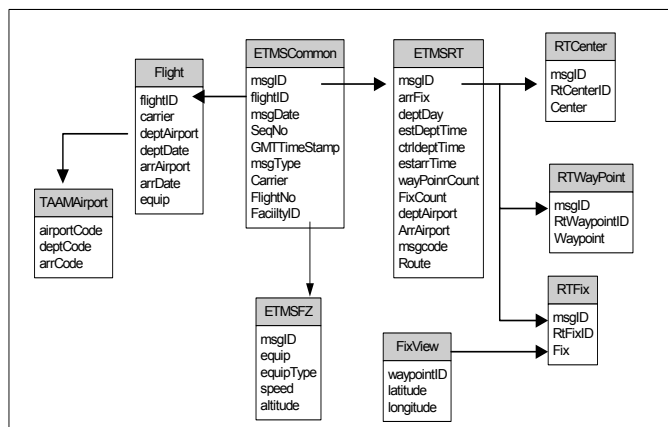

Figure 7a. Flight-Based Relational Model

Figure 7b. TAAM Input File Format

Table 3. Raw SFML File for TAAM Application

```
<SpecFormatFile name = "TAAM">
<Line id = "1A" number = "1" type = "1" fileName = "TimeTable">
<Query>
<QueryString>SELECT DISTINCT flight.flightid, flight.carrier, flight.flightno, flight.deptairport, flight.arrairport FROM flight</QueryString>
</Query>
  <NumberOfElements>1</NumberOfElements>
</Line>
<Line id = "2A" number = "2" type = "2" constraintOnID="1A" fileName = "TimeTable">
<Query>
<QueryString> SELECT DISTINCT flight.flightno,dept.icaocode deptcode, arr.icaocode arrcode, flight.deptdate, flight.arrdate,
flight.flightid, flight.carrier, flight.equip, etmscommon.msgid, ROWNUM FROM etmscommon,etmsrt, flight, taamairport arr, taamairport
dept WHERE etmscommon.msgid = etmsrt.msgid and flight.flightid = etmscommon.flightID and flight.deptairport = dept.airportcode and
flight.arrairport = arr.airportcode and ROWNUM = 1</QueryString>
<QueryConstraint format = "number" type = "and" table_name = "Etmscommon" local_name = "flightId">flightId</QueryConstraint>
</Query>
  <NumberOfElements>8</NumberOfElements>
<Element number = "1">
    <DBTable>Empty</DBTable>
    <DBField>Empty</DBField>
    <StaticValue> {</StaticValue>
    <Type>String</Type>
    <SpecialTransform>LINEBREAK</SpecialTransform>
  </Element>
<Element number = "2">
    <DBTable>Flight</DBTable>
    <DBField>Carrier</DBField>
    <StaticValue>Empty</StaticValue>
    <Type>String</Type>
    <SpecialTransform>NOSPACE</SpecialTransform>
</Element>
<Element number = "3">
    <DBTable>Flight</DBTable>
    <DBField>FlightNo</DBField>
    <StaticValue>Empty</StaticValue>
    <Type>String</Type>
    <SpecialTransform>Regular</SpecialTransform>
</Element>
*** 5 OTHER LINE ELEMENTS ****
</Line>
<Line id = "3A" number = "3" type = "2" constraintOnID="2A" fileName = "TimeTable">
<Query>
<QueryString> SELECT  altitude FROM etmsfz,etmscommon, flight WHERE etmscommon.msgid = etmsfz.msgid(+) and flight.flightid =
etmscommon.flightID and ROWNUM = 1</QueryString>
<QueryConstraint format = "number" type = "and" table_name = "Etmscommon" local_name = "flightId">flightId</QueryConstraint>
</Query>
  <Element number = "1">
    <DBTable>EtmsFz</DBTable>
    <DBField>Altitude</DBField>
    <StaticValue>0</StaticValue>
    <Type>String</Type>
    <SpecialTransform>DEFAULT</SpecialTransform>
  </Element>
</Line>
*** 4 OTHER INTER-RELATED LINES *****
```

This SFML file for TAAM Traffic files demonstrates the ability for the SFG implementation to support five levels of chained queries. Information is shared among lines from multiple levels. The method for chaining queries is similar to sub-queries in SQL. As such, these queries have far less overhead than making joins on the tables, especially in cases when small amounts of information are needed. Also in the TAAM specification, the main-line is mainly for information purposes where resultant information is used in sub-lines and printed as values in the sub-lines. This is important when there is a need for auxiliary information. For TAAM, all information is printed to files as opposed to streaming the information into the simulation. However, lines are sent to multiple files. In fact, three different files are generated from one TAAM SFML file. Finally, this file inserts a great deal of static information that is necessary as instructions to the TAAM simulation. Static text, such as "@A" and "@W", are embedded as necessary for TAAM operation.

6. DISCUSSION

In this paper, we introduce a new architecture for distributed database dissemination, particularly that data that is retrieved from a relational database. This new SFG architecture served as an enhancement to the initial work of the CAASD Repository System (CRS). We discussed the implementation of this architecture using web-based technologies and the XML-based, SFML. Through a survey and comparison of existing work, we showed how the requirements of this dissemination domain are not currently met in related projects and tools. Existing research projects and tools only achieve partial support (i.e. XML transformation, relational-to-XML transformation, or database reporting, but not all three). This implementation has been highly successful in supporting software simulation input files that can be derived from database information but have cryptic formats. In this paper, we evaluate this architecture and implementation by integrating it in the TAAM domain.

We have highlighted the benefits of using one file to specify multiple concerns. In this domain, one centralized file is logistically easier to handle. There is only one specification language to learn and one file to store and manage per project. In addition, using an XML-based approach greatly enhances the readability and presentation of the specification. Moreover, web interfaces can be dynamically generated by transforming the SFML specification using general XSL technologies.

In the five teams that use SFG at the MITRE Corporation, processed files are typically no more than five megabytes, in size. The SFG easily produces these files in less than 10 minutes for up to five levels of chained queries. The users were extremely pleased with this performance. Most projects initially were accustomed to manually running queries and using several cumbersome scripts that relied on human intervention. Several projects have stated that they have become more efficient and *thorough* now that the preparation effort has been greatly decreased.

There are several limitations and many areas of future work discovered. One area is for future performance. Though the current performance of the SFG component is acceptable to the five teams that are currently supported, if this approach is used to build files 100 times the current sizes 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, there is the issue of extending the architecture for additional formatting features (*SpecialTransforms*). In the current approaches of XML/XSL, XSL is a more powerful formatting language. A major limitation of SFG is new formatting requirements require additional components to be added to assist the Line Element generation. On-going work on SFML is toward the extension of the schema to address newly discovered file format constraints. As these extensions are made, we are attempting to reuse formatting functionality currently available as opposed to writing new code. Another limitation is that the SFG is a "one-pass" process, while XML/XSL is "multi-pass". Since SFG queries and formats simultaneously, there is no opportunity to develop statistics over the entire file. A resulting limitation is deleting redundant records or giving a count of specific record types. Though these functions can be easily added, it would require additional software to be created. With multi-pass approaches, this type of enhancement is easier since formatting occurs once the entire file is created.

7. CONCLUSION

The SFG architecture has been undoubtedly useful in the area of distributed data management and dissemination, particular as shown in one research organization. In particular, the SFG approaches have greatly enhanced the initial CRS tools and techniques. One area of future work is the investigation of using SFG/SFML for data retrieval, dissemination, and formatting from multiple data sets as in the well-established area of federated databases. Such future work would necessitate the creation of a *data source* specification in the SFML specification and the supporting software in the SFG components. Another area of future work is toward enhancing the system into domains such as business process management and bio-informatics.

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., "A Specification-Driven Architecture to Support Distributed Database Dissemination in Custom Formats" (under review)

[3] 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, pp.13-34, April 2002, Kluwer Academic Publishers

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

[5] Cooper, B.F., Sample, N., Franklin, M.J., Olshansky, J., Shadmon, M., and Cohen, L., "Extensible Data Management in the Middle-Tier," In Proceedings of the 12th International Workshop on Research Issues in Data Engineering (RIDE'02), IEEE Computer Society Press, San Jose, California 2002

[6] Crystal Reports (2002) http://www.crystaldecisions.com/products/crystalreports/

[7] Enosys Software, http://www.enosyssoftware.com/

[8] Fernandez ,M., Morishima , A. , Suciu ,D., Tan ,W., "Publishing Relational Data in XML: The SilkRoute Approach", IEEE Data Engineering Bulletin , no. 24(2) , pp. 12--19 , 2001

[9] Fernandez, M., Simeon, J., Wadler, P., "A Semi-monad for Semi-structured Data" Proceedings of the International Conference on Database Theory, London, UK 2001

[10] Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., and Widom, J., "Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS Project" In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995

[11] 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

[12] IBM WebSphere Data Interchange, http://www-3.ibm.com/software/integration/appconn/wdi/

[13] JAVA SERVER PAGES (2003), http://java.sun.com/products/jsp/

[14] JAVA SERVLETS (2003), http://java.sun.com/products/servlet/

[15] Leslie, D.M.,"Transforming documentation from the XML doctypes used for the apache website to DITA" Annual ACM Conference on Systems Documentation /ACM Press, Sante Fe, NM 2001

[16] Oracle Corporation (2002), WebDB Application 3.0 http://oradoc.photo.net/ora816/webdb.816/a77075/basics.htm

[17] Petropoulos, M., Vassalos, V., Papakonstantinou, Y.,"XML Query Forms(XQForms): Declarative Specification of XML Query Interfaces", Proceedings of the 10th Conference on the WWW, Hong Kong, 2001

[18] WebInTool(2002), http://www.ri.bbsrc.ac.uk/webintool.html

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

[20] XML (2002), http://www.w3.org/XML/

[21] XML Light Weight Extractor (XLE) (2002) http://www.alphaworks.ibm.com/tech/xle

[22] XMLDBMS (2002), http://www.rpbourret.com/xmldbms/

[23] XSLT (2002), http://www.w3.org/TR/xslt