13th World Conference on Earthquake Engineering
Vancouver, B.C., Canada
August 1-6, 2004
Paper No. 956

STRUCTURAL SENSOR DATA REPOSITORY: METADATA AND WEB-BASED USER INTERFACE

John-Michael WONG[1], Bozidar STOJADINOVIC[2]

SUMMARY

The structural sensor data repository provides the information technology framework necessary to assess the state of a structure. Although a multitude of structural data can be collected from sensors, there is no standard system for aggregating, storing, and fusing this diverse information for rapid civil engineering visualization. The structural sensor data repository attempts to fill this need by providing a viable metadata description for collecting relevant structural state data. It also integrates the data into a web-based graphical user interface. The web interface allows quick state presentation and access to individual sensor data.

INTRODUCTION

A civil engineering structure, a building or a bridge, undergoes many changes during its lifetime. Knowing the state of a structure is very important because we always strive to save lives in emergency situations and improve overall quality of life for people who use the structure. Recent advances in sensing and network technologies make it possible to collect a multitude of various data in and around the structure and to establish its state in a much more comprehensive manner than ever before by integrating this data. The goal of this structural sensor data repository is to conceptualize, develop and implement a prototype of a framework for gathering, classifying, and integrating structural state data collected in and around a structure, and to enable effective visualization and fusion of such data to define the state of a structure.

Framework Components

The structural sensor repository consists of multiple components working together as an application for storing and visualizing structural data. There are three major components in the structural sensor repository: the metadata schema, data access and storage systems, and the web-based user interface. The metadata schema defines the data structure requirements for storing structural sensor data. The data

[1] Graduate Student, Department of Civil and Environmental Engineering, University of California, Berkeley, USA, e-mail: jmwong@pobox.com
[2] Associate Professor, Department of Civil and Environmental Engineering, University of California, Berkeley, USA, e-mail: boza@ce.berkeley.edu

access and storage systems make it possible to search for and retrieve data from the database. The web-based user interface provides the facility to understand and visualize the structural data.

METADATA SCHEMA

The metadata framework provides the data structure for storing state data in a computer database. The framework contains five main classes of metadata. The first class describes actual collected or post-processed data such as accelerations, displacements, and inter-story drift. The second class describes the data collecting sensors and post-processing routines. Other classes describe locations on the structure, and events such as an earthquake or windstorm. The metadata schema contains the definitions for all of these classes.

Design Process

Development of the metadata schema began with defining different types of structural sensor data. The data sets differ by a few key parameters: source, scale, type, location, and time. The source of the data can be either *point* data from sensors monitoring a small, local region, or *field* data collected about a larger area or interpolated data from multiple sensors. The scale of the data can be *local*, focused on material-level properties, *intermediate*, focused on overall joints and members, or *global*, pertaining to the structure as a whole. Each category of data needs to be correctly described by its metadata.

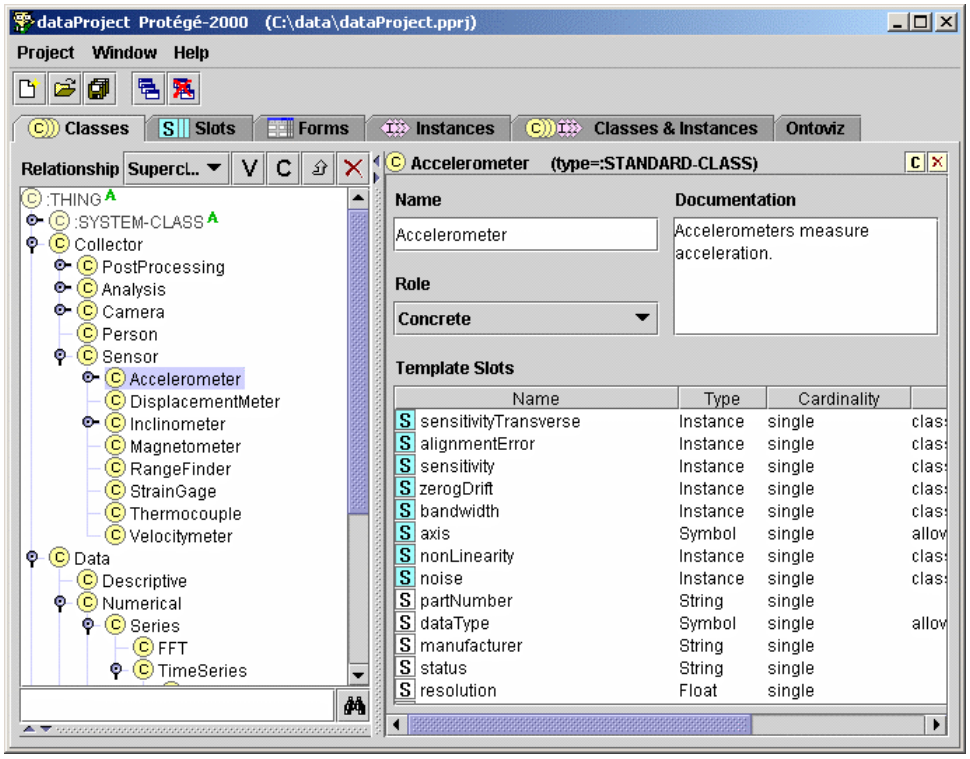

Figure 1: Protégé 2000 ontology development

Once general data types were established, more exact data structures were created to describe the precise data to be collected. An inheritance-based, object-orientated data structure was established to generate metadata and describe raw structural state data. The data structure was designed using the Protégé 2000 software tool (Figure 1). This program enables the user to construct knowledge domain ontology and enter domain knowledge. It allows for quick changes and modifications to object classes through a graphical user interface.

A preliminary version of the metadata schema was created drawing ideas from Futrelle [1] and Noy and McGuinness [2] as a basis for constructing the primary structural data model. Once the first version was created, several more refinements were made by investigating the information necessary to store data from a demonstration shaking table test at UC Berkeley. Attempting to store information from a simulation pointed out deficiencies in the early version and indicated areas for improvement. A revised data model contains a more complete set of classes and relationships for structural sensor monitoring.

Metadata Classes

The five primary metadata classes contain the definitions for object types and information to be stored in the repository (Table 1). Each object stored in the repository is an instance of one these types.

Table 1 Metadata classes and description

Class Name	Example Objects	Description
Collector	• Sensors (e.g., accelerometers) • Post-Processing routines • Analysis Routines • Human Observers • Cameras / Imaging	Objects that collect qualitative or quantitative data about the structural state.
Data	• Acceleration • Displacement • Interstory Drift • Photographs	Information collected about the structural state.
Event	• Earthquake • Wind storm	Durations of time when meaningful data was collected.
Location	• Ground Floor • Midspan Joist 3-A	Position inside the structure. Can refer to a specific position, or a general area.
Alert	• Maximum drift • Maximum acceleration	Limit states that indicate changes in structural state.

Collector

A Collector is a general object that collects or generates data about the structure. Each type of Collector is capable of obtaining different types of data. There are five main types of collectors: sensors, post-processing routines, analysis routines, human observers, and cameras. These are each subclasses of the main Collector class (Figure 2).

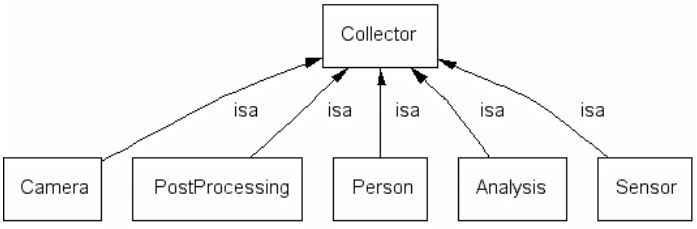

Figure 2: Collector class

Sensor objects describe actual structural sensors placed in and around the building (Figure 3). The schema defines slots for storing calibration data, sensor state data, and links to data the sensor collects. Sensors take input from physical changes in the building and output numerical data.

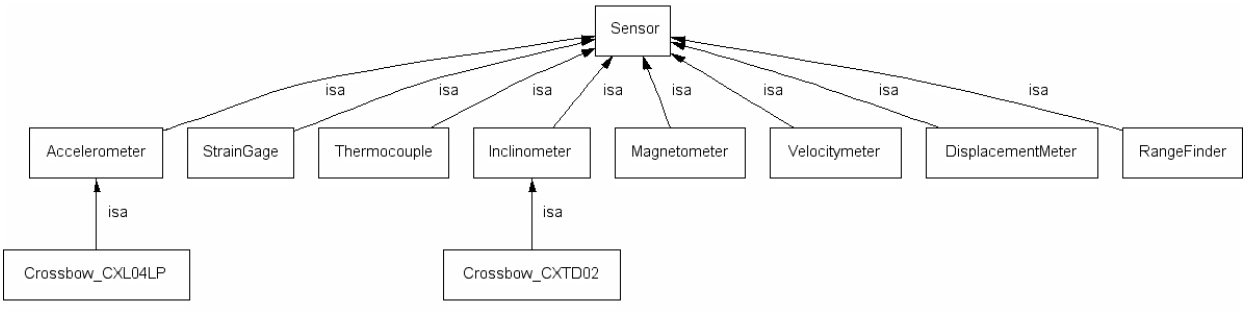

Figure 3: Sensor class

Post-processing routines are used to produce meaningful engineering data that are not directly read from sensors. For example, inter-story drift is not something that comes from a sensor. It requires calculating the difference between two displacement values. Post-processing routines take in numerical data as input and output new numerical data. For inter-story drift, a routine might take in two sets of accelerometer readings, perform a double integration to get displacement, and then find the difference between the two. The data sources and output are important to keep track of in the metadata schema.

Analysis routines produce information about the structure based on theoretical modeling of structural behavior. An example of this would be output from a finite element analysis program or a pushover curve. Analysis routines take in model information and output engineering response parameters.

Human observers and image collectors produce data that are qualitative. Some of the most important characteristics about structural state include things that are impossible to quantify explicitly. Engineering inspections for collecting material failure information and physical appearance is vital yet not simply numerical. Damage to non-structural elements, façade cracking, and photographs are other examples.

Data

Based on the characteristics of the Collector classes, there are three main classes of Data: numerical, descriptive, and visual (Figure 4). These are each subclasses of the main Data class.

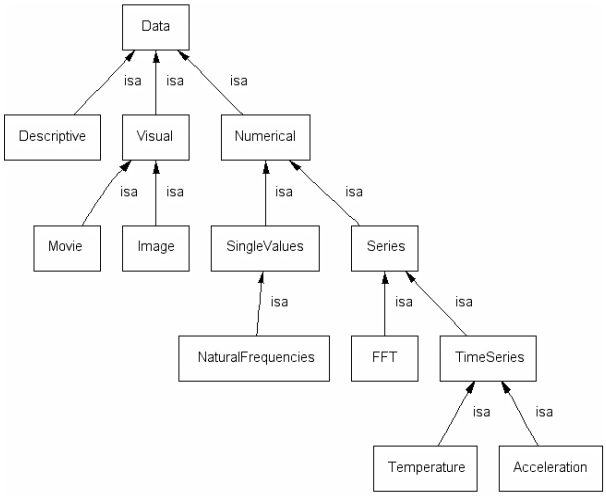

Figure 4: Data classes

Numerical data includes information such as accelerations, displacements, and natural frequencies. Within numerical data there are two divisions of information. The first is for time series data such as an acceleration response. This type of data is meaningless without the associated time parameter. However,

other information such as natural frequencies and peak displacements are not related to time and are meaningful on their own. This is the second class of single-value data.

Descriptive data includes written information that is generally non-numerical in nature. Examples include construction drawings for the structure, and a description of the non-structural damage inside the building after an earthquake. Visual data includes photographs, and movies of the structure.

Event
The data collected about the structure need to be categorized by more than just type. Since this system is designed for structural monitoring over a long period of time, there needs to be a facility for organizing the data into meaningful groups. It is not useful to collect data from a structure and record it all the time. Doing so would populate the database with millions of "zero" values when there is no activity. Rather, it is more meaningful to collect data only when something occurs, such as an earthquake, windstorm, or impact. The Event object provides for this functionality by defining a specific duration of time and coupling it with a description of the event.

Location
The Location class describes physical or logical positions within the structure. Although it is easy for a computer program to understand XYZ coordinates to describe a precise location, this is usually not meaningful to engineers. It is more useful to describe locations logically. For example, "midspan at joist 3-A" or "Node 5" is a lot more meaningful than "(0,100.34,120.0)". This class provides those linkages between physical coordinates and logical descriptors.

Structural state data differs in that some data are not point-specific, but rather express a quantity applicable for a larger location. There are three main scales of data to be considered. Point-specific data involve measurements such as material-level stress and strain. Intermediate-level data involves element deformations, joint rotations, and deflections. Global-level data involves overall drift and differential settlement effects. The Location class can encompass these different scales by creating different logical Locations that exist in the same physical location.

Alert
The Alert class exists to store information about threshold limit states and when they are exceeded. For example, maximum allowable interstory drift or an acceleration that will cause damage to non-structural components.

Integration
Fusing the data together involves managing links between the types of information. For this, a number of slots are designed to maintain the connections between the objects in the metadata repository. The Collector class contains slots "locatedAt" which ties it into its Location and a "producesData" slot which points to the data observed by the collector. The Data class contains links to the Collector that produced them and to the Event when the data was collected. The Alert class contains links to the Collector or Data set to monitor for limit state exceedance.

Metadata Examples

Collector:Sensor:Accelerometer
The Accelerometer object contains specific slots for the calibration information and standard manufacturer specifications. A Crossbow CXL04LP accelerometer is used as an example (Figure 5).

Each sensor has a unique identification name and slots for cataloging its serial number, manufacturer, and part number. The tolerance values for alignment error, bandwidth, noise, sensitivity, transverse sensitivity, and zero g drift are all in standard units. The detailed documentation for each slot tells the user which units to use. Different types of sensors have a separate set of technical specification slots. The sensor object also contains a slot for inputting calibration information on the sensor. This can point to an external file which contains the calibration details.

The sensor object also contains a slot *triggeredBy* for Events that triggered the beginning of data collection. It is only desirable to collect data when a meaningful event happens. In the case of an experiment, it would be the start of testing. Because it would make no sense to start collecting data before the test starts. For structural monitoring applications, data collection would be triggered by an earthquake, windstorm, or similar event. The *producesData* slot contains a list of the data sets produced by the sensor. The *status* slot is used for tracking whether the sensor is operating, turned off, or malfunctioning.

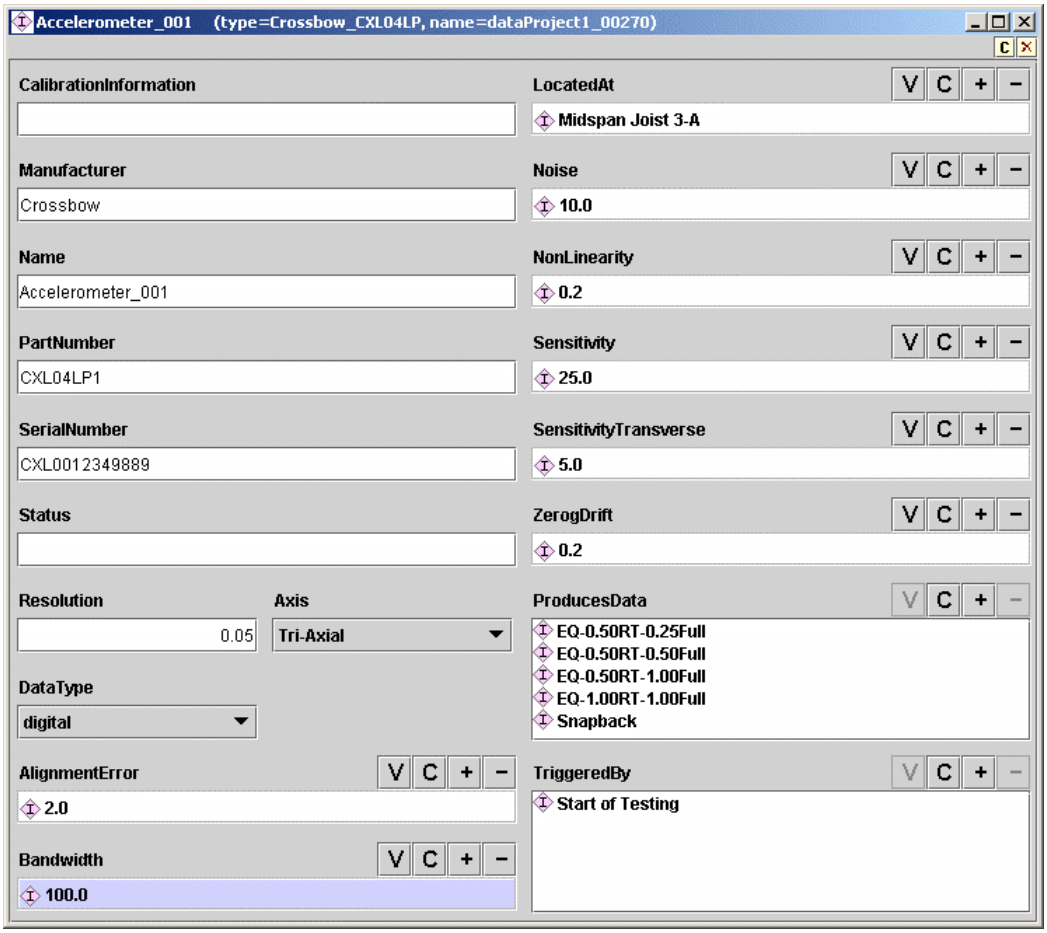

Figure 5: Accelerometer sensor example

Collector:Post-Processing:FFT
Sensors are not the only objects that can collect data. Individual sensors are equipped only to collect one specific type of data. Accelerometers measure only accelerations, LVDTs only measure displacements, and strain gages only measure strain. But, often the most useful engineering information is gathered from combining the readings from multiple sensors or from looking at the time history of sensor readings. This requires post-processing of the sensed data for producing new data sets.

It is important to separate the collected sensor data from the processed data. For example, it might be useful to obtain displacements from acceleration readings by numerical integration. However, there are several methods for performing it (e.g., midpoint, trapezoids, Simpson's Rule). It is important to be able to go back to the source data and be able to try different computational methods. This example shows an object for FFT post-processing of data (Figure 6).

The post-processing objects each contain a slot *name* for the unique name identifying the analysis routine, a *usesData* slot for listing input data sets used for processing, and a *producesData* slot for listing output data sets. This allows the repository user to search for all of the data processed by this analysis routine, determine which data sources were used for calculating the post-processed data, and to find which post-processing routine uses a specified data set. Because the input data contains links to its Collector, the repository can search recursively to determine which sensors were responsible for collecting the input data.

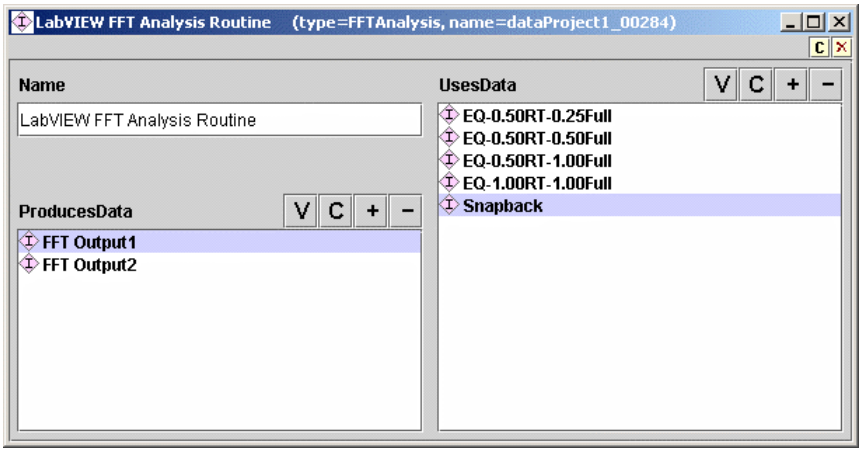

Figure 6: FFT Post-Processing Example

Data:SeriesData:Acceleration
Data objects can contain the data values in two different ways. It can be stored in a data file specified in the *dataFile* slot, or the values can be placed directly in the metadata object. If the data has not finished collecting, then it makes more sense to use a file link so that when the file is updated, the data can be read again. It is also best for extremely large data sets. For smaller data sets and instances where it is desirable to search for data by its values, placing it directly in the metadata object is preferred.

In this example (Figure 7), the data exists in both ways – inside the metadata entry and also linked to an external data file which contains the values. The *timeStamp* slot indicates the start of the data collection. ISO 8601 time stamps are used when possible for the data values. But, the times can also start at 0 and the *timeStamp* slot value can be used as an offset to calculate the actual time of the data values.

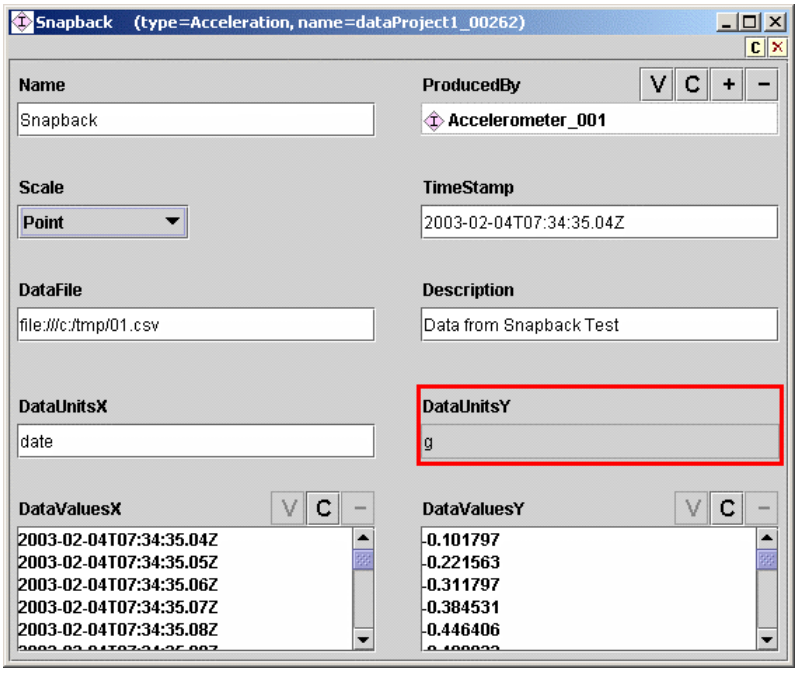

Figure 7: Acceleration data example

Location:PhysicalLocation

Physical locations are described by X-Y-Z coordinates relative to a defined origin, or other starting location. In this example (Figure 8), the joist at grid line 3-A is specified by an absolute coordinate based relative to the Origin. Making locations relative is useful because it allows the engineer to describe locations more intuitively. For example, if a sensor is placed next to beam-column connection on the roof, it would make more sense to describe its location relative to the beam-column centerline intersection rather than to an absolute coordinate system starting on the ground floor. Logical location objects can also be used when absolute position is not required, such as for correlating nodes in a structural model to locations in the structure. Sensors can have more than one location.

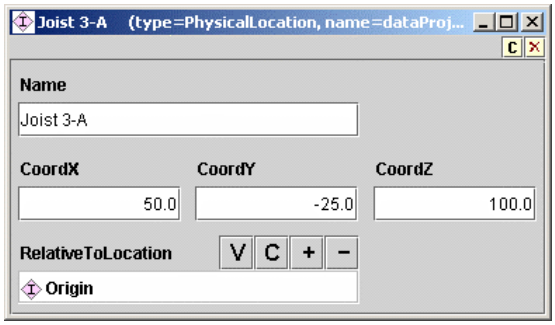

Figure 8: Physical location example

Implementation

The structural state data schema was designed in Protégé 2000 and then implemented by writing it in NEESML [3]. NEESML is the standard XML format for defining metadata schemas and communicating with the NEESgrid repository. NEESML is designed to be easy to understand and much simpler to write than normal XML. Using NEESML for implementing the data schema allows using the NEESgrid services for data access and storage.

DATA ACCESS AND STORAGE

The data access and storage routines provide the functions for reading and writing metadata and sensor data. Data is only useful if it can be efficiently found, stored, and retrieved. The system used for the repository involves three separate layers for accessing the data: the storage layer, access layer, and transaction layer (Figure 9). The storage layer is responsible for maintaining the actual files. This is usually handled by a network file server or a local hard drive. The access layer controls permissions to read and write to the data file in the repository. The transaction layer provides the interface for applications, such as the web interface, to access the data.

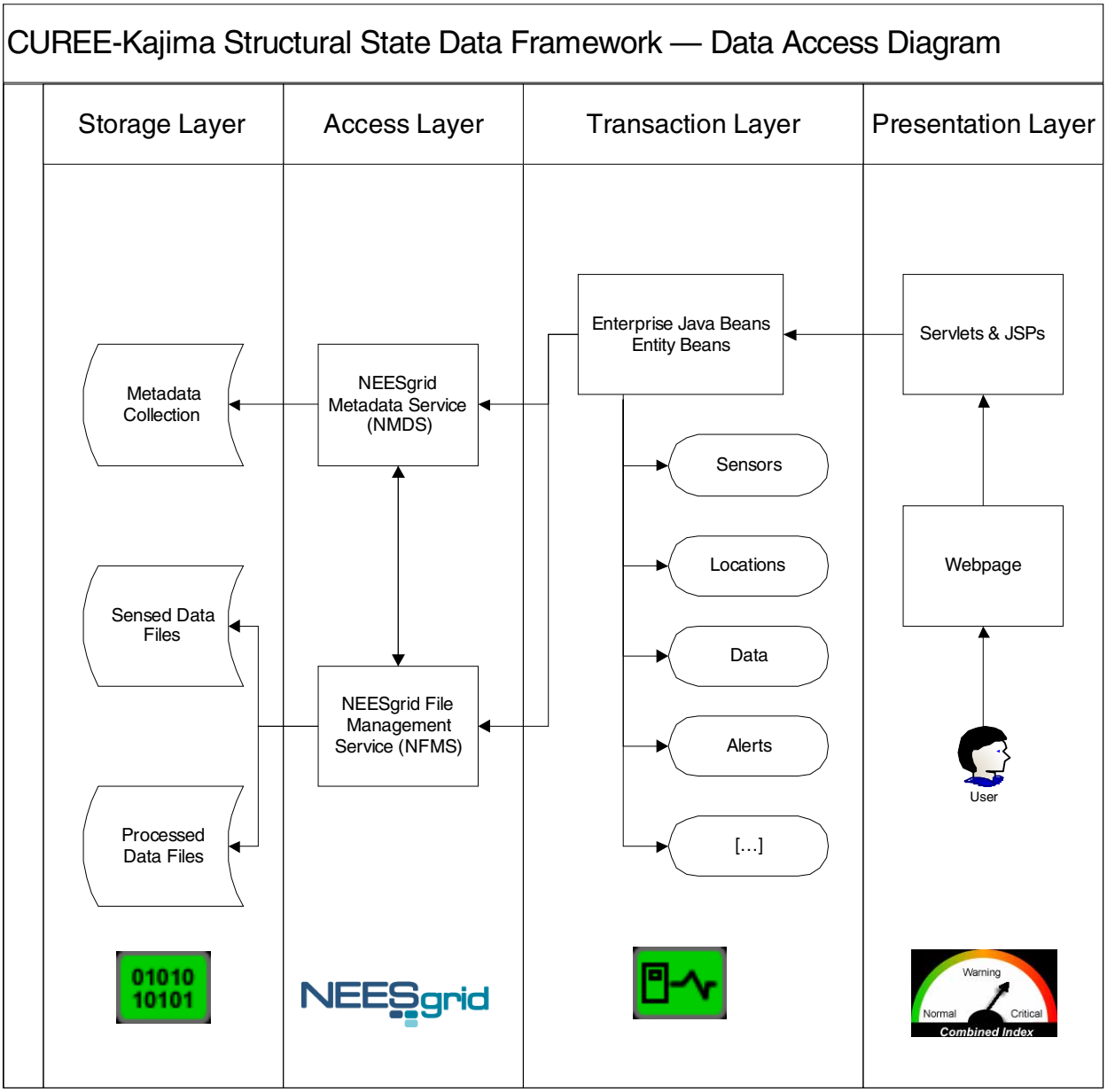

Figure 9: Data access diagram

Metadata Transactions

Enterprise Java Beans (http://java.sun.com/products/ejb) were chosen as a programming interface to access the data and metadata stored in the system. Using EJBs creates a standard interface for managing application-level data access and for implementing the web user interface. EJBs were chosen for the system because they are system portable, not requiring a specific operating system or hardware to run. It separates the database transactions from the user interface application.

The structural data repository also fits all three suggestions from Armstrong [4] for when to use EJBs. The repository must be scalable to accommodate a potentially large number of users and data sets. The data inside the repository must also be protected to ensure data integrity and concurrency. And, the data stored in the repository should be available for other applications in the future other than the web-based interface. Enterprise Java Beans provide the technical solution for the project requirements.

Each major object type in the data schema has a corresponding Bean written for it. Each Bean contains the logic for storing and retrieving its data from the repository. It also has class functions for performing search queries and returning data. For example, the Bean class written for an acceleration Data object can return a reference to the sensor that produced it, a plot showing the data, and the maximum acceleration felt by the sensor.

The structural state data needs to be navigable by different methods to allow the user to find information quickly and intuitively. There are three main search keys that would be used by an engineer to retrieve different information (Table 2). These requirements helped form the requirements for the web interface.

Table 2 Search queries and expected information

Search By	Examples of Information to Retrieve
Sensor	• List of all sensors in the structure • Attributes of the sensor (status, calibration information, etc.) • Data collected by the sensor • Events monitored by the sensor • Locations of the sensor • Alerts triggered by the sensor
Events / Time Interval	• Sensors active in collecting data during that interval • Time interval for an event / Events in a time interval • Data from all sensors during an event • Alerts triggered during the event
Location	• Sensors at the location • Alerts focused on the location

Data Storage

Sensor data and post-processed data are stored in plain text files. These data files reside on the main application and web server for the repository. They are organized by collector and location. Each data file is linked to a metadata repository entry which contains a URL pointing to the location where the data file can be accessed. The metadata entry also includes information about the data file contents such as: data source, file format, and measurement units. Using metadata to define information in the data file makes it possible to store new data types in the future and include non-quantitative data such as images or documents using the exact same system.

Implementation

Metadata access and storage is handled by using the NEESgrid metadata service (NMDS). NEESgrid metadata objects provide all of the features that are necessary for implementing the structural state data repository. Its objects are based on an RDF-like model and allow linking references to other metadata objects as well as primitive types like strings and integers [5]. Using the NMDS API obviates the need to program database calls for storing objects in tables. The API allows for manipulating the metadata on the object level. Each of the Java Beans uses the API to retrieve and store the metadata.

WEB INTERFACE

A visual access paradigm is the best way for engineers to access the data and for clients to visualize the state of their structures. The clients need a visual tool to understand the state of the structure after an earthquake, and to monitor their structure over time. An engineer user requires a more detailed look into the state of the structure, such as access to individual sensor readings and the ability to extract the data into numerical tools for structural modeling and assessment. A web-based interface was a natural choice for delivering information given the widespread familiarity with web browsers and ability to present different types of information. A demonstration of the web interface is available on the project website (http://www.ce.berkeley.edu/~boza/research/Curee-Kajima/).

Dashboard Analogy

The primary user interface was inspired by the instrument cluster on a car – instantly readable and understandable by any user (Figure 10). Just as a driver knows to pull over and stop when the check engine warning light comes on, the user interface can also alert the building owner whether to call the engineer for an inspection after a moderate earthquake with a check building light.

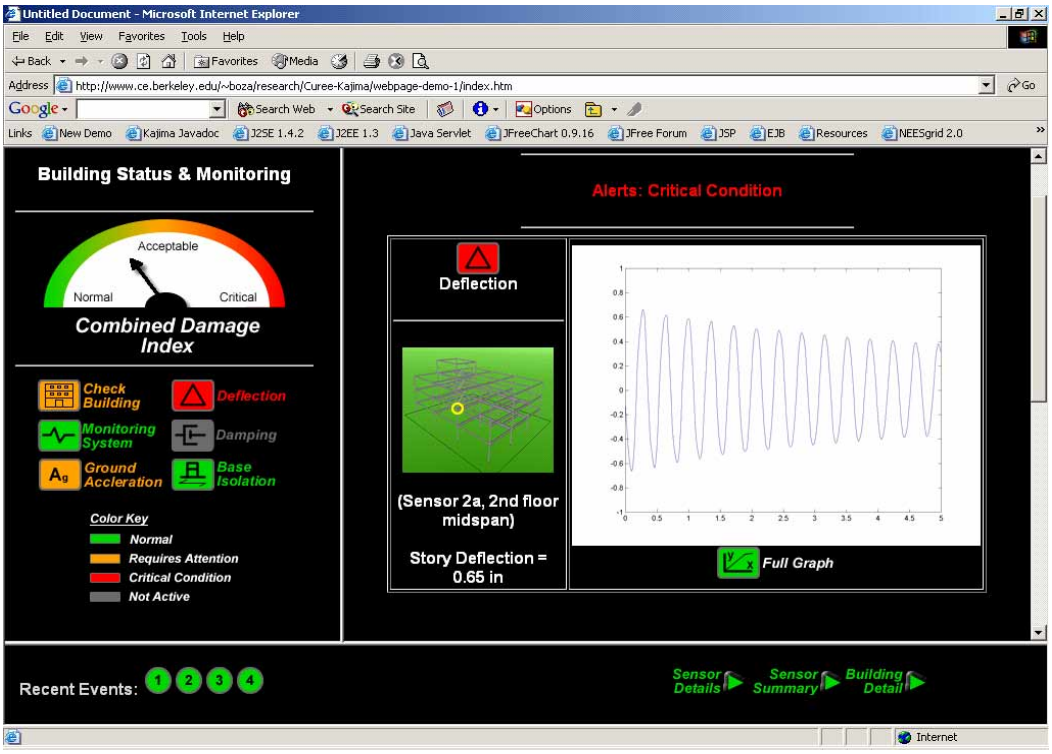

Figure 10: Web-based interface

On the right hand side of the website, there is a panel with color-coded icons showing the status of the items being monitored for performance. A green color indicates normal function, yellow indicates a warning, and red indicates a critical condition. The limits are established by the structural engineer and are stored in an Alert object in the repository. Above the indicator "lights" there is a dial which represents a weighted average status of all the alerts.

An opening page shows the name of the building, a street map showing the orientation of the building plan and a photograph of the building. Other pages link to detailed sensor information and sensor summary information. The sensor summary page gives a quick listing of the major structural sensors, their status and an indicator of their recent readings. The sensor detail page retrieves the calibration information and specifications for the sensor from the repository and displays it along with the sensor's status and a plot of its recent readings. From here, the user can then download a text file containing the sensor data and other information. A separate building detail page provides links to other static information about the structure. The data repository can also store information such as architectural and structural drawings and other documentation. This makes is an effective single place to find all the information an engineer needs to evaluate the structure.

Visualization Methods

Numerical data is visualized in the web interface with 2D data plots. Clicking on a sensor detail page will display information about the location where the sensor is located, the engineering characteristics of the sensor, as well as links to graphs of data collected by the sensor. Varieties of different graphs are supported including time-series graphs, X-Y graphs, bar charts, and scatter plots. Acceleration and displacement are usually best visualized using time-series graphs. Structure periods changing over time are better visualized using scatter plots.

Drill-down graphs are also used to allow users to click on points in a graph to retrieve more detailed information about that data point. The ability to have this kind of interaction with a graph makes it possible for advanced navigation and more natural data presentation.

Implementation

The web-based user interface is written using a combination of Java Server Pages and Servlets. This is a natural choice because the metadata access methods are also written in Java using Enterprise Java Beans. The JSPs and Servlets written for the interface contain the necessary calls to the beans to retrieve the information from the repository. Both JSPs and Servlets behave the exact same way from the users' perspective, but each has distinct advantages and disadvantages from a programming perspective. Saimi [6] notes that using a combination of both makes it more efficient to design a web application service.

The plots used in the interface are generated using the JFreeChart library (http://www.jfree.org/). JFreeChart was chosen because of its ability to generate all of the different types of plots that the data require. It is also capable of running in a servlet environment allowing it to plug in directly to the rest of the data and visualization framework. It is also easily extendible and customizable to suit structural visualization needs.

RESULTS AND DISCUSSION

Metadata Schema

The metadata schema generated for the structural data repository was evaluated based on its effectiveness at representing data from an actual building in service. Collaborators from the Kajima Corporation provided sample monitoring data and discussed the ways the system is envisioned to be used by practicing structural engineers.

The data were examined and the information was processed into a set of data files and metadata descriptions were written for inclusion in the repository. Although the metadata framework was robust enough to store the data, there were a few problems. Manually editing an XML file containing the metadata for the data set was time consuming and was error prone. A better tool needs to be built for entering in the structural data. This can be handled by building additional modules for the web interface.

Experimental Testing

Utility of the structural data repository will be evaluated using a small-scale shaking table test of an instrumented model frame. The instrumented model will serve as a test structure to determine whether the system is robust enough to handle storage, access, and visualization of structural sensor data. Modifications will be made based on the results of the test.

Jan Goethals at University of California, Berkeley has developed a simplified data acquisition system using TinyOS with Crossbow wireless motes. This system will be linked with the structural data repository for testing (Figure 11).

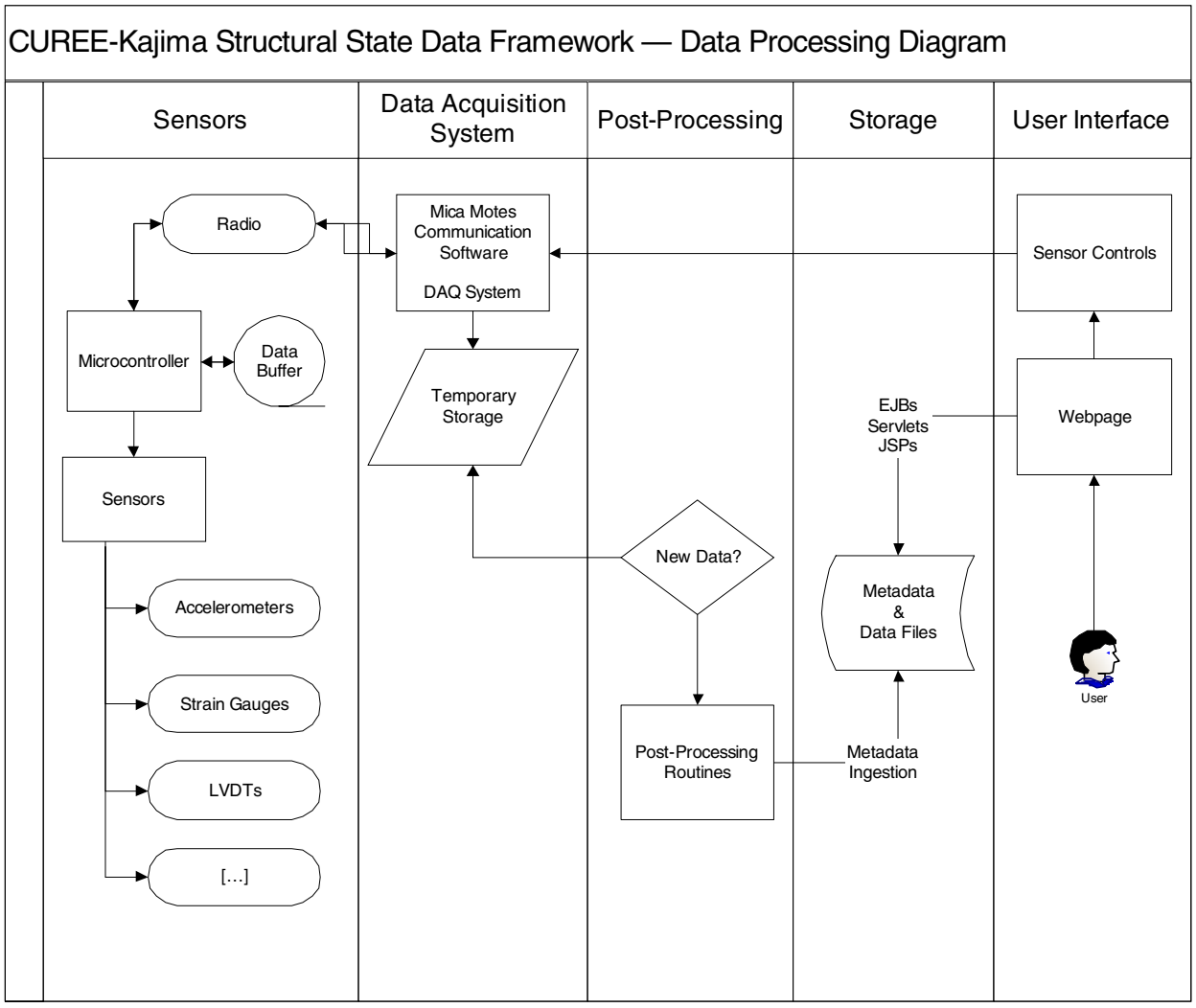

Figure 11: Data processing system

Modular System

The repository is designed using a modular framework making it easily extendible in the future. Because the data storage and access components are separated from the interface programming, changes to the database backend or changing needs for engineering visualization require only modifying one component. This makes it possible to tailor the system to the visualization needs of a specific structure while using the same structural data framework.

Using the system with a different storage and retrieval mechanism will only require modification to the Java Beans. The storage and access is separated from the interface programming. The beans serve as the connector between what the user sees and interacts with in the interface and the data structures that are behind the whole system. This allows the use of other data storage systems and to provide a mechanism for linking the visualization system with existing monitoring systems and existing data in different formats.

Further improvements to the visualization system are possible simply by programming additional servlets of JSPs. This allows for developing site-specific visualization in tandem with the existing general visualization components. Examples of this could include the monitoring of a base isolation system or an active mass damper.

CONCLUSIONS

This new framework for a structural sensor repository provides a metadata structure capable of storing data collected by structural sensors. It also provides the basic facilities for post-processing collected data for structural state determination based on engineering limit states. A dashboard-style, web-based user interface was created for users to interact with and visualize the data stored in the repository. Access to the data is managed through a neutral, application-centered API specification using EJBs that allows for future extension to the web interface and for new data acquisition methods.

The particular implementation of the structure state data repository presented in this paper is just one of many possible implementations, especially regarding the design of the web interfaces. A variety of different interfaces can be designed to suite a particular purpose. Experiments using a small wireless sensor network piggybacked on ongoing shaking table tests at UC Berkeley laboratories will be a practical challenges the authors will use to improve the existing implementation of the repository. However, the underlying design and the relations among the components of the structural sensor data repository presented in this paper are more important. They present the first attempt to design a general framework for structural sensor data collection. Such frameworks are needed to handle the data generated by a large number of sensors expected to be deployed on civil infrastructure in the very near future.

ACKNOWLEDGEMENTS

This project is part of Phase V of the CUREE-Kajima Joint Research program. The CUREE-Kajima Research program is a cooperative program between the Kajima Corporation and the Consortium of Universities for Research in Earthquake Engineering for developing collaborative studies with researchers in the U.S. and Japan. Kajima funding, and particularly the feedback from Mr. Satoshi Ohrui and Mr. Yoshifumi Yamamoto, Kajima engineers on the Japanese side of this project, is gratefully acknowledged.

REFERENCES

1. Futrelle, J. "NEESgrid Strawman Metadata Model v1.1." http://www.neesgrid.org/repository/metadata/1.1/index.html. National Center for Supercomputing Applications, 2002.
2. Noy NF, McGuiness DL, "Ontology Development 101: A Guide to Creating Your First Ontology." Knowledge Systems Laboratory, Paper no. KSL-01-05. Stanford University, Mar 2001.
3. Futrelle, J. "NEESML Reference User Guide." Technical Report NEESGRID-2003-15. National Center for Supercomputing Applications, Aug 2003.
4. Armstrong E, et al. "The J2EE 1.4 Tutorial." Sun Microsystems, Nov 2003.
5. Futrelle, J. "NMDS Reference User Guide." Technical Report NEESGRID-2003-21. National Center for Supercomputing Applications, Oct 2003.
6. Saimi A, Syomura T, Suganuma H, Ishida I. "Presentation layer framework of Web application systems with server-side Java technology." Computer Software and Applications Conference, Taipei, Taiwan. IEEE, 2000: 473-478.