Information Reuse and System Integration in the Development of a Hurricane Simulation System*

Shu-Ching Chen[1], Shahid Hamid[2], Sneh Gulati[3], Guo Chen[1], Xin Huang[1], Lin Luo[1], Chengjun Zhan[1], Chengcui Zhang[1]

[1]Distributed Multimedia Information System Laboratory, School of Computer Science
Florida International University, Miami, FL 33199, U.S.A.
{chens, gchen01, xhuan001, lluo0001, czhan002, czhang02}@cs.fiu.edu

[2]Department of Finance, Florida International University,
Miami, FL 33199, U.S.A.
hamids@fiu.edu

[3]Department of Statistics, Florida International University,
Miami, FL 33199, U.S.A.
gulati@fiu.edu

Abstract - *This paper presents our effort in designing and implementing an advanced hurricane simulation system on the platform of the World Wide Web for the purpose of supporting decisional processes and hazard mitigation. The development of the system is based on amalgamated methodologies which come from diverse disciplines and is an integration of a variety of techniques. Superior to other similar systems in hurricane study, our system assembles and utilizes information and techniques in a more flexible and robust manner by taking advantage of system layering technique, component modeling strategy and database technologies. Preliminary experiments have been conducted to demonstrate the performance of the system.*

Keywords: Information reuse, component-based design, system integration, hurricane simulation.

1 Introduction

Hurricanes are one of the most devastating and costly natural perils in the U.S. Their accompanying hazards such as storm-driven surge and hurricane-forced wind can further aggravate the destructive power, and consequently incur tremendous property damage and loss of life [8]. For instance, the insurance industry was totally shaken to the tune of $15.5 billion losses caused by Hurricane Andrew [20]. In fact, hurricanes and tropical storms accounted for the major share of all property insurance losses in the past years.

Due to hurricanes' vast potential to incur property damage and loss of life, it is imperative that public, property insurance industries and related government agencies can be informed of the future occurrence of hurricanes and consequently are aware of hurricane related damages.

Quite a few research works have converged in this area and resulted in a number of models and tools to address this issue in both the commercial and public domains. The most notable commercial products include RISKLINK [12], AIR [2], ARA [3], CATALYST and USWIND [7]. Generally, most of those models utilize data from standard hurricane data resources, and have functional modules which can perform hurricane forecast, wind field modeling, damage assessment and financial loss estimation. HAZUS (Hazards, U.S.) [9], developed by Federal Emergency Manage Agency, is one of the most well known methodologies which exist in public domain. Originally, HAZUS was developed to reckon potential losses from earthquakes and gradually has been extended into a multi-hazard methodology with support for wind and flood hazards. HAZUS has been extensively used for the purpose of natural hazard loss estimation, mitigation and emergency preparation.

Although these models have been widely employed and produced promising results, they do have the following disadvantages:

1. **Poor data management and data reuse**. For data-intensive applications like hurricane risk and loss simulation, the following characteristics hold: a) a tremendous amount of historical and simulated data are required, b) the data needs to be stored and integrated in a structured and meaningful way, c) the data needs to be processed and accessed efficiently. However, most of the abovementioned systems do not apply real database management techniques. Their so-called databases are merely served by groups of data sets that are stored in the format of text files.
2. **Insufficient system portability and flexibility.** Most of the software is written in Fortran or C/C++. Although they are powerful languages suitable for

* **© 2003 IEEE**

engineering and scientific applications, Fortran and C++ provide no or only low-level source code portability due to their semantic looseness. Therefore, cross-platform migration of the applications is inconvenient or even impossible. A lot of effort is required to achieve the purpose and usually unpredicted errors are introduced.

3. **Limited accessibility.** Most of the products exist as stand-alone systems. The associated software has to be installed and run on every single machine. The access capacity is limited and it is inconvenient for information exchange and resource sharing.

Our system has been designed and implemented considering all of the aforementioned issues. In our system, advanced database management and modeling techniques are employed to facilitate the data reusability and manageability, and diverse system building languages are explored and assembled together to achieve system integration flexibility and efficiency, specifically Java is utilized as the major programming language for its promise of portable applications. To support universal and extensive accessibility, Internet platform is selected to serve as the running environment for our system.

The rest of the paper is organized as follows. Section 2 briefly introduces the envisioned system from the system integration perspective. The database design, specifically the database reuse is discussed in Section 3. In Section 4, the advantage of techniques and components reuse is illustrated through the design and implementation of a subsystem. System performance is analyzed in Section 5. Finally, Section 6 concludes this paper.

2 System Integration

The system is to be used for reckoning hurricane risk and loss simulation. Such a system is complicated because of the following issues that need to be addressed: 1) experts knowledge and techniques from various domains such as meteorology, civil engineering and insurance, need to be seamlessly integrated into the system; 2) statistical rationales and mathematical computing modules are indispensable for the target system, due to the requirement of enormous amount of data for simulation and the fact that there is not enough historical-referenced data available; and 3) efficient approaches and strategies are required to be incorporated into the system to store, model and manipulate the huge volumes of heterogeneous data.

2.1 System Architecture Selection

Almost all the applications must be equipped with functionalities that display the interactive user interface, perform the main application logic, and store/retrieve data. Although it is possible for all the tasks to be interwined together within a single module, such an approach is not preferable due to its high difficulties in maintenance and deployment. The architecture employed in our system is derived from the advanced three-tier architecture for its elegant support in building robust, sharable and extensible applications.

The three-tier architecture technology is one of the most widely used application development architecture. Its attraction lies in the idea of separating the main tasks into different layers, and allows the efficient construction of robust and extensible systems. The essential tiers of the three-tier architecture are the user interface tier, the application logic tier and the database tier. Their major functionalities are briefly summarized as follows:

User Interface Tier: The user interface tier comprises the entire user experience. This layer provides a graphical interface to the users for them to interact with the application, to perform task such as input data and view the results of the requests. It also handles the data manipulation and formatting once the client receives the resulting data from the server.

Application Logic Tier: The application logic tier realizes all the functionalities of the system. Application logic and control coordination are performed in this tier. Application logic can be shared among services with different user interfaces.

Database Tier: The database tier manages the storage of data, raw data processing and the access to the data.

2.2 System Environment Selection

From the service-providing point of view, the system needs to: 1) provide service to various groups of users with different domain knowledge. Some groups of users, such as meteorologists, have certain knowledge background. Some groups of users do not, for example, the residents who are interested in knowing hurricane projection, and the occasional web surfers who visit the application site by chance; 2) provide hurricane risk and loss estimation in a fast and convenient manner to the users. The simulated results should be distributed quickly and widely so that people can benefit from the system.

It is natural to prototype the target system in a web environment. Firstly, Internet is known for its tremendous capability in information transmitting and exchange, which suits our system requirement of fast distribution of the simulation results. Secondly, if designed and implemented appropriately, web applications are able to encapsulate complicated technical details and such an ability meets our intention to serve the general-purpose users as well as the expert users. Another advantage of Internet is that there is no need for additional hardware/software installation at the client side in most cases, which is also very valuable for our system.

2.3 System Deployment

The system is prototyped as a web-based system. Figure 1 depicts the overall architecture of our system. The users access the system via a conventional web browser such as Internet Explorer or Netscape Navigator.

System functionalities are provided using Java Beans. Oracle Database serves as the database tier.

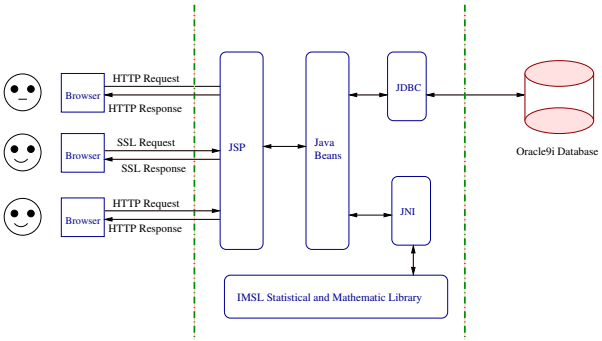

Figure 1. Illustration of the Overall System Architecture

2.3.1 Utilized Technologies

There are a variety of technologies utilized in the system. Detailed information is summarized in Table 1.

Table 1: Summary of Utilized Technologies

	Technologies
Front End	HTML
Server Side	JSP script
Middleware	Java Beans
RDBMS	Oracle 9i
Application Server	Oracle 9i AS
Mathematical Lib	IMSL C++ Version
Back End	JDBC, JNI

2.3.2 System Architecture Deployment

The system is designed and deployed with respect to the three-tier architecture.

User Interface Tier

The user interface (web page) is rendered automatically via Java Server Pages (JSP) scripting language [5][18]. JSP technique [5][18] can easily and flexibly generate dynamic content of web pages. The basic idea of JSP is to allow Java codes to be mixed together with static HTML or XML templates. The Java logic handles the dynamic content generation while the markup language manages the layout.

The user interface contains both the HTML components to collect the information and the HTML components to display the results. Components responsible for collecting the information, such as control buttons and combo boxes, can obtain user input and requests. The information-displaying components, for example text fields, display the results returned by the system. Information transmission between the clients and the server are governed via application-level protocols such as HTTP and SSL. Moreover, the Java Applet technique is exploited as well to make the web page vivid.

Application Logic Tier

The application logic tier bridges the gap between the user interface tier and the underlying database tier. An Oracle9i Application Server (Oracle9i AS) is deployed in the system that provides quality of services such as security and persistence services. Its OC4J container embeds a web server that responds to events such as data receiving, translating, dispatching and feed-backing [6][19]. Functional components in this tier receive the requests from the interface tier and interpret them into apropos actions controlled by the pre-defined work flow in accordance with certain pre-defined rules.

Java Beans perform the appropriate communication and calculation activities such as storing/retrieving information to/from the database, carrying out necessary computing work with respect to proper statistical and mathematical models. For the sake of performance, complex computation tasks are actually achieved by using C/C++ codes that are seamlessly integrated into the corresponding Java codes via the JNI (Java Native Interface) mechanism [4]. In addition, JDBC [11] is utilized to allow the access from Java Beans to the underlying physical database.

Database Tier

The database tier is responsible for modeling, storing hurricane information, and optimizing the data accesses. In our system, the object-relational concept is applied to model the spatial-temporal data for hurricane tracking. Hence, an oracle9i database is deployed in our system and acts as the database tier.

3 Database Information Reuse

Hurricane data used in this system are imported from the North Atlantic “best track” HURDAT [10]. Currently, this data set has been extended from 1851 to 2000. One problem with the original data representation of the storm tracks of the Atlantic basin is that they are manually recorded in the text files, and there is no unified format for the data entries. As a consequence, the contained hurricane track information cannot be efficiently analyzed nor fully utilized. In order to overcome this obstacle and to facilitate the information reuse, we processed the original data and populated it into the Oracle9i database. Several programs in a variety of programming languages (C++, Java, etc.) are developed to automate the processing and the populating tasks.

When selecting the suitable database management system (DBMS) for this system, we take into account the scalability of the database. For each storm, there is a collection of tracking records to keep its spatio-temporal trajectory and strength during its lifecycle. The landfall information also needs to be stored. Considering that the tracking data for one hurricane may contain more than one hundred records, the small-scale and mid-scale DBMSs, such as MySQL and Microsoft Access, are not suitable for

managing such a large-scale database. Instead, an Oracle9i database is deployed in our system due to its efficient data management for large-scale databases. An object-relational database schema is designed to facilitate the data reusability and manageability. The major advantage brought by the object-relational concepts is the ability to incorporate higher levels of abstraction into our data models; while current relational databases are usually highly normalized models but with little abstraction.

3.1 Object-Relational Schema for Hurricane Database

Figure 2 shows an overview of the basic structure of hurricane tracking data. As can be seen from this figure, a storm may last from 1 up to *N* days. On each day, there are 4 fix points (0 am, 6 am, 12 noon, 6 pm) recording the locations and strength of that specific storm at a specific time instance. In addition to the basic information, landfall information, such as the landfall states and the corresponding storm categories, are also recorded. In HURDAT, the maximum number of lasting days is 35, while the maximum number of landfall states is 4 for some storms. Figure 3 depicts the overview of the major part of the hurricane database schema.

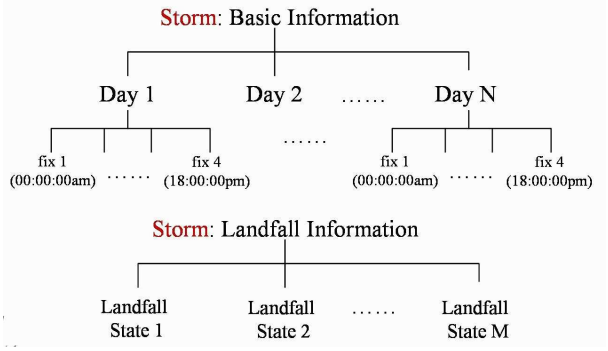

Figure 2. Hierarchy of Hurricane Tracking Data

For example, the object table, *Atmosevent_list*, is used to hold the high-level information for storms and hurricanes (atmosevent). Such information includes the storm_id, begin_date, category, storm_name, etc. As for individual fix points, the table, *Stormfix_list*, is used to store the fixes for all the atmosevents. A specific fix record is related to its corresponding storm via a foreign key *event_id* or via a reference field *for_event*, which refers to an *Atmosevent* object. By using the reference concept in ORDBMS (Object-relational database management system), it is straightforward to access a storm's abstract information while accessing only one of its fix records without the explicit JOIN operation. The table, *Landfall*, is used to store the landfall information for storms. Since a storm may make several landfalls, we embed a nested table, *Landfall_obj*, for each storm to hold the various numbers of landfalls that the storm may make. In this way, it is very easy to retrieve all the landfall information for a specific storm. Based on this object-relational schema, there are also some user defined procedures (UDP) in the ORDBMS. Applications can access the data directly or via UDPs. Sometimes, in order to achieve some level of domain-transparency, UDPs are more preferred. Intensive schema-driven optimization work has also been done on the SQL queries, which has greatly improved the system performance. Our system was extensively tested by the professionals at Florida International University, Florida Institute of Technology, and Hurricane Research Division at NOAA in Miami, and the results were satisfactory.

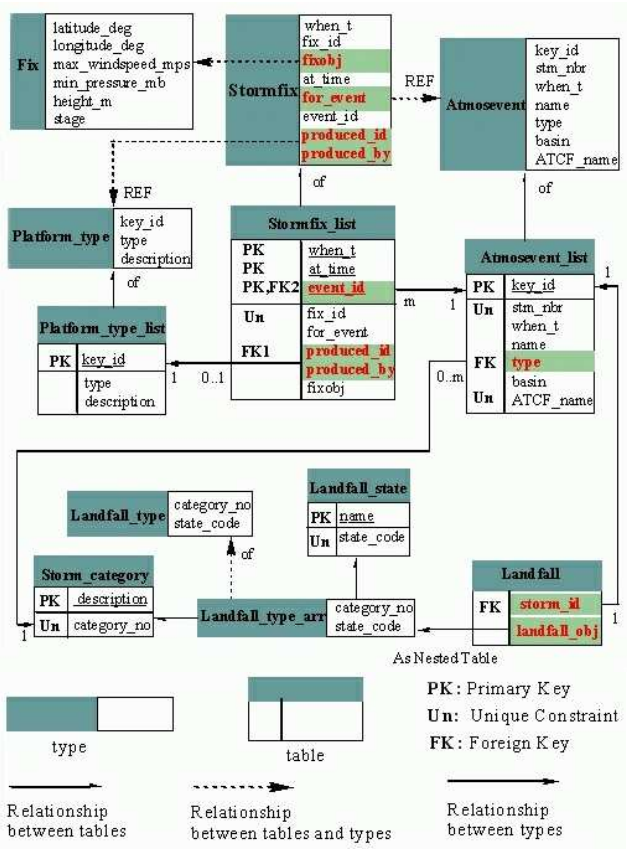

Figure 3. Database Schema

3.2 Database Query Performance Tuning

The performance of the database, especially the performance of database queries, is the main factor of consideration during the design of the entire system. The query language for our database is Structural Query Language (SQL). Retrieving, querying and updating the database are achieved by the embedded SQL statements. Although SQL is a relatively easy language, its non-procedural nature tends to obscure the performance-related issues. Since the databases used in this project are expected to be extremely large (including hurricane database, simulation database, insurance loss database, etc.) and are subject to a great deal of ad hoc query activities, the query performance of SQL statements becomes a critical issue.

Oracle SQL tuning is a phenomenally complex subject. In this paper, our experience and results in SQL

query tuning for two major subsystems, the Annual Hurricane Occurrence (*AHO*) sub-module and Storm Genesis Time (*SGT*) sub-module, are reported. In both of these two sub-modules, hurricane data are grouped into several pre-defined data sets such as year range (1851-2000), Multi-Decadal year, and ENSO year, etc. The major queries for *AHO* and *SGT* involve the join operations with more than two tables. We tune the original queries by using SQL hints and transforming all the subquery joins to equal-joins. A large number of experiments were performed on different data sets at different times across several days to evaluate the effectiveness of the tuned SQL statements. Table 2 shows some experiment results for the execution time for *AHO*. In this table, the testing cases are for the year range (1851-2000), Multi-Decadal year, and ENSO year. The dates for the experiments are October 15, 2002 (10/15/02), October 17, 2002 (10/17/02), October 28, 2002 (10/28/02), and November 4, 2002 (11/04/02). The time is measured in seconds. For example, the execution time for the 1851-2000 data set on 11/04/02 is 0.05 seconds.

Table 2: Execution Time of the *AHO* Queries

	10/15/02	10/17/02	10/28/02	11/04/02
1851-2000	0.03	0.04	0.03	0.05
Multi-Decadal	0.02	0.02	0.01	0.02
ENSO	0.01	0.01	0.01	0.02

According to our experiments, the execution time of the SQL queries for *AHO* has been dramatically reduced by a factor ranging from 4-30 after SQL tuning, and the execution time for *AHO* queries is relatively stable. However, the execution time for *SGT* queries is not very stable due to its complex query structure and the high network usage. In order to improve its efficiency and maintainability, we use the stored PL/SQL procedure to generate an intermediate table containing the first fix records for all the hurricane data in the database, then the *SGT* queries can be performed based on this intermediate table, avoiding the unnecessary table joins each time when a user query is issued. The consistency of this intermediate table with its related original tables are maintained automatically by a stored database trigger. Table 3 demonstrates our experimental results by using the above mentioned tuning skill. Based on our experiments, the performance of *SGT* queries after tuning have been improved by a factor greater than 30. Also, since it also reduces the tables accessed and the consumptions of computer resources, the performance becomes quite stable even during the peak usage of network.

Table 3: Execution Time of the *SGT* Queries

	10/15/02	10/17/02	10/28/02	11/04/02
1851-2000	1.02	1.00	1.04	1.00
Multi-Decadal	1.01	0.09	1.03	1.00
ENSO	0.08	0.06	0.07	0.06

4 System Design and Implementation Reuse

Taking advantage of reusability is a important aspect for effective and robust application construction. Reuse can be enhanced at different levels of the development process such as design schemas, content and even physical application pages [17]. In this section, we address the issues pertaining to design schemas and components reuse, and then exemplify the idea with concrete sub-module implementation.

4.1 Rationales

To better explore the reuse issue at all levels during the development process, it is important to investigate the system, identify its inherent modularity and relations, and select the suitable design and programming methodologies.

4.1.1 System Characteristics

The major tasks and functionalities of the system can be divided into four essential modules: (1) the Storm Forecast Module, (2) the Wind Field Module, (3) the Damage Estimation Module, and (4) the Loss Estimation Module. The four modules are related to each other in the sense that the prior module's output serves as the input for the following module; while they are also independent of each other in the sense that each module is self-contained and designed to perform important tasks of the system independently.

The process of handling each module is briefly described below [16]:

- **Login and Authentication**: An end user needs to login with username/password pair in order to access the system. The system has an authentication procedure to evaluate the login information.
- **Simulation and Computation**:
  - ❖ Retrieve data from database;
  - ❖ Execute required simulation and statistical computation procedures;
  - ❖ Restore necessary results back to the database.
- **Result Visualization**: Present the simulation results to the end user.

The proper separation of system functionalities along with the similar handling process in each module make it possible for us to reuse one module's design schema and implementation codes to the others to a great extent.

4.1.2 Methodology Selection

Two widespread methodologies have been investigated. The first one is the pure Object-Oriented methodology. Object Oriented Programming (OOP) method has been widely used in the past because of its ease of use, adaptability, maintainability, and reusability characteristics. However, OOP does not guarantee

reusability and is unsuitable for large-scale projects due to the complexity of relations among the objects [21].

The other one is the Component-based methodology [13][14]. Unlike OOP, the Component-based methodology is more suitable for large-scale projects since it enables the cost-efficient construction of large-scale software systems by reusing and integrating software components that already exist [15]. Well-defined software components can perform specific functionalities and usually are error-pruned. By reusing the well-developed components, Component-based methodology provides practical approaches to reduce the development expense and increase the productivity. System requirements are established and analyzed, and then the key software components are identified. Once the fundamental component interfaces and interactions have been specified, the actual application is created based on assembly and reuse of suitable components.

Considering the advantages of Component-based methodology over pure Object-Oriented methodology and our system requirements, Component-based methodology is a natural choice.

4.2 Case Study

Two sub-modules of the system, Annual Hurricane Occurrence (*AHO*) and Storm Genesis Time (*SGT*) are used as examples to illustrate information and technologies reuse during the design and implementation. Both sub-modules come from the Storm Forecast Module. After carefully investigating the requirements of these example sub-modules, we identify the fundamental software components. The development of later sub-module benefits from the design and implementation of the preceding sub-module.

4.2.1 Design and Implementation of AHO Sub-module

The goal of the *AHO* system is to estimate the number of hurricanes occurring per year based on an associated hurricane occurrence probability distribution, which is obtained through statistical analysis and calculation of the historical hurricane data sets. Hurricane data sets are induced by categorizing the available hurricane data records according to the climate cycles or qualifications. Different data set selections may significantly influence the estimation of the probability distribution, and hence greatly impact the final simulation results. Several statistical models are applied against the data set to identify the statistical distribution, which can best approximate the data set according to the domain knowledge in meteorology. The identified distribution is then used to perform the actual simulation and predict the future number of occurrences [16].

Five major components are identified for *AHO*, which are Authentication, Simulation, Statistical Model, Visualization and Database. The relationships among them is shown as Figure 4. Database component is the central part, both the Authentication and Simulation components need to communicate with the Database component to retrieve needed information. The Authentication component is used to verify user login information. The Simulation component controls the simulation process and interacts with the end users. It calls the Statistical Model component to perform the statistical functionalities and then uses the Visualization component to graphically display the results to the end users.

To implement the above software components, various underlying techniques and third-vendor packages are integrated into the system. For example, in the implementation of the Visualization component, the Ptolemy Applet package from Berkeley [1] is introduced because of its strong support for drawing diverse kinds of figures in the web environment.

The implementation of *AHO* sub-module has completed whereas the web pages are dynamically generated at runtime. Mathematic models are implemented in C++ program which is compiled as static libraries called by the Java Beans. Major Java classes and JSP files used for *AHO* are summarized in Table 4. Figure 5 and Figure 6 show the screen dumps of *AHO* at running time.

Table 4: Major Java Beans and JSP Files in *AHO*

Java Beans/JSP Files	Description
SystemLogin.jsp	For user authentication
Dataselection.jsp	Provide dataset information to proceed
Simulation.jsp	Simulation control and interface
Plot.jsp	Display graphical results
DataBean	Retrieve/store data from/to database
SimuBean	Simulation control
PlotBean	Result plot handling

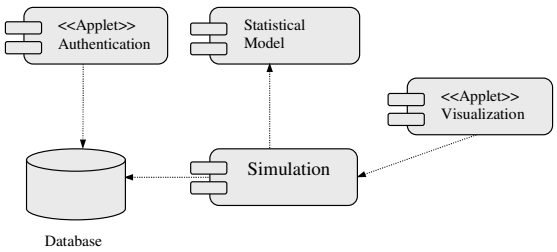

Figure 4. Component Diagram for *AHO*

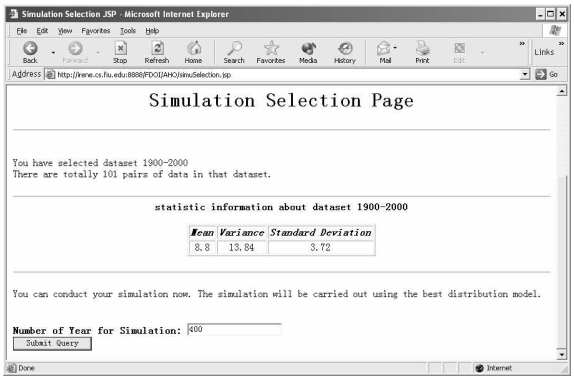

Figure 5. Snapshot of the Running Interface for *AHO:* Simulation Specification

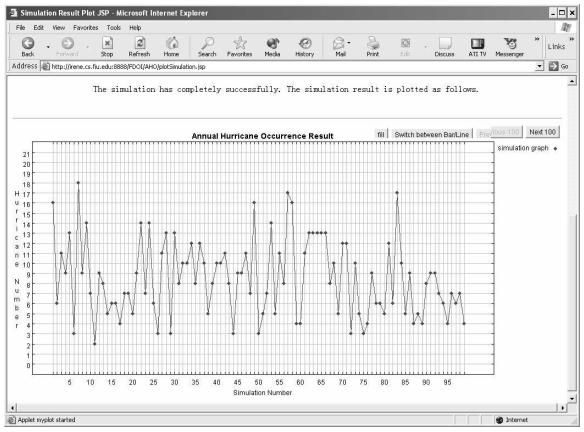

Figure 6. Snapshot of the Running Interface for *AHO:* Simulation Result

4.2.2 *System Design and Component Reuse of SGT From AHO*

SGT aims to model the genesis time of the hurricanes which is defined as the first fix data of the storm [16]. The basic idea is to analyze and model the occurrence pattern of the intervals of the genesis time of the hurricanes. Once being identified, that pattern is used to predict the time intervals of future storms, and then the storm genesis time of each storm can be predicted as well.

The two sub-modules *SGT* and *AHO* are similar except the actual statistical models are different. Hence, the major parts of the design and components of *AHO* could be reused. Basically, *SGT* contains similar components as those in *AHO*, where none or very few changes are needed for the Authentication, Simulation, Database and Visualization components. As shown in Table 5, major Java Beans and JSP files of *SGT* are very similar to those of *AHO*. As a matter of fact, the JSP pages for User Login are simply the same as in *AHO* while the Java Beans and JSP files for data retrieval and simulation need merely slight modification compared to that of *AHO*.

Benefited from the functionalities separation of three-tier architecture and the reuse of Component-based developing methodology, both cost and time for implementing *SGT* are reduced.

Table 5: Major Java Beans and JSP Files in *SGT*

Java Beans/JSP Files	Description
SystemLogin.jsp	For user authentication
SGTDataselection.jsp	Provide dataset information to proceed
SGTSimulation.jsp	Simulation control and interface
SGTDataBean	Retrieve/store data from/to database
SGTSimuBean	Simulation control

5 Performance Analysis

Experiments have been conducted to evaluate the system performance. Since the ability to handle high request rate and intense accesses is of most importance to Internet-based data-intensive systems, the target system is evaluated with respect to its response time and access capabilities under the circumstances of simultaneous multi-user access scenarios.

In our experiments, *AHO* and *SGT* are used as testbed. Test cases are designed to simulate 1 user access scenario, 10 user access scenario and 100 user access scenario respectively under normal traffic network condition using different input parameters, such as different historical hurricane datasets, different number of simulations, etc. For the multi-user access scenario, the results are averaged. For example, for the 100 user access scenario, the average system response time for the 100 users is used as the results. In our experiments, all test cases are repeated ten times.

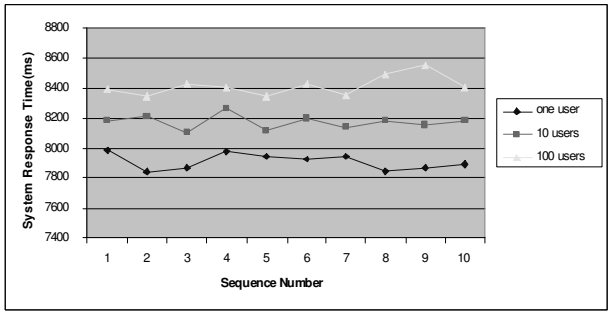

Figure 7. Performance Experiment Result of *AHO*

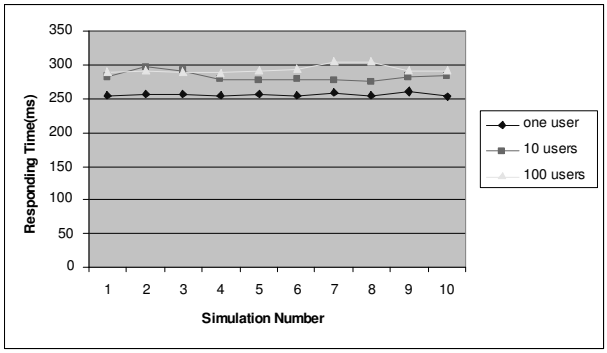

Figure 8. Performance Experiment Result of *SGT*

The results are presented graphically in a manner of average response time versus the sequence of simulations. The unit of response time is millisecond. We simulated 1,000 years' hurricane occurrence based on the historical hurricane records from the year 1851 to the year 2000 and the results are shown in Figure 7. The scenarios of 1 user, 10 users and 100 users are plotted. Although the figures of other combinations of datasets and number of simulations are omitted here, they do have almost the same characteristics as shown in Figure 7. Similarly, Figure 8 is taken as an example to depict the experimental results of *SGT*. The input dataset for this test case is the historical hurricane records from the year 1851 to the year 2000. We can observe from both figures that as the number of users increases exponentially, the system response time does not

increase dramatically. Actually, as shown in Figure 7, the average response time for the 100 users scenario is only less than 10 percent slower than that of the one user scenario. As to Figure 8, although the average response time for the 100 users scenario is about 20 percent slower than that of the one user scenario, it is still acceptable considering the exponential increase of simultaneous accesses to the system. One thing we need to mention is that because the underlying statistical model of *SGT* is less computation-intensive than that of *AHO*, the response time of *SGT* is much less than that of *AHO*.

From the above analysis, we can see that although it takes a slightly longer time for the system to answer the users' request as the number of users grows dramatically, the overall performance is still satisfactory regarding to the criteria of response time. The experimental results show that our system is a scalable high performance system.

6 Conclusions

In this paper, we have presented our ongoing experience in designing and developing a web-based system for real-time hurricane risk and loss projection with a real database support, which is built upon an assembled knowledge base from a variety of disciplines and utilizes on the collaboration of diverse functionalities.

To practically and efficiently integrate such a complicated system requires to pull together various advanced knowledge and techniques and reuse information and approaches as much as possible. We elaborated such issues during our developing endeavor from three aspects: (1) the aspect of system integration, (2) the aspect of data modeling and reusing, and (3) the aspect of information sharing and reusing. Experiments were conducted and the results have demonstrated the scalability and efficiency of our system.

7 Acknowledgement

This work is partially supported by Florida Department of Insurance under "FIU/IHRC Public Hurricane Risk and Loss Model Project." While the project is funded by the Florida Department of Insurance (DOI), the DOI is not responsible for this paper content.

References

[1] http://ptolemy.eecs.berkeley.edu/papers/99/HMAD/html/plotb.html.

[2] Applied Insurance Research, Inc. (AIR) page. http://www.air-boston.com_public/html/rmansoft.asp.

[3] Applied Research Associates, Inc. (ARA) page. http://www.ara.com/risk_and_reliability_analysis.htm.

[4] Java Native Interface. http://java.sun.com/docs/books/tutorial/native1.1/.

[5] Java Server Pages (TM) Technology. http://java.sun.com/products/jsp.

[6] Oracle9iAS Container for J2EE. http://technet.oracle.com/tech/java/oc4j/content.html

[7] EQECAT home page. http://www.eqecat.com/.

[8] FEMA hurricanes page. http://www.fema.gov/hazards/hurricanes.

[9] HAZUS manuals page. http://www.fema.gov/hazus/li_manuals.shtm.

[10] HURDAT. http://www.aoml.noaa.gov/hrd/hurdat/Data_Storm.html.

[11] The JDBC API universal data access for the enterprise. http://java.sun.com/products/jdbc/overview.html.

[12] RMS home page. http://www.rms.com.

[13] B. Boehm and C. Abts, "COTS Integration: Plug and Pray?", *IEEE Computer*, 32(1), pp. 135-138, Jan. 1999.

[14] P. Brereton and D. Budgen, "Component-Based Systems: A Classification of Issues", *IEEE Software*, 33(11), pp. 54-62, Nov. 2000.

[15] X. Cai, M. R. Lyu and K. Wong, "Component-based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes", Proc. 7th Asia-Pacific Software Engineering Conference (APSEC 2000), Singapore, pp. 372-379, Dec. 2000.

[16] S. Chen, S. Gulati, S. Hamid, X. Huang, L. Luo, N. Morisseau-Leory, M. Powell, C. Zhan and C. Zhang, "A Three-Tier System Architecture Design and Development for Hurricane Occurrence Simulation", Proc. IEEE International Conference on Information Technology: Research and Education (ITRE 2003), Newark, New Jersey, pp. 113-117, Aug. 2003.

[17] P. Fraternali, "Tools and Approaches for developing data-intensive web applications: A survey", *ACM Computing Survey*, 31(3), pp. 227-263, Sep. 1999.

[18] N. Morisseau-leroy, M. K. Solomon and J. Basu, *Oracle8i: Java Component Programming with EJB, CORBA, and JSP*, Oracle Press (McGraw-Hill/Osborne), 2000.

[19] D. Panda, "Oracle Container for J2EE(OC4J)", Jan. 2002, http://www.onjava.com/pub/a/onjava/2002/01/16/oracle.html.

[20] All Industry Research Advisory Council (AIRAC), "Catastrophic Losses: How the Insurance Industry Would Handle Two $7 Billion Hurricanes", Oak Brook, Illinois, AIRAC, 1986.

[21] F. T. Sheldon, K. Jerath, Y.-J. Kwon and Y.-W. Baik, "Case Study: Implementing a Web Based Auction System Using UML and Component-Based Programming", Proc. 26th International Computer Software and Applications Conference (COMPSAC 2002), Oxford, England, pp. 211-216, Aug. 2002.