

AN INFORMATION MODELING FRAMEWORK FOR BRIDGE MONITORING

Seongwoon Jeong¹, Rui Hou², Jerome P. Lynch², Hoon Sohn³, Kincho H. Law¹

¹Department of Civil and Environmental Engineering, Stanford University,
473 Via Ortega, Stanford, CA 94305-4020, USA

²Department of Civil and Environmental Engineering, University of Michigan,
2350 Hayward St., Ann Arbor, MI 48109-2125, USA

³Department of Civil and Environmental Engineering, Korea Advanced Institute of Science and
Technology, 291 Daehak-ro, Yuseong-gu, Daejeon 305-701, Republic of Korea

ABSTRACT

Bridge management involves a variety of information from different data sources, including geometric model, analysis model, bridge management system (BMS) and structural health monitoring (SHM) system. Current practice of bridge management typically handles these diverse types of data using isolated systems and operates with limited use of the data. Sharing and integration of such information would facilitate meaningful use of the information and improve bridge management, as well as enhance bridge operation and maintenance and public safety. In many industries, information models and interoperability standards have been developed and employed to facilitate information sharing and collaboration. Given the success of building information modeling (BIM) in the Architecture, Engineering and Construction (AEC) industry, efforts have been initiated to develop frameworks and standards for bridge information modeling (BrIM). Current developments of BrIM focus primarily on the physical descriptions of bridge structures, such as geometry and material properties. This paper presents an information modeling framework for supporting bridge monitoring applications. The framework augments and extends the prior work on the OpenBrIM standards to further capture the information relevant to engineering analysis and sensor network. Implementation of the framework employs an open-source NoSQL database system for scalability, flexibility and performance. The framework is demonstrated using bridge information and sensor data collected from the Telegraph Road Bridge located in Monroe, Michigan. The results show that the bridge information modeling framework can potentially facilitate the integration of information involved in bridge monitoring applications, and effectively support and provide services to retrieve and utilize the information.

Keywords: Bridge information modeling, bridge management, bridge monitoring, NoSQL database

1. INTRODUCTION

Bridge management involves copious and diverse information, including geometry, engineering model, inspection reports and monitoring data. Current practice of bridge management employs isolated systems to manage and process different types of information wherein information managed in one system is neither shared among other systems nor integrated with information managed by other systems. However, as bridge monitoring and management technologies advance and the number of bridge monitoring and management applications increases, the demand for efficient information sharing and data exchange will grow. Sharing and integration of such information would enable meaningful uses of information and improve bridge management services, as well as enhance bridge operation, maintenance and public safety.

Much research has been conducted in developing data exchange and interoperability standards in many industry domains to avoid error-prone and time-consuming manual data conversion as well as to facilitate automated exchange of information and machine-to-machine interaction (Nagel *et al.* 1980, ISO 1994). In the architecture, engineering and construction (AEC) industry, building information modeling (BIM) has been widely adopted as a means to support integrated project delivery process and data exchange

throughout the project lifecycle (Eastman *et al.* 2011). One of the *de facto* BIM standard data models is the Industry Foundation Classes (IFC) (buildingSMART 2016). The IFC standard specifies platform-neutral file format using EXPRESS modeling language to enable digital data exchange among building design and analysis systems. The IFC-EXPRESS schema has been translated into XML (eXtensible Markup Language) format, a commonly used representation of industry standards. (buildingSMART 2016).

Given the success of BIM, research efforts have been initiated to develop frameworks and standards for bridge information modeling (BrIM). The main objectives of BrIM research are twofold: enabling an integrated bridge data repository and developing electronic data exchange standards for bridge applications (Chen and Shirolé 2006). Focusing on the spatial and physical entities of bridge structures, IFC-Bridge for bridge modeling has been proposed (Yabuki *et al.* 2006). There have also been research efforts towards developing information modeling framework for bridge management. Marzouk and Hisham (2011) proposed a BrIM framework to support bridge management by connecting a 3-dimensional bridge model, inspection sheets and structural condition assessment modules. Samec *et al.* (2014) developed a BrIM system, along with a 3-dimensional visualization tool and a mobile application, to manage bridge life cycle information that includes inspection and maintenance data. To facilitate information interoperability in the bridge domain, one of the most notable efforts for developing software-neutral BrIM data schema is the OpenBrIM standards (Chen and Shirolé 2013). Supported by the US Federal Highway Administration (FHWA), OpenBrIM is “a bridge industry consensus standard for engineering data description, modeling, and interoperability for integrated structural design, construction, and lifecycle management of bridges” (see <https://collaboration.fhwa.dot.gov/dot/fhwa/ascbt/brim/>). OpenBrIM uses XML as the basic syntax to define object-based information model (ParamML 2016). OpenBrIM (version 1.0 and version 2.0) was originally developed by the research team at the State University of New York at Buffalo (Chen 2013). The current version (version 3.0) of OpenBrIM is led by CH2M Hill and sponsored by FHWA (Bartholomew 2015). The efforts so far have been focused mostly on the 3-dimensional representation of bridge structures. As such, current OpenBrIM standards lack the data entities for representing the information pertinent to bridge monitoring applications. In our framework, we use the OpenBrIM as the base schema and extend it to include data entities for capturing bridge monitoring information. Specifically, we extend the standard OpenBrIM schema to include engineering entities for analysis modeling and sensor information.

Relational database (RDB) systems are often employed as the primary data storage for bridge modeling, bridge management and bridge monitoring applications (Lee and Jeong. 2006, Marzouk and Hisham 2011, Robert *et al.* 2003, Wang *et al.* 2009, Li *et al.* 2006, Zhou *et al.* 2006). However, the tabular structure of RDB is not convenient for handling semi-structured data (e.g., XML document and tagged-text) and unstructured data (e.g., text and image), which are commonly found in engineering applications. For bridge monitoring, the collected sensor data need to be managed and processed efficiently. Because of the variable lengths in the data records, the data sets are not conveniently structured in a RDB system. Research has shown that RDB systems have fundamental drawbacks in satisfying the performance and scalability requirements for the new era of “big data” with a variety of formats and a large volume of data (Stonebraker *et al.* 2007). To overcome the shortcomings of RDB systems for handling big data, NoSQL (Not-Only-SQL) database systems have been widely adopted in applications such as real-time analysis, knowledge representation and large-scale data management (Hecht and Jablonski 2011). Recent studies have shown that, in comparing to RDB systems, NoSQL systems enable higher scalability, better flexibility and faster performance by reducing some of the consistency requirements and supporting more flexible data schemas (Li and Manoharan 2013, Grolinger *et al.* 2013). Because of their flexibility, scalability and performance, NoSQL database systems can be an effective alternative for handling bridge monitoring and management data and supporting bridge monitoring applications (Jeong *et al.* 2016).

In this paper, we present a bridge information modeling (BrIM) framework for bridge monitoring applications. The framework facilitates data exchange and integration of information involved in bridge management applications. The BrIM framework adopts and extends the data schema of the OpenBrIM standards to facilitate data interoperability among bridge monitoring and management applications. We define data entities to capture information that is needed for bridge engineering analysis, sensor description and bridge monitoring. The BrIM framework also provides data link to the time-series sensor data and image data so as to allow users to locate the data via the information model. Apache Cassandra database (Lakshman and Malik 2010), an open-source column family NoSQL database, is employed as the backend database

system to guarantee flexibility and scalability of the framework. Scripts are written to illustrate data exchange between different data formats, such as Cassandra data schema, BrIM model and data model required by bridge monitoring applications. To demonstrate the NoSQL-based BrIM framework, the bridge model and the sensor data collected from the Telegraph Road Bridge (TRB) located in Monroe, Michigan are employed in this study.

2. BRIDGE INFORMATION MODEL

This section describes a bridge information model designed for bridge monitoring applications. Engineering information modeling, such as BIM, typically adopts an object-based approach in which an information model is composed of objects, each of which contains attributes about the object (Eastman 1999). Information modeling standards and tools specify a predefined set of object families that are used to capture the data entities involved in the targeted domain. For instance, the OpenBrIM standards include object families for describing 3-dimensional geometry of bridge structure (Bartholomew *et al.* 2015). The bridge information modeling (BrIM) schema described herein extends the OpenBrIM schema with newly defined objects for representing engineering analysis model and sensor information. New objects are identified based on relevant standards and software tools. Specifically, we examine CSI Bridge (a structural modeling and analysis software tool) and SensorML (an open standard for sensor description) to ensure that the BrIM is capable of supporting typical applications in bridge engineering.

2.1 OpenBrIM

The OpenBrIM standards describe a bridge structure as a collection of hierarchical objects and their parameters (Bartholomew *et al.* 2015). Each object represents either a physical entity (e.g., beam, column and deck) or a conceptual entity (e.g., project, group and unit system) of a bridge structure. On the other hand, each parameter either represents a property (e.g., length, width and thickness) of an object or refers to another object. Figure 1(a) and (b) show the schema definitions of a basic “Object” entity and a “Parameter” entity, respectively, in OpenBrIM (OpenBrIM 2016). The data schema of the basic “Object” entity includes attributes, such as N (name), X, Y and Z (coordinates), RX, RY and RZ (angles of rotation), and AX, AY and AZ (angles of rotation about the origin of the 3-dimensional workspace). Similarly, the data schema of basic “Parameter” entity includes attributes, such as V (value), T (type), D (description), UC (name of unit system), UT (type of unit), Category (category of the parameter) and Role (specifying whether a user can edit the parameter). The data schema of any other data entities in OpenBrIM is defined by extending the basic “Object” and “Parameter” entities.

To encode bridge information, OpenBrIM standards use ParamML, which is a variation of the extensible markup language (XML) for engineering applications (Bartholomew *et al.* 2015). For example, Figure 2(a) shows the data schema of the Shape object written in the XML schema definition (XSD) format (OpenBrIM 2016). In the schema definition, xs refers to the XML schema namespace (<http://www.w3.org/2001/XMLSchema>). The definitions of the data components from the XML schema namespace are as follows (<http://www.w3schools.com/xml>).

- `xs:complexType` includes other elements and/or attributes.
- `xs:complexContent` specifies extensions or restrictions on a `xs:complexType` element.
- `xs:extension` extends `xs:complexType` element.
- `xs:sequence` defines the child elements that can occur.
- `xs:element` defines an XML element.
- `xs:alternative` dynamically assigns the type of its parent `xs:element` based on the specified test condition.
- `xs:attribute` contains data related to its parent entity.
- `xs:assert` specifies the condition used to validate XML data entity.

```

<xs:complexType name="Object" abstract="true" mixed="false">
  <xs:attribute name="N" type="xs:string" />
  <xs:attribute name="X" type="xs:string" />
  <xs:attribute name="Y" type="xs:string" />
  <xs:attribute name="Z" type="xs:string" />
  <xs:attribute name="RX" type="xs:string" />
  <xs:attribute name="RY" type="xs:string" />
  <xs:attribute name="RZ" type="xs:string" />
  <xs:attribute name="AX" type="xs:string" />
  <xs:attribute name="AY" type="xs:string" />
  <xs:attribute name="AZ" type="xs:string" />
  <!-- The rest is omitted-->
</xs:complexType>

```

(a) Object definition

```

<xs:complexType name="Parameter" abstract="true">
  <xs:attribute name="V" type="xs:string" use="required" />
  <xs:attribute name="T" type="xs:string" default="Expr" />
  <xs:attribute name="D" type="xs:string" />
  <xs:attribute name="UC" type="xs:string" />
  <xs:attribute name="UT" type="xs:string" />
  <xs:attribute name="Category" type="xs:string" />
  <xs:attribute name="Role" type="xs:string" />
</xs:complexType>

```

(b) Parameter definition

Figure 1. Definition of a fundamental Object and Parameters (OpenBrIM 2016)

Given the definitions of the data components, the schema definition for a Shape object as shown in Figure 2(a) specifies the following. The `xs:extension` indicates that Shape is a subtype of “Object” entity. The `xs:sequence` component contains the eligible child objects and parameters using “O” and “P” tags, respectively. As shown in the `xs:alternative` element, the type of child objects is assigned based on the T (type) attribute of the objects. Similarly, the type of child parameters is assigned based on the N (name) attribute of the parameters. Furthermore, the data schema allows rules to be specified for the object. For example, the `use="required"` option in `xs:attribute` enforces that every Shape object must include the T (type) attribute, which has value “shape”. Another example is the `xs:assert` element that specifies the condition that the “A Shape object must contain at least 3 Point objects.”

The schema structure in XSD format can be displayed as an XSD diagram using visualization tools, such as Liquid XML Studio (<https://www.liquid-technologies.com/xml-studio>) and XMLSpy (<https://www.altova.com/xmlspy.html>). For example, the schema structure of the Shape object as shown in Figure 2(a) can be displayed by an XSD diagram as shown in Figure 2(b). It should be noted that the names of the data components are abbreviated in the diagram. In addition, the numbers written at the left side of the XML components represent the possible numbers of the components. For example, “0..*” next to the “O” element in Figure 2(b) means that its parent component (i.e., Shape object) can have zero to any number of child “O” elements (i.e., child objects). In the following sections, we will describe XML data schema using the XSD diagrams for readability purpose. Specifically, we use Liquid XML Studio to display XML schema.

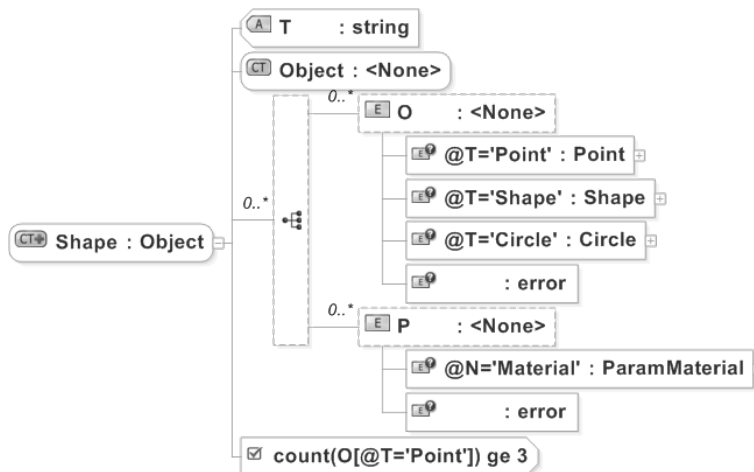
The current OpenBrIM standards include schema definitions for a collection of objects and parameters with particular emphasis on the geometry information of a bridge, but with few entities related to engineering model and material for structural analysis and structural monitoring.

```

<xs:complexType name="Shape" mixed="false">
  <xs:complexContent>
    <xs:extension base="Object">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:element name="O" minOccurs="0" maxOccurs="unbounded">
          <xs:alternative test="@T='Point'" type="Point" />
          <xs:alternative test="@T='Shape'" type="Shape" />
          <xs:alternative test="@T='Circle'" type="Circle" />
          <xs:alternative type="xs:error" />
        </xs:element>
        <xs:element name="P" minOccurs="0" maxOccurs="unbounded">
          <xs:alternative test="@N='Material'" type="ParamMaterial" />
          <xs:alternative type="xs:error" />
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Material" type="xs:string" />
      <!-- omitted -->
      <xs:attribute name="T" type="xs:string" fixed="Shape" use="required" />
      <xs:assert test="count(O[@T='Point']) ge 3"
        xerces:message="Shape object must contain minimum 3 Point object(s)." />
      <!-- omitted -->
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

(a) Data schema in XSD format



(b) XSD diagram

Figure 2. Data schema of the Shape object (OpenBrIM 2016)

2.2 Finite Element modeling

The OpenBrIM standards currently include only a few basic objects for finite element (FE) modeling (OpenBrIM 2016). The objects defined in the OpenBrIM standards are insufficient for FE modeling because structural analysis software tools often involve much more complex data entities. Using CSI Bridge, one of the widely used commercial bridge modeling and analysis tools (Computer & Structures, Inc. 2016), as an example, an FE model for an overpass bridge would consist of about fifty tables, where each table contains several attributes, many of which are not defined in the current OpenBrIM standards. For the representation of FE model, we extend the OpenBrIM standards' model by adding the data entities required by CSI Bridge software to OpenBrIM standards' data schema definition.

While information models for finite element analysis exist (such as STEP Part 104 (ISO 2000) and Industry Foundation Classes (buildingSMART 2016)), the existing models usually lack data entities to represent complex load and analysis conditions (such as time-variant vehicle load) required in bridge engineering applications. This study focuses specifically on FE model for bridge engineering applications. In this work, the data entities for FE modeling are divided mainly into two categories: data entities for representing bridge structure and data entities for representing load and analysis conditions. The OpenBrIM standards include some of the objects that can be extended to represent bridge structure information. On the other hand, the OpenBrIM standards include very limited objects for representing load and analysis conditions. Therefore, we focus on enhancing the data entities of existing objects with new parameters and child objects for data entities for representing a bridge structure, as well as on defining new objects for representing the load and analysis conditions.

2.2.1 Data entities for representing bridge structure

The OpenBrIM standards include `Node`, `FELine`, `FESurface` and `Material` objects that can be used for the representation of the geometry and material properties of bridge structures. However, the current OpenBrIM schema definitions of these objects are not sufficient for describing an FE model for a typical software tool. A `Node` object, which is the most fundamental data entity in FE modeling, specifies the nodal coordinates and restraints. The `Node` object as currently defined in the OpenBrIM standards, however, lacks parameters necessary for defining a reference coordinate system, which is convenient for creating models by using multiple coordinate systems. To augment the `Node` object in OpenBrIM, we create a new object `FECordinateSystem` that includes information about the coordinate type and the origin of the reference system. We add to the data schema of `Node` object a reference to `FECordinateSystem` as shown in the XSD diagram in Figure 3.

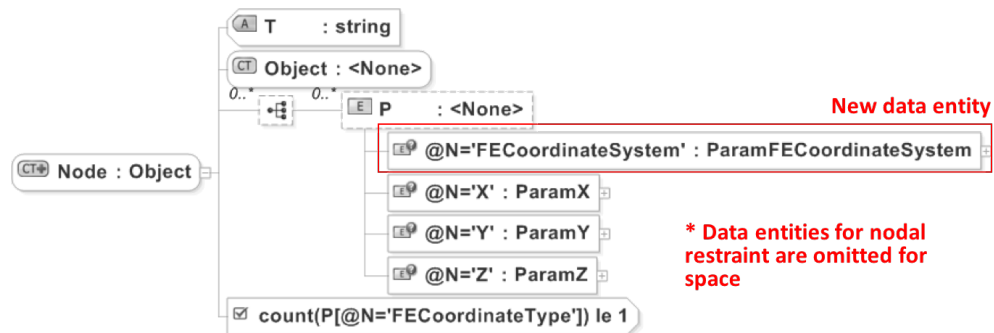


Figure 3. Entity: Node

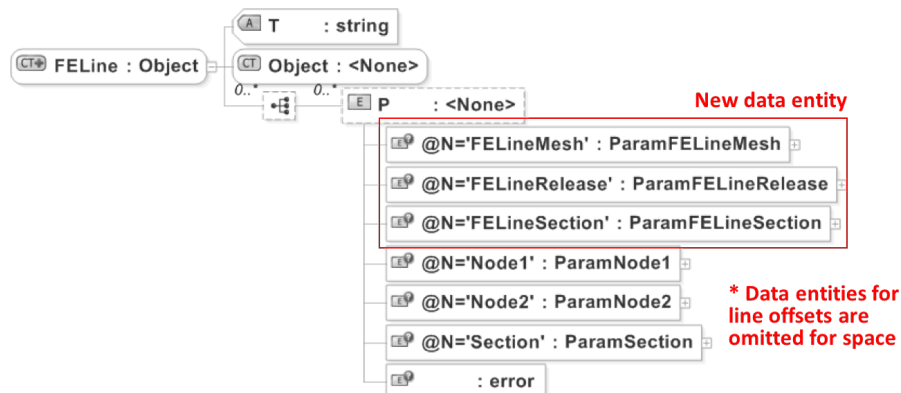


Figure 4. Entity: FELine

An `FELine` object represents an element that consists of two nodes and section information in an FE model. The current definition of `FELine` object in OpenBrIM includes data entities for describing the two `Nodes` and `Section`, but the object definition does not include data entities to represent information

about discretization and member-end-release. In addition, the `Section` object, while it is suitable to represent user-defined section shape composed of many section points, does not allow representing standard section shapes that are described by other parameter types. We extend the description of `FELine` by creating new objects, namely `FELineMesh`, `FELineRelease` and `FELineSection` to represent mesh information, member-end release information and standard section shapes, respectively. The data schema diagram of the `FELine` object that includes parameters referring to the new objects is shown in Figure 4.

Similarly, the current definition of `FESurface` object in OpenBrIM represents an element consists of vertices, thickness and material types, such as shell and wall, but the schema does not include discretization information, section information (e.g., surface type and material angle) and surface-constraint information (e.g., edge constraint). Furthermore, the current `FESurface` object can only have up to four vertices, while elements such as shell element can compose of more than four vertices. To extend the description of the `FESurface` object, we create new objects, namely `FESurfaceMesh`, `FESurfaceSection` and `FESurfaceConstraint`, to represent, respectively, the discretization information, section information, and surface constraint information. We also increase the number of vertices (i.e., `Nodes`) that an `FESurface` object can contain up to thirty vertices. (It should be noted that the number can easily be modified. Furthermore, as discussed later, a NoSQL database allows variable length records to easily handle any number of vertices in an element.) The data schema diagram of the enhanced `FESurface` object is shown in Figure 5.

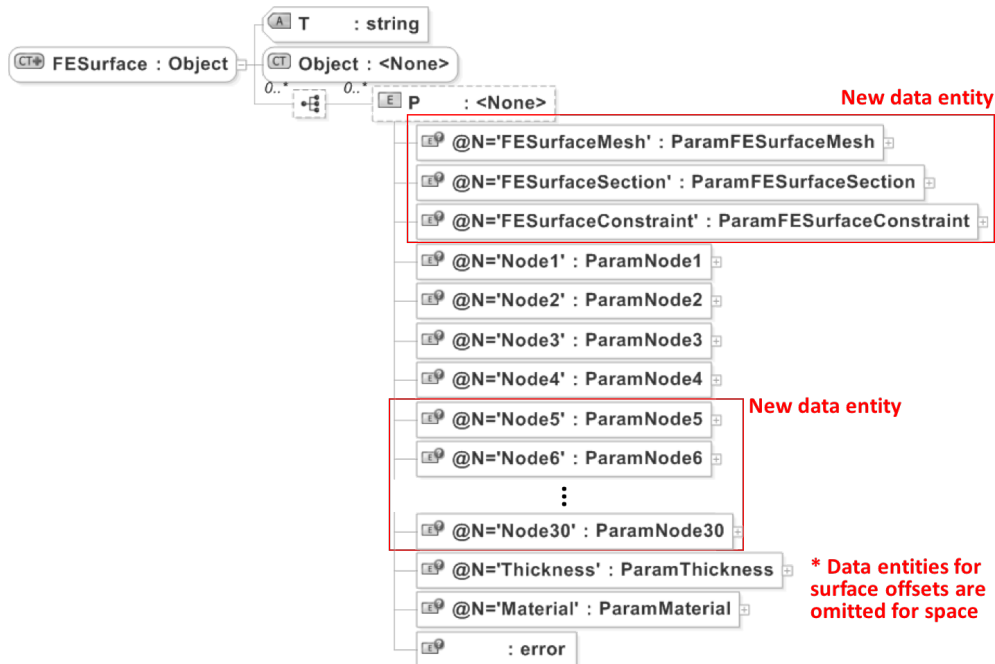


Figure 5. Entity: `FESurface`

A `Material` object defined in OpenBrIM is used to represent material property data for concrete and steel elements. The `Material` object includes basic material properties, such as modulus of elasticity, Poisson ratio, density, steel yield stress and concrete 28-day strength. To further enhance the definition of the `Material` object for structural analysis purpose, we add new parameters, such as `Symmetry`, `TemperatureDependency`, `ShearModulus` and various damping properties. The enhanced `Material` object can describe uniaxial and isotropic materials in linear analyses. Currently, the definition of `Material` object does not include material properties for describing orthotropic materials and for performing nonlinear structural analysis. Figure 6 shows the data schema diagram of the enhanced `Material` object.

In addition to the entities described above, new data entities and their parameters are defined, as summarized in Table 1, to complete the schema definitions of the Node, FELine, FESurface and Material objects in OpenBrIM.

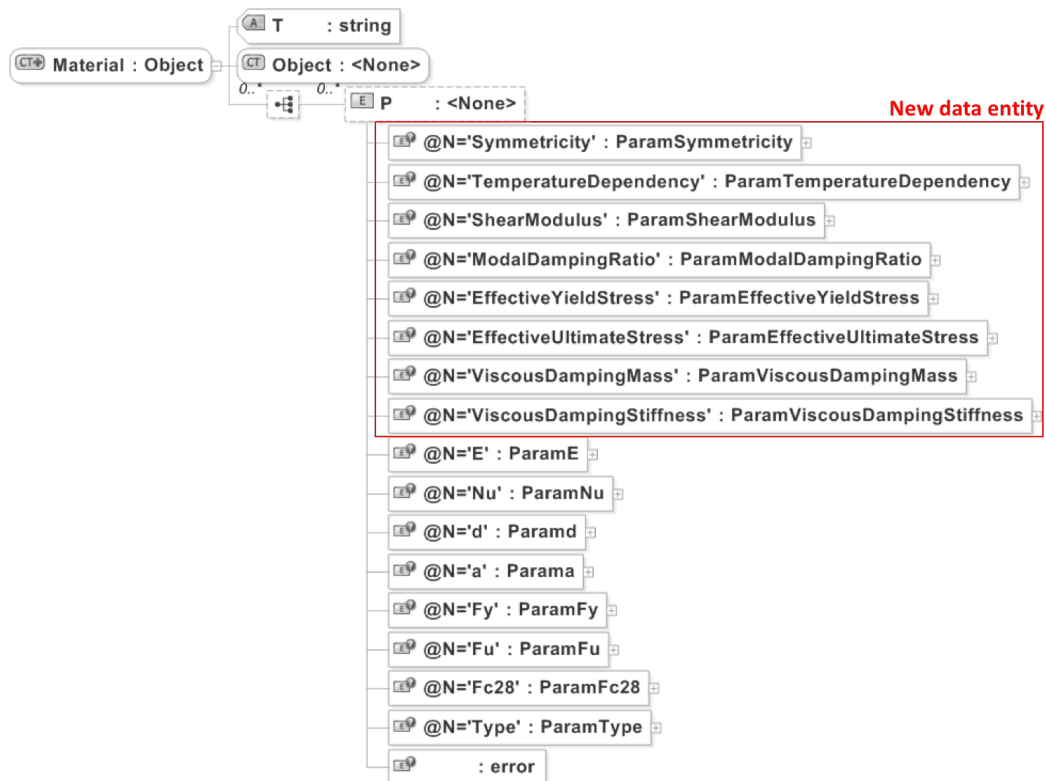


Figure 6. Entity: Material

Table 1. Objects added to the OpenBrIM model for representing structural elements

| Object | Parameters |
|----------------------------|---|
| <i>FECordinateSystem</i> | FECordinateType, OriginX, OriginY, OriginZ, OriginRX, OriginRY, OriginRZ |
| <i>FELineMesh</i> | AutoMesh, MeshAtJoints, MeshAtFrames, NumberOfSegments, MaxMeshLength, MaxMeshDegree |
| <i>FELineSection</i> | Material, Shape, Width, Height, WebThickness, FlangeThickness |
| <i>FELineRelease</i> | NodeV1, Node1V2, Node1V3, Node1M1, Node1M2, Node1M3, Node2V1, Node2V2, Node2V3, Node2M1, Node2M2, Node2M3 |
| <i>FESurfaceMesh</i> | Meshtype, MeshGroup, NumberOfObject, NumberOfObject2, MaxSize1, MaxSize2, MeshFromSelectedLine, MeshFromSelectedPoint, ConstraintEdge, ConstraintFace |
| <i>FESurfaceSection</i> | Material, MaterialAngle, SurfaceType, Thickness, BendThickness |
| <i>FESurfaceConstraint</i> | EdgeConstraint |

2.2.2 Data entities for representing load and analysis conditions

OpenBrIM standards include AnalysisCase, NodeLoad and Combination objects for the representation of load conditions and analysis conditions (OpenBrIM 2016). While these objects are able to describe simple load conditions, they do not have sufficient detailed information to describe complex load conditions (e.g., time-variant loading) and detailed analysis conditions (e.g., convergence tolerance). CSI Bridge, for example, describes load conditions and analysis conditions using “load patterns” and “load cases”, respectively (CSI Bridge 2016). The load patterns are the spatial distribution of forces and other effects acting on a structure, while the load cases are the analysis options that include applied load pattern, type of response, and type of analysis. The load patterns and load cases are further divided into many different types of loads

(e.g., dead load, wind load and moving load) and different cases of analyses (e.g., static analysis, modal analysis, multi-step static analysis and time history analysis). Instead of extending the existing objects, we define a set of new objects based on the data entities defined in CSI Bridge software to describe practical load and analysis conditions.

We create a new object `FELoadPattern` to represent data corresponding to load patterns. The data schema of the `FELoadPattern` object is shown in Figure 7. The new object `FELoadPattern` has parameters representing the types of a load (`LoadType`) and the self-weight factor (`SelfWeightFactor`). The `FELoadPattern` may have child objects that contain the details of specific load patterns. For example, a new child object `FEMultiStep` is created to contain information about the vehicle crossing the bridge, its traveling lane and speed. Furthermore, a new object `FELane` is created to describe the vehicle lane information, including referencing objects, stations (i.e., longitudinal distance from the referencing objects) and width of the lane. We also create `FEVehicle` object and its child object `FEVehicleLoad` to capture vehicle axle load data. `FEVehicle` object includes parameters for the name of the vehicle, a scale factor and the number of axle loads, and `FEVehicleLoad` object includes parameters such as the type of load (e.g., uniform load, axle load), width of an axle, and distance between axles.

For the representation of analysis conditions, we create a new object `FEAnalysisCase`. Figure 8 shows the schema diagram of `FEAnalysisCase`. The `FEAnalysisCase` object contains descriptions for different types of analysis. The parameters are `LoadType` (e.g., dead load and live load), `AnalysisCaseType` (e.g., static, modal, multistep-static and direct integration time history analysis) and `InitialCondition`. Furthermore, `FEAnalysisCase` consists of child objects for specific analysis cases. For instance, `FESstatic` contains data entities for representing a static analysis case and the `FEModal` contains data entities for representing a modal analysis case. The object `FEMultiStepStatic` includes data entities for representing the applied vehicle and the object `FEDirectIntegrationHistory` includes data entities about the time-step information. Table 2 summarizes the new data entities created under the `FELoadPattern` and `FEAnalysisCase` objects to represent load and analysis conditions.

Table 2. Objects added to the base OpenBrIM model for representing load and analysis conditions

| Object | Parameters | Child object |
|------------------------------------|---|---------------------|
| <i>FEMultiStep</i> | LoadDuration, LoadDiscretization, Vehicle, Lane, Station, StartTime, Direction, Speed | - |
| <i>FEVehicle</i> | VehicleName, NumLoad | FEVehicleLoad |
| <i>FEVehicleLoad</i> | LoadType, UniformLoad, UniformType, AxleLoad, AxleType, AxleWidth, MinDistance, MaxDistance | - |
| <i>FEVehicleClass</i> | VehicleName, ScaleFactor | - |
| <i>FELane</i> | LaneFrom, ReferenceLayout, ReferenceFrame, Station, Width, Offset, Radius, DiscretizationAlongLane, DiscretizationAcrossLane, LeftEdgeType, RightEdgeType | - |
| <i>FESstatic</i> | LoadType, LoadName, ScaleFactor | - |
| <i>FEModal</i> | ModeType, MaxNumModes, MinNumModes, FrequencyShift, CutoffFrequency, ConvergenceTolerance | - |
| <i>FEMultiStepStatic</i> | LoadType, LoadName, ScaleFactor | - |
| <i>FEDirect-IntegrationHistory</i> | NumStep, LoadType, LoadName, Function, ScaleFactor, TimeFactor, ArrivalTime, IntegrationMethod, Alpha/Beta/Gamma (Integration parameters) | - |

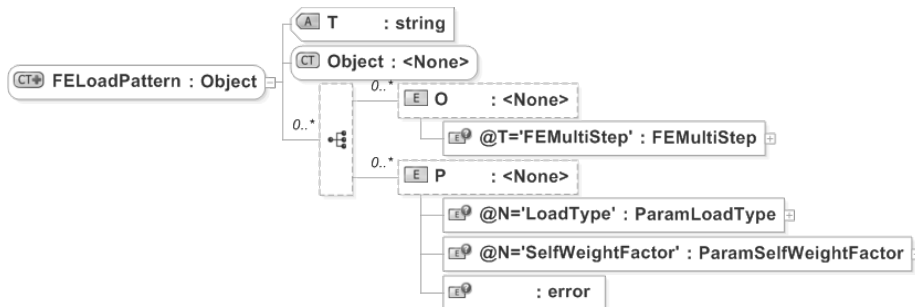


Figure 7. New entity: FEMultiStep

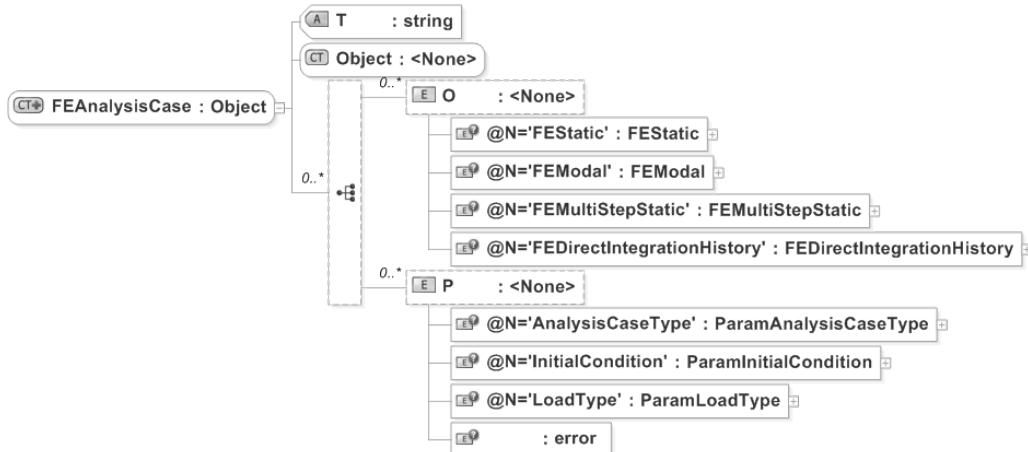


Figure 8. New entity: FEAnalysisCase

2.3 Sensor description

There have been several standards developed to describe sensor information and measurement data (Lee 2000, Pschorr *et al.* 2010). The Sensor Web Enablement (SWE) standards by the Open Geospatial Consortium (OGC) are among the most common suites adopted by the sensor web community. The SWE includes suites of standards such as Sensor Model Language (SensorML), Observation & Measurement (O&M) and Sensor Observation Service (SOS). Among these standards, SensorML standards provide XML-based data format for the description of sensor metadata as well as processes and processing components associated with the sensors (Open Geospatial Consortium 2014). In this work, we draw on the data entities defined by the SensorML standards to enable the BrIM schema to include the sensor information. Specifically, we import SensorML's schema and namespace to the BrIM schema by adding XSD codes as shown in Figure 9. The "sml" prefix in the figure refers to the namespace of SensorML standards.

```

<xs:schema id="openbrim3" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sml="http://www.opengis.net/sensorml/2.0"
  xmlns:swe="http://www.opengis.net/swe/2.0"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  xmlns:xerces="http://xerces.apache.org"
  elementFormDefault="qualified" vc:minVersion="1.1">
  <xs:import namespace="http://www.opengis.net/sensorml/2.0"
    schemaLocation="http://schemas.opengis.net/sensorML/2.0/sensorML.xsd" />
  <xs:import namespace="http://www.opengis.net/swe/2.0"
    schemaLocation="http://schemas.opengis.net/sweCommon/2.0/swe.xsd" />
  ...
</xs:schema>

```

Figure 9. Importing SensorML's schema

For the description of sensors, we mainly use two SensorML elements, namely `DescribedObjectType` and `PositionUnionPropertyType`. The `DescribedObjectType` contains a rich set of data entities to encode common sensor information as follows (Open Geospatial Consortium 2014).

- *keywords* are short strings that can be understood by the general users or certain community of users.
- *identification* includes the terms (e.g., long name, short name, serial number and manufacturer) that are used to identify sensors.
- *classification* includes terms (e.g., sensor type and intended application) that can be used to classify sensors.
- *validTime* denotes the time period during which the sensor information is valid.
- *securityConstraints* describe security tags for the sensor description document.
- *legalConstraints* define the legal terms (e.g., privacy acts, intellectual property rights and copyrights) for the sensor information.
- *characteristics* represent the physical properties (e.g., dimension and weight) and the electrical requirements (e.g., voltage and current) of the sensor.
- *capabilities* are the properties (e.g., sensing range, sensitivity and threshold) that describe the sensor measurement outputs.
- *contacts* refer to the information about the person or group (e.g., manufacturers, experts and equipment owner) with knowledge of the sensor.
- *documentation* refers to the additional information (e.g., manual, datasheets and images) from external sources
- *history* records the list of events (e.g., calibration event and maintenance event) related to the sensor.

Additionally, the `PositionUnionPropertyType` includes data entities to describe the sensor location in a number of formats (e.g., textual form, coordinate and vector), so that users can choose the most appropriate way to describe the sensor location.

Based on the `DescribedObjectType` and `PositionUnionPropertyType` defined in SensorML, we define `SensorMetadata` and `SensorLocation` objects to describe sensor metadata and sensor location, respectively. While we define the `SensorMetadata` object by simply referring to the `DescribedObjectType`, we add optional data entities to the `PositionUnionPropertyType` to enable `SensorLocation` object to describe the sensor location in a structural monitoring system. One of the added entities is the `TargetObject`, whose value type is string to refer to the ID of geometric element that the sensor is attached to. Another entity added is the `FENode`, whose value type is also string, to include the ID of finite element model's node in the case that the sensor position coincides with the node.

To describe a complete sensor that includes both the metadata and location information, we create an object called `Sensor`. A `Sensor` object includes parameters referring to the `SensorMetadata` and `SensorLocation` objects. The data schema diagram of the `Sensor` object is shown in Figure 10.

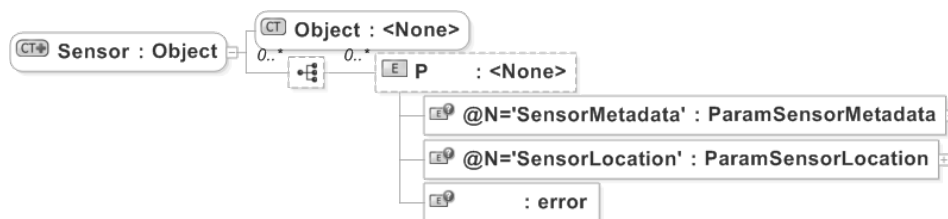


Figure 10. New entity: Sensor

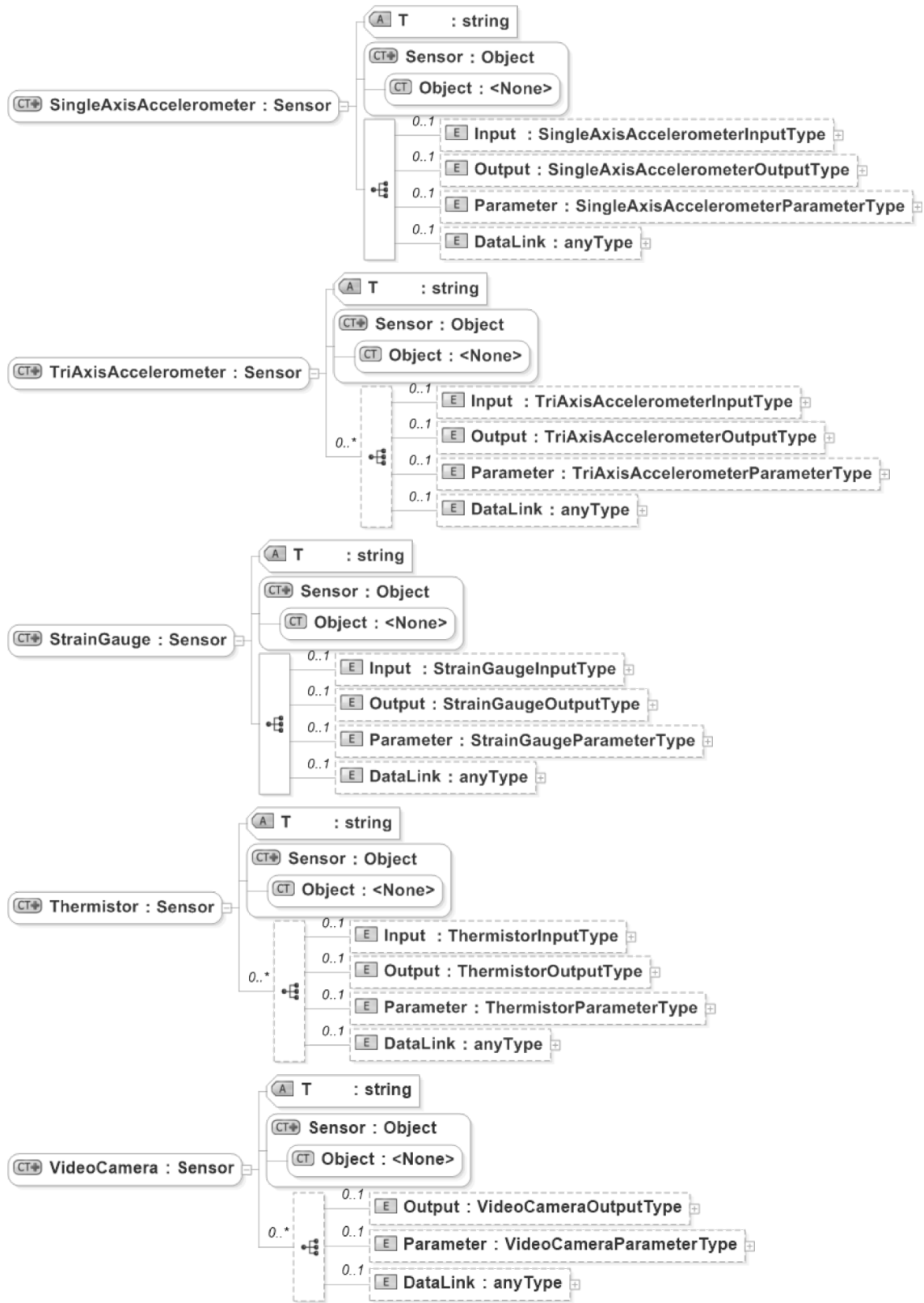


Figure 11. New entities: Sensor subtype objects

As shown in Figure 11, the sensors, such as *SingleAxisAccelerometer*, *TriAxisAccelerometer*, *StrainGauge*, *Thermistor* and *VideoCamera* that are commonly employed in bridge monitoring, are defined as subtypes for the *Sensor* object. Furthermore, each of the subtype objects contains data entities describing the input, output, parameters and data link for the particular sensors. Table 3 summarizes the new data entities for the different sensor types. The data entities (as denoted as grandchild elements in Table 3) of the subtype objects are designed to reflect the features of each object. For example, the *SingleAxisAccelerometer* object has a single input element and a single output element, while the *TriAxisAccelerometer* object has three inputs and three outputs to represent 3-dimensional acceleration measurements. Last but not least, the *DataLink* entity is defined to allow linking to the data storages of the sensor measurements.

Table 3. Data entities included in the Sensor subtype objects.

| Sensor subtype object | Child element | Grandchild elements |
|--------------------------------|---------------|--|
| <i>SingleAxisAccelerometer</i> | Input | RawAccelerationData |
| | Output | Acceleration |
| | Parameter | Gain, ConversionFactor, SamplingRate |
| | Datalink | - |
| <i>TriAxisAccelerometer</i> | Input | RawAccelerationDataX, RawAccelerationDataY, RawAccelerationDataZ |
| | Output | AccelerationX, AccelerationY, AccelerationZ |
| | Parameter | Gain, ConversionFactor, SamplingRate |
| | Datalink | - |
| <i>StrainGauge</i> | Input | RawStrainData |
| | Output | Strain |
| | Parameter | Gain, ConversionFactor, SamplingRate |
| | Datalink | - |
| <i>Thermistor</i> | Input | RawTemperatureData |
| | Output | Temperature |
| | Parameter | C1/C2/C3 (J-Curve Coefficient), SamplingRate |
| | Datalink | - |
| <i>VideoCamera</i> | Output | Image |
| | Parameter | FramePerSecond, Resolution |
| | Datalink | - |

3. A BRIDGE INFORMATION REPOSITORY FRAMEWORK

This section discusses the data repository framework for managing the bridge model and monitoring information. The overall bridge information management framework is shown in Figure 12. The bridge information model is mapped to the database schema and stored in the main server. An onsite computer receives sensor data from the sensor network and transmits the data to the server. An inspection tool records the information from bridge inspectors and sends the information to the database system. A database system is implemented on the main server to store the sensor data and inspection data according to the bridge information model.

Given the large volume of modeling and measurement data and the wide variety of data types, it is important to select an appropriate database system so as to facilitate efficient storage, sharing, retrieval and utilization of the bridge information. For scalable and flexible bridge information management, Jeong *et al.* (2016) have proposed a NoSQL-based data management framework for bridge monitoring applications. In this study, we use Apache Cassandra column-oriented NoSQL database (Lakshman and Malik 2009) for the data store. As for the applications, we use CSI Bridge (version 2015) as an engineering analysis tool. Python programming language is employed for implementing the interface among the software components as well as the scripts for data mapping. Interfaces utilizing data manipulation language (DML) and application programming interfaces (APIs) of the Cassandra database system are developed to support data retrieval. In addition, the data mapping scripts are written to ensure seamless data exchange between the bridge information model and the database models.

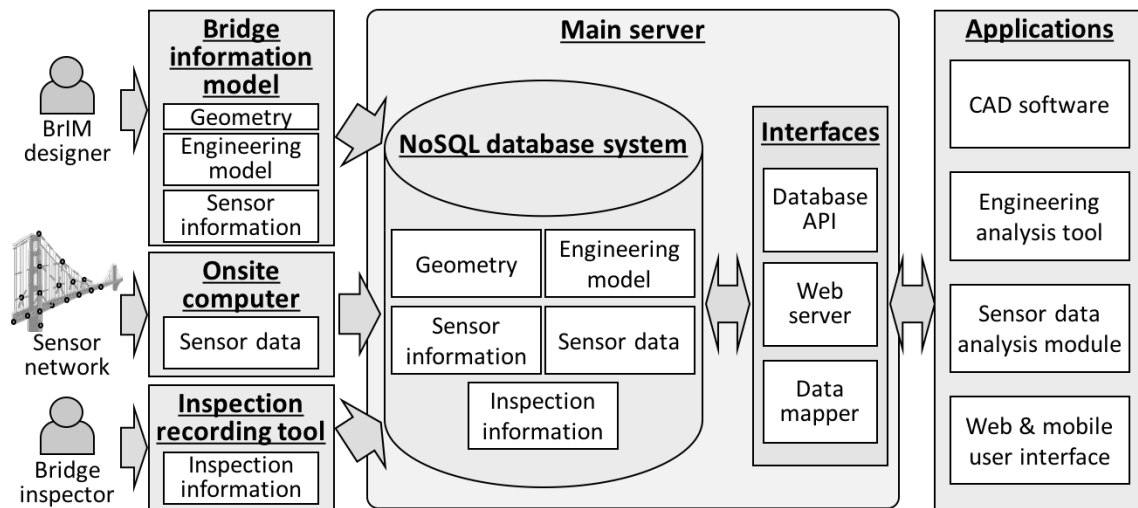


Figure 12. Bridge information management framework

3.1 Data schema definition

For the column-oriented Cassandra system, the basic feature for defining a database consists of “key space,” “column family,” “row” and “column,” which are analogous to, respectively, “database,” “table,” “row” and “column” of relational database (RDB) (Hewitt 2010). The column-oriented data store is known to be more flexible than the tabular structure of RDB. In the column-oriented data model, each row can have different sets of columns and new columns can be dynamically added to a row without changing the data schema of other rows, whereas RDB requires the rows in a table having the same schema (i.e., same number of column attributes). Each row in a column family has a row key as a unique identifier, and each column stores the data using a name-value pair. The flexible data model is particularly suitable to manage the bridge information captured by the object-oriented BrIM schema, where the objects contain different sets of information. Using the data definition language (DDL) of Cassandra database, we define column families and their schema based on the object definitions of the BrIM schema. In the current framework, we store all the data entities of an object in a single Cassandra column family. For example, Figure 13 shows data mapping between the BrIM object `FEAnalysisCase` and the corresponding Cassandra column family `FEAnalysisCase`. The column family contains all the data entities corresponding to the data entities of `FEAnalysisCase` object in BrIM with additional data entities defined for unique ID (`uid`) and the hierarchical relation (`parent` and `child`).

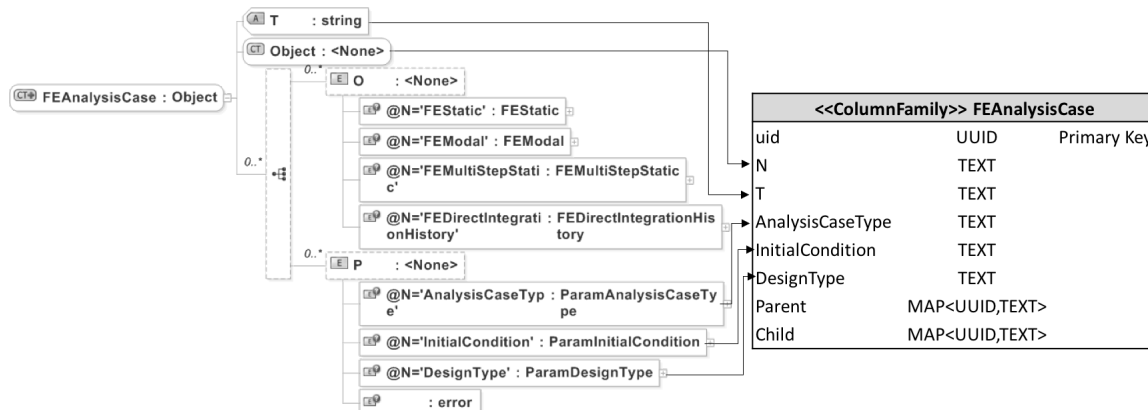


Figure 13. Data mapping between BrIM object and Cassandra column family

In our approach, a single row stores the information corresponding to a single object of bridge information model using a series of columns, each of which stores an attribute or a parameter of the object. Figure 14 shows an example of storing BRIM objects to the Cassandra database. In this example, two `FESurface` objects contain different numbers of `Node` parameters. Although these two objects have different sets of data entities, the Cassandra database can store the `FESurface` objects in the same column family without requiring two separate rows having the same schema. In addition to the data entities defined for an object, the bridge information model composes hierarchical relationships among the objects. Figure 15 shows a database schema example for storing a hierarchy of objects, where a single `Shape` object has four child `Point` objects specifying the four vertices of the shape. To store the information of the `Shape` object while maintaining the hierarchical relationship, we define the `parent` and `child` columns to store the row keys and the object type corresponding to the parent object and child objects, respectively. In Figure 15, the row for storing the `Shape` object has a `child` column that contains the row keys of the `Point` objects, namely, `pt001`, `pt002`, `pt003` and `pt004`. The rows for the `Point` objects have the `parent` column storing the row key of the `Shape` object “`shp001`”.

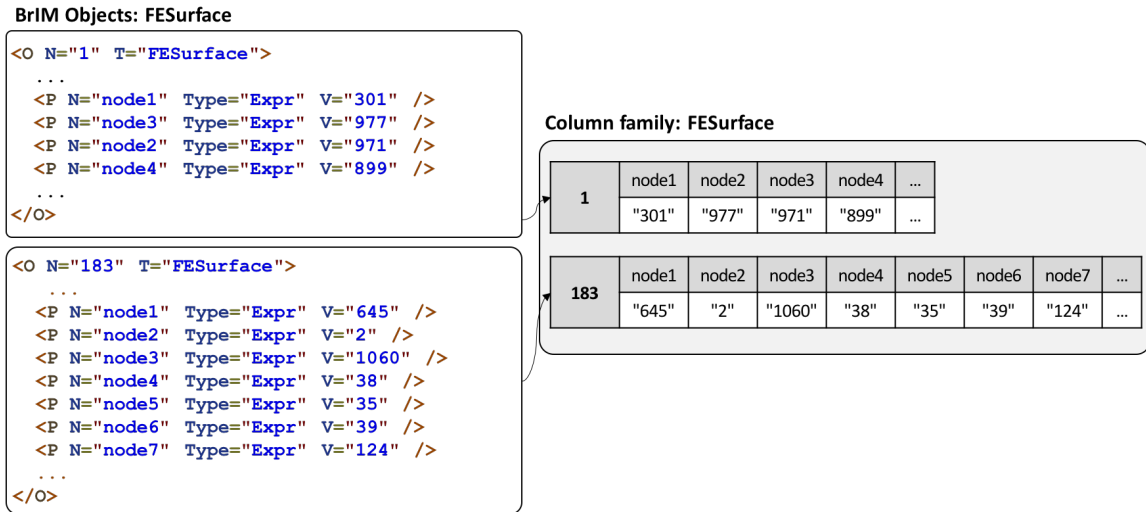


Figure 14. Storing BRIM objects to Cassandra database

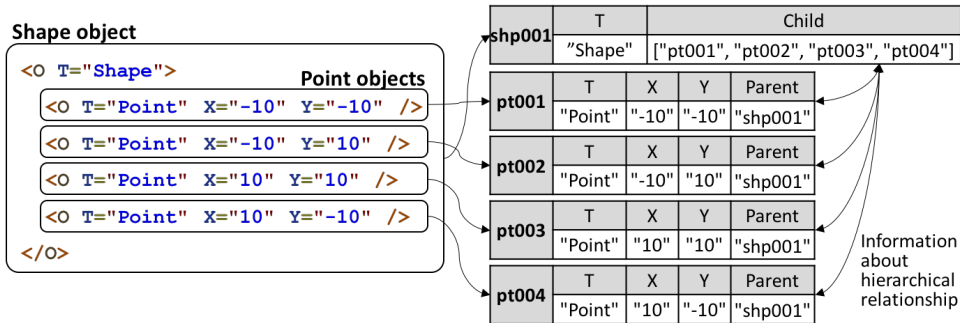


Figure 15. Storing hierarchical BRIM objects to Cassandra database

For sensor data, a column family `sensordata` is defined. The most important consideration when dealing with sensor data is that the sensor data typically involves range query for retrieving a sequence of time-series data. For efficient performance, we use the time-series data modeling strategy of Cassandra database (DataStax 2016b). The timestamp of sensor data is assigned to a clustering key of the database so that the sensor measurements are stored in contiguous locations within a disk in a sorted order. In addition, we use part of the timestamp (such as year and month) in the row key to partition the row according to the date that the data collected, thereby preventing excessive amount of data stored in a single row. Similarly, a column family `imagedata` is defined to manage image data collected from video cameras. Here, the image

data which is converted to binary format is stored as binary large object (BLOB). The data schema of `sensordata` and `imagedata` column families are shown in Figure 16. The sensor measurements and the images stored in the column families can be accessed via the `DataLink` entities defined in the subtypes for the `Sensor` object.

| <<ColumnFamily>> sensordata | | | <<ColumnFamily>> imagedata | | |
|-----------------------------|--------------|----------------|----------------------------|-----------|----------------|
| sensor_id | TEXT | Primary Key | camera_id | TEXT | Primary Key |
| month | TEXT | Primary Key | date | TEXT | Primary Key |
| event_time | TIMESTAMP | Clustering Key | event_time | TIMESTAMP | Clustering Key |
| data | LIST<DOUBLE> | | image | BLOB | |

Figure 16. data schema of `sensordata` and `imagedata` column families

3.2 Query and mapping

Once the data schema is defined, bridge information can be stored and retrieved using the Cassandra query language (CQL) (DataStax 2016a), which is similar to the structural query language (SQL) of RDB. Specifically, CQL uses `INSERT-INTO-VALUE` statement for data insertion and `SELECT-FROM-WHERE` statement for data retrieval. Figure 17 shows an insert query and a retrieval query. The query statement in Figure 17(a) inserts a row, which has `uid`, `N`, `T`, `x`, `y`, `z` values, to the `Node` column family, and the query in Figure 17(b) retrieves the `uid` from the `Node` column family where the `N` (Name) of the `Node` is “100”. Data insertion and retrieval request can be carried out by sending the query statements to Cassandra database system via either the CQL shell (CQLSH) interface or Cassandra Driver API. However, CQL has limited query possibilities comparing to SQL. One limitation is that CQL does not support complex query, such as to combine two or more column families, through “join” operation. To implement complex query, an application script can be used to encode the queries and to pass the query result from one query onto another.

| | |
|---|--|
| <pre>INSERT INTO Node (uid, N, T, x, y, z) VALUES (UUID(), '100', 'Node', '0', '0', '0');</pre> | <pre>SELECT uid FROM Node WHERE N = '100';</pre> |
| (a) Insert query | (b) Retrieval query |

Figure 17. Cassandra query example

The bridge information described in BrIM data model needs to be mapped to the data schema of the Cassandra database, and vice versa. For data mapping, we develop scripts using Cassandra driver API (DataStax 2016c) to interact with Cassandra database and an XML parser (such as `xml.etree.ElementTree` package (Python Software Foundation 2016a)) for parsing and modifying the information written in XML. Figure 18 shows an example of data mapping from BrIM file to Cassandra database. First, the BrIM file written in XML is parsed into an object-tree using the XML parser. The parsed objects include information about its attributes, parameters, and parent and child objects. The mapping script then accesses the root object in the object-tree, which is the `Project` object in the example, and maps the information of the object into the data model of Cassandra database. The mapped information is then used to create an `INSERT` query request. Finally, the query request is sent to the Cassandra database using Cassandra driver API. The process is performed recursively for the child objects in the object-tree until child objects have been processed.

Data mapping from Cassandra database to BrIM file can be done by reversing the mapping process for translating the BrIM file to Cassandra database. Figure 19 shows an example that retrieves the `Project` object and its child objects, which are stored through the mapping process shown in Figure 18. For the retrieval of the bridge information, we first execute a `SELECT` query to obtain the root object (i.e., `Project` object in this example) using the Cassandra Driver API. The query result is then mapped into the XML object using the XML parser. Using the child object list in the `child` column, the process is performed recursively for all the child objects in the object-tree. Once the data retrieval process is complete, the object-tree is parsed as XML string and stored in an XML file.

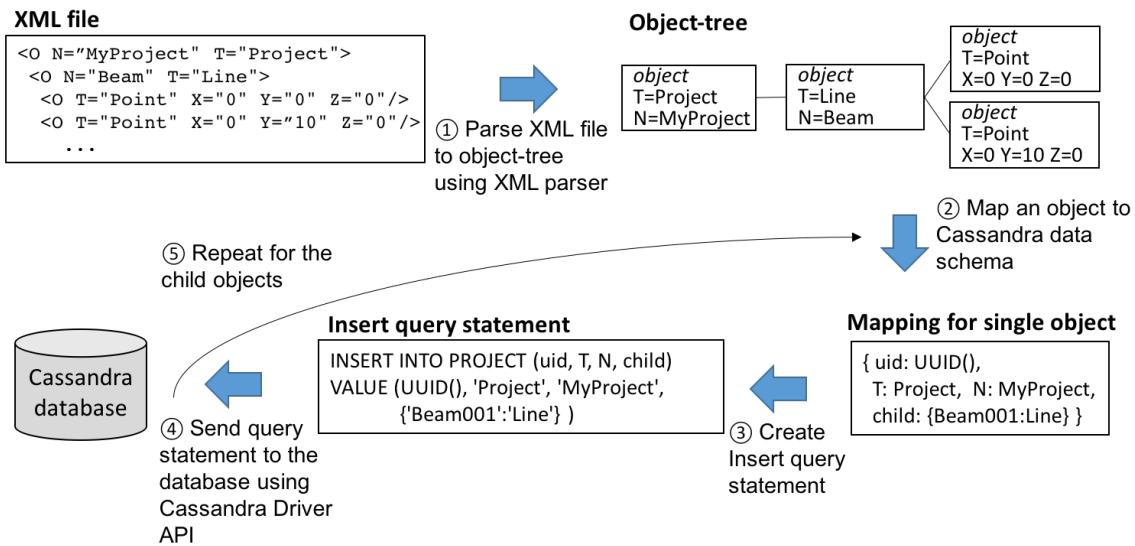


Figure 18. Data mapping from BrIM schema to Cassandra database schema

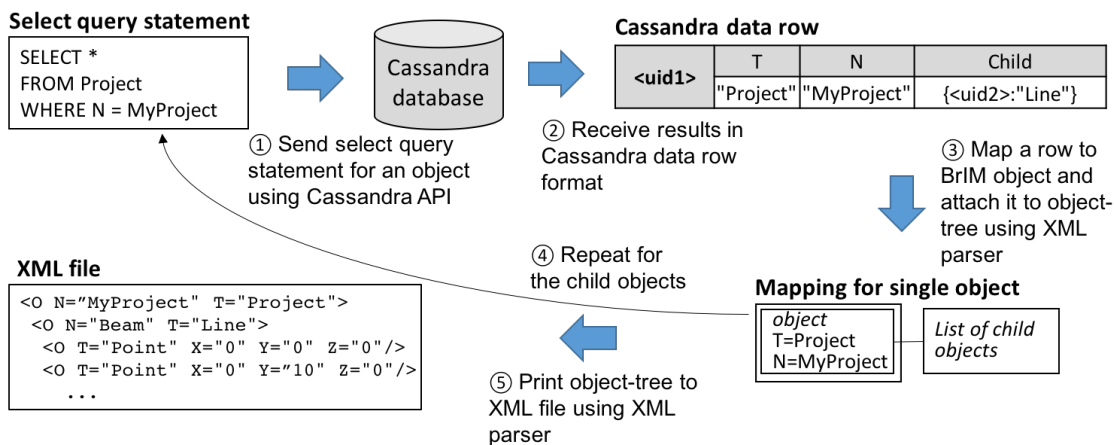


Figure 19. Data mapping from Cassandra database schema to BrIM schema

4. CASE EXAMPLE

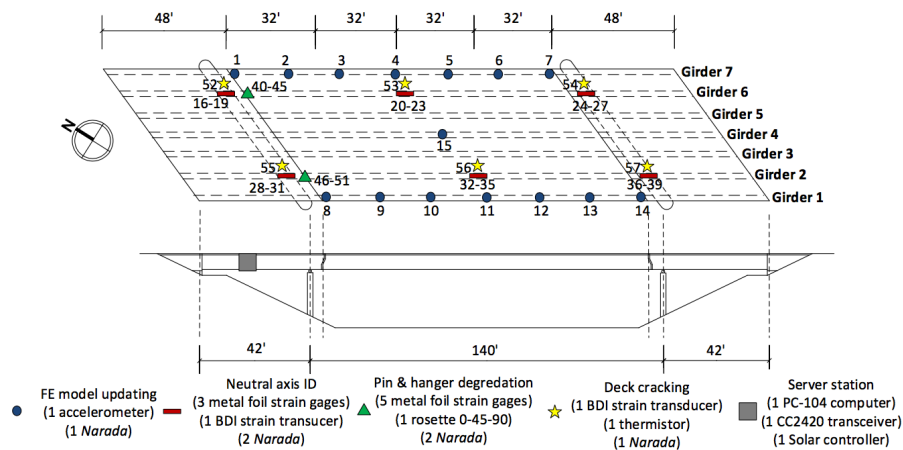
This section describes a case example using the Telegraph Road Bridge (TRB) located in Monroe, Michigan, to demonstrate the utilization of the bridge information model and the information repository. Two scenarios involving different types of bridge information are presented. The first scenario compares the bridge response collected by the sensors with that obtained from a structural analysis software. The second scenario illustrates the retrieval of sensor data along with the corresponding traffic-image data.

The TRB, as shown in Figure 20(a), is a 224 feet long (about 68 meters) steel girder bridge along the I-275 corridor in Michigan. The TRB is instrumented with wireless sensor network that consists of 60 sensors, including accelerometers, strain gauges and thermistors, as shown in Figure 20(b) (Zhang *et al.* 2016). In addition, there is a public traffic monitoring system managed by the Michigan Department of Transportation (MDOT) (<http://mdotnetpublic.state.mi.us/drive>). In this study, we create an XML-based bridge information model of the TRB that covers the geometry, engineering model and sensor description as follows:

- *Geometry*: Based on the 2-dimensional drawings of the TRB, a 3-dimensional geometric model is constructed following the OpenBrIM data schema (ParamML 2016). The geometric model is created in XML using a text editor.
- *Engineering model*: The engineering analysis model is created by converting the finite element (FE) model of TRB created using the CSI Bridge software (Hou *et al*, 2015). To convert FE model to the BrIM model, the CSI Bridge file is first exported to Microsoft Excel Spreadsheets using the “export” function of the software. The exported Excel Spreadsheets are then mapped into the BrIM model by using an Excel data parser (namely, the openpyxl package (Gazoni and Clark 2016)), an XML parser (the xml.etree.ElementTree package (Python Software Foundation 2016a)) and a data mapping script specifying the mapping rules between CSI Bridge’s Excel data and the BrIM data model.
- *Sensor description*: Based on the sensor information of the TRB, *Sensor* objects are created. In the object-tree, a *Sensor* object is assigned as a child object of the geometry object where the sensor is attached. For the demonstration purpose, *Sensor* objects include an *FENode* entity to refer to the closest node in the engineering model, although their locations may not coincide exactly with each other.



(a) Side view



(b) Sensor layout (Zhang *et al*. 2016)

Figure 20. Telegraph road bridge (located in Monroe, Michigan)

The created bridge information model is loaded onto the Cassandra database using the data mapping process from BrIM to Cassandra as described earlier and illustrated in Figure 18. In addition to the information described in BrIM model, sensor data and traffic monitoring images are also collected and stored in the database. The wireless sensor network installed on TRB collects acceleration, strain and temperature measurements for one-minute durations every two hours. The sampling rates vary from 100Hz for strain

gauges and thermistors to 200Hz for accelerometers. Traffic monitoring images are collected every 2 to 5 seconds from the MiDrive system (<http://mdotnetpublic.state.mi.us/drive>) by MDOT.

4.1 Example scenario 1: Comparison of sensor data and analytically computed bridge response

In the first example scenario, we compare the bridge response measurements collected by the accelerometers with the bridge responses computed at the FE nodes corresponding the accelerometer locations. Since the bridge information, the sensor information and the FE model are integrated in the database, scripts can be written to automate the process. Figure 21 shows the basic steps implemented for this example scenario.

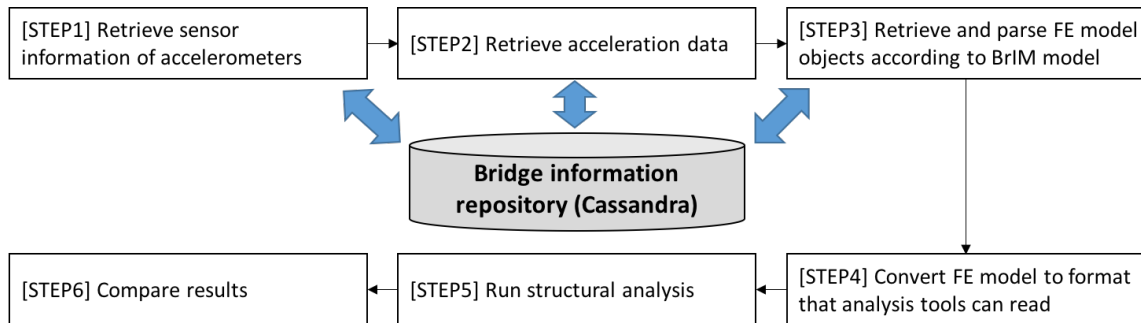


Figure 21. Workflow of scenario 1

As shown in Figure 21, the comparison of measurement and computed data are implemented in six steps. In step 1, we retrieve sensor information from the Cassandra database using CQL and Cassandra API. As shown in Figure 22, CQL query statement is issued to retrieve the sensor id and the FENode from the sensor column family, which stores the metadata and position information of all the sensors. The WHERE statement specifies the query for the SingleAxisAccelerometers attached on the bridge. The query is transmitted to the Cassandra database and the (selected) query results are shown in Figure 22. The query results include the id of the accelerometers and their corresponding FENodes. This information will be used in step 2 and step 5.

In step 2, we retrieve acceleration data from the database. Figure 23 shows the query requesting the retrieval of sensor data collected from sensors whose IDs are in the list retrieved from step 1. The WHERE clause of the query specifies the time range from 2014-08-01 00:00:00 to 2014-08-01 02:00:00. As shown in Figure 23, the query results are presented in sorted order according to their timestamp. The query results are stored and will be used in step 6 where the sensor data and analysis results are compared.

| Query statement | | |
|---|---------|--------|
| <pre> SELECT id, FENode FROM sensor WHERE sensor_type = 'SingleAxisAccelerometer'; </pre> | | |
| Query result | | |
| u131ch0 | id | FENode |
| | u131ch0 | 718 |
| u184ch0 | id | FENode |
| | u184ch0 | 626 |
| ... more rows ... | | |

Figure 22. Step 1: Retrieving sensor information

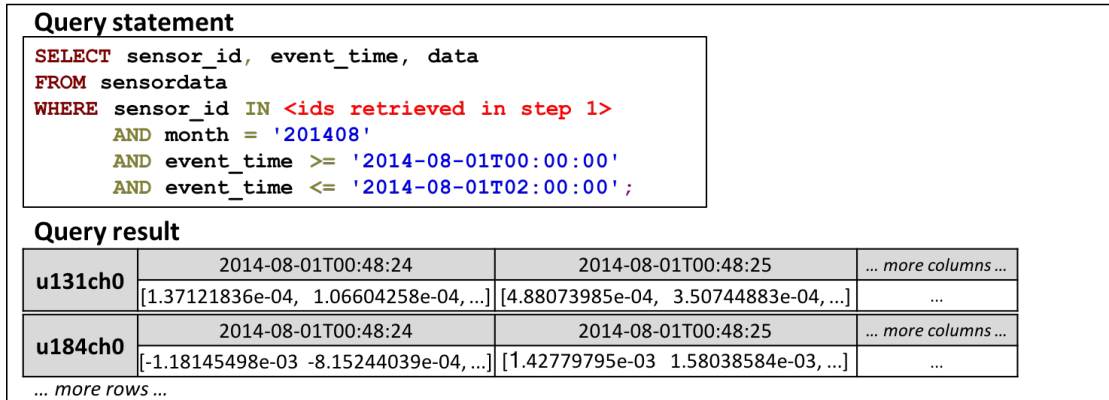


Figure 23. Step 2: Retrieving sensor data

In step 3, the FE model of the bridge is retrieved from the Cassandra database. Using the hierarchical relationship between objects using child and parent columns, we can automate the process to rebuild the XML-based BrIM model. Figure 24 shows the pseudo code for retrieving and rebuilding BrIM model using the recursive function `RetrieveDataByKey` (lines 1 to 10), which includes the query (see lines 2-4) to (recursively) retrieve the FE model information. If retrieved object contains `child` column, the function `RetrieveDataByKey` calls itself with input argument specifying the `uid` and column family of child objects (line 9). As shown in line 14, the recursive function starts from the root object of FE model whose `uid` is `448f641e-8e04-11e6-8f0d-3c15c2e54ea0` and column family is `Project`. The query results are received in the Python Dictionary data format and then converted into hierarchical XML object using `xml.etree.ElementTree` package (Python Software Foundation 2016a), as illustrated in Figure 25.

In step 4, the XML-based BrIM model created in step 3 is mapped to the Microsoft Excel spreadsheet file that can be processed by the CSI Bridge software. Figure 26 shows the pseudo code for the data mapping from BrIM to the Excel spreadsheet. The pseudo code has a recursive function to explore every object in hierarchical object-tree structure. Specifically, the recursive function parses attributes, parameters and child objects of a single object (lines 2-4), and then maps the parsed data entities onto an Excel spreadsheet (line 5). Specifically, we develop a mapping dictionary that matches BrIM object type with corresponding spreadsheet name (see Figure 27). The `xml.etree.ElementTree` package (Python Software Foundation 2016a) and `openpyxl` package (Gazoni and Clark 2016) are employed for parsing XML and Excel spreadsheet files.

Pseudo code

```

1 def RetrieveDataByKey(key, columnfamily):
2     Query = "SELECT *
3           FROM columnfamily
4           WHERE uid = key"
5     Object = RunQuery(Query)
6     if RetrievedObject["child"] is not None:
7         children = Object["child"]
8         for child in children:
9             Object.attach(RetrieveDataByKey(child, children[child])) # Recursive Function
10    return Object
11
12 def main():
13     # Start recursive function from the root object
14     RetrieveDataByKey("urn:uuid:448f641e-8e04-11e6-8f0d-3c15c2e54ea0", "Project")

```

Figure 24. Step 3: Pseudo code of a recursive function for FE model retrieval

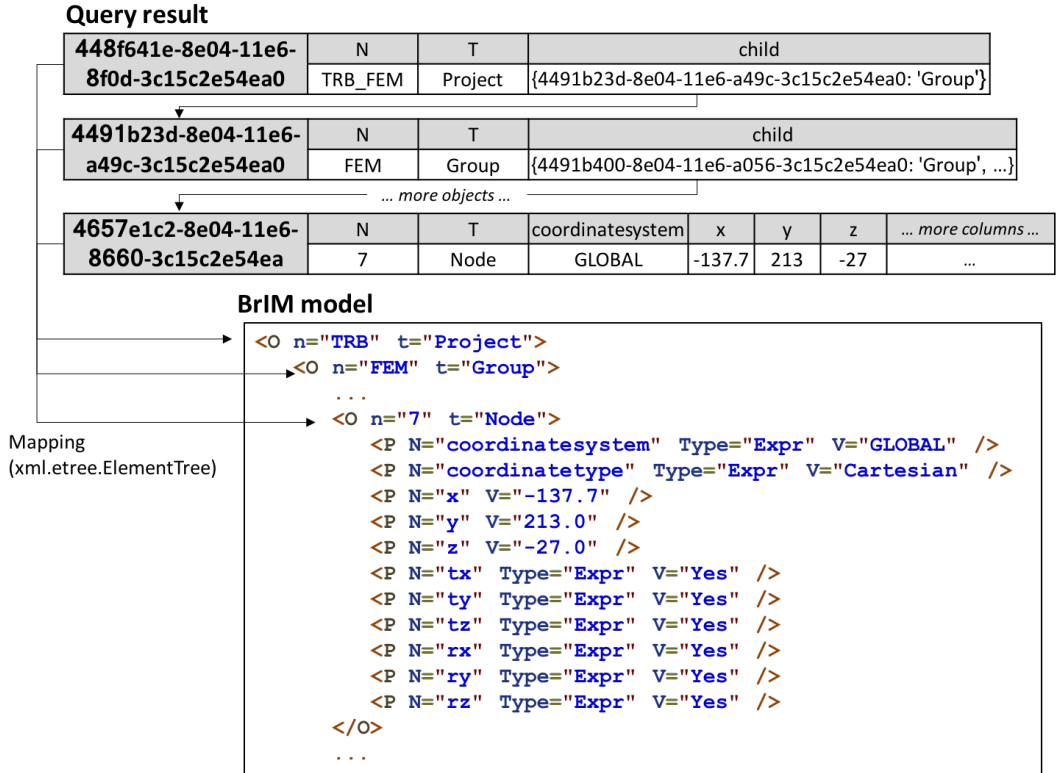


Figure 25. Step 3: Data mapping from query result to hierarchical BrIM model

Pseudo code

```

1 def ParseObject(object):
2     attribute = GetAttribute(object)
3     parameter = GetParameter(object)
4     childObject = GetChildObj(object)
5     ParseDataUsingMappingDictionary(attribute, parameter)
6     for child in childObject:
7         ParseObject(child)
8
9 def main():
10    # Start recursive function from the root object
11    ParseObject(getroot(inputXMLFile))

```

Figure 26. Step 4: Pseudo code for FE model mapping from BrIM to Excel spreadsheet

Mapping dictionary (BrIM – Excel)

```

{'Node': {'Joint Coordinates': {'coordinatesystem': 'CoordSys',
                                'N': 'Joint', 'x': 'XorR', 'y': 'Y', 'z': 'Z'},
          'Joint Restraint Assignments': {'tz': 'U3', 'tx': 'U1', 'ty': 'U2',
                                           'rx': 'R1', 'ry': 'R2', 'rz': 'R3', 'N': 'Joint'}},
'FELine': {'Connectivity - Frame': {'node1': 'JointI', 'node2': 'JointJ',
                                     'iscurved': 'IsCurved', 'N': 'Frame'}, ...},

```

BrIM object type

Excel spreadsheet name

BrIM attribute/parameter name

Excel spreadsheet column name

... more mapping dictionary ... }

Figure 27. Step 4: Mapping dictionary from BrIM to Excel spreadsheet

In step 5, the FE model created in step 4 is analyzed. To automate the analysis process, we develop two Visual Basic for Application (VBA) scripts using CSI Bridge’s APIs. As shown in Figure 28, Python script calls the VBA scripts using Python extensions for Windows (pywin32) package (Python Software Foundation 2016b). The first script accepts the list of FENodes as an input argument called “nodeList ()”.

The script then reads the FE model file created in step 4, runs the analysis, and records the response at the specified FENodes to spreadsheets. Here, for demonstration purpose, we set a moving truck load at the middle lane of the bridge and perform a time-history analysis. The results are then parsed with the second VBA script to retrieve the analysis results for the specified FENodes.

Python Script

```

xl = win32com.client.Dispatch("Excel.Application")
xl.Workbooks.Open(filepath, ReadOnly=1)

# Run 1st VBA script
xl.Application.Run("FEanalysis", filename, node)

# Run 2nd VBA script
result = xl.Application.Run("GetReturn")

```

VBA Script 1: FEanalysis

```

Public Sub Feanalysis (filename As String, nodeList() As Variant)
    Dim myCSIObject As COAPI 'Create CSI Bridge Instance
    ... Omitted ...

    ret = mySapModel.File.OpenFile(filename) 'Open FE model
    ret = mySapModel.Analyze.RunAnalysis 'Run analysis
    ... Omitted ...

    'Get response at specific node in node list
    For i = 0 To UBound(nodeList, 1)
        ret = mySapModel.Results.JointAcc(CStr(nodeList(i)), ... StepNum, ..., U3, ...)
        'Record result to active spreadsheet
        For j = LBound(StepNum) To UBound(StepNum)
            ThisWorkbook.ActiveSheet.Cells(j + 1, i + 1) = U3(j)
        Next j
    Next i
End Sub

```

VBA Script 2: GetReturn

```

Public Function GetReturn() As Variant
    Dim nRow, nCol As Integer
    nRow = Sheets("Sheet1").Cells(Rows.Count, 1).End(xlUp).Row
    nCol = Sheets("Sheet1").Cells(1, Columns.Count).End(xlToLeft).Column
    'Return computed response
    GetReturn = ThisWorkbook.ActiveSheet.Range(Cells(1, 1), Cells(nRow, nCol)).Value
End Function

```

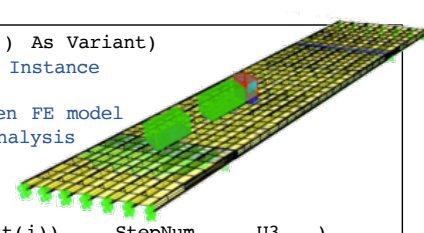


Figure 28. Step 5: Running FEA using CSI Bridge and its APIs

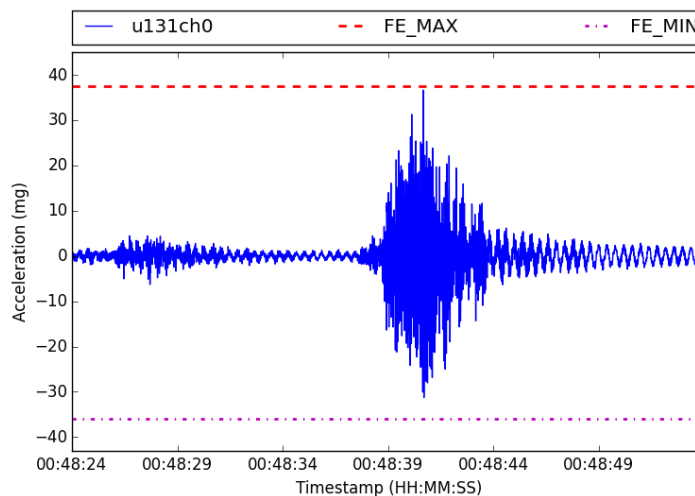


Figure 29. Step 6: Plotting retrieved sensor measurement (u131ch0) along with the maximum and minimum values of the response obtained from FE simulation

Finally, in step 6, the sensor data retrieved from step 2 and the analysis results obtained in step 5 are compared. In this example scenario, we calculate the minimum and maximum values of the computed response and plot them with the sensor measurements as shown in Figure 29. The sensor “u131ch0” is an accelerometer that measures vertical vibration at the leftmost girder of the bridge. The sensor measurements range from -31.28mg to 36.66mg, while the minimum and maximum values of computed response are -35.92mg and 37.48mg. The results show that the bridge structure behaves within the range of the analytically computed responses during the specified time period.

4.2 Example scenario 2: Retrieval of sensor data along with traffic-image data

In the second example scenario, we retrieve the sensor measurement data along with the traffic-image data. This example illustrates the retrieval of not only bridge response data collected by a sensor, but also the traffic information that causes the bridge response. Figure 30 shows the basic steps for implementing the data retrieval process.

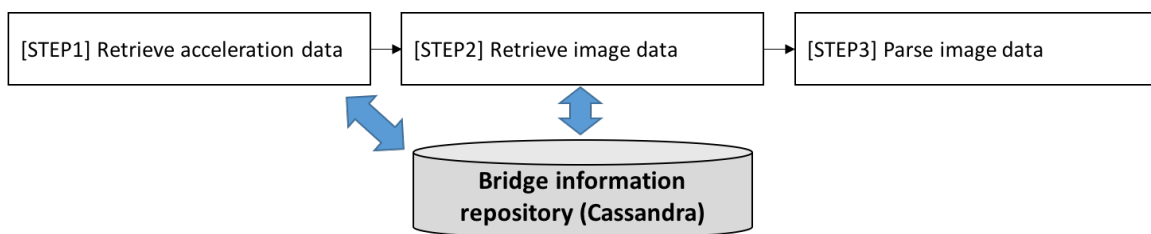


Figure 30. Workflow of scenario 2

In step 1, the sensor measurements are retrieved using CQL. As shown in Figure 31, the acceleration data is selected using the WHERE clause where the sensor ID is “u131ch0”, the month of the timestamp is “201608”, and the time period between 2016-08-23 10:02:09 and 2016-08-23 10:03:08.

Query statement for sensor data

```

SELECT event_time, data
FROM sensordata
WHERE sensor_id = 'u131ch0'
      AND month = '201608'
      AND event_time >= '2016-08-23T10:02:09'
      AND event_time <= '2016-08-23T10:03:08';
  
```

Figure 31. Step 1: Query statement for retrieving sensor data

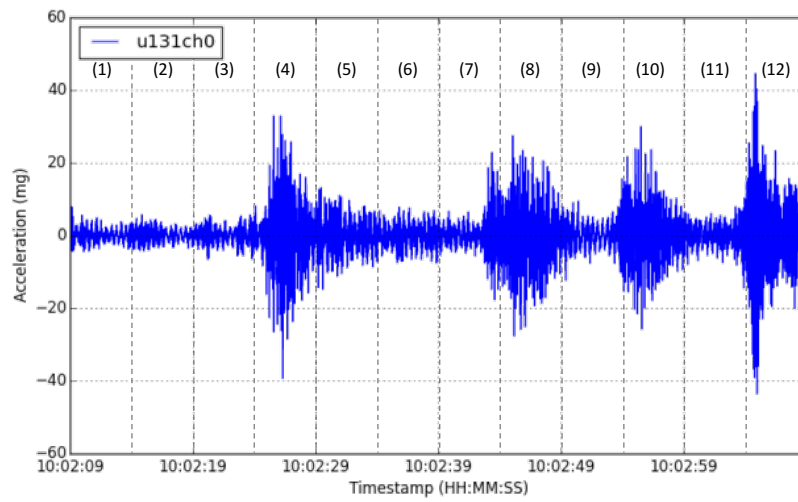
Query statement for image data

```

SELECT event_time, image
FROM imagedata
WHERE camera_id = 'telegraph2'
      AND date = '20160823'
      AND event_time >= '2016-08-23T10:01:54'
      AND event_time <= '2016-08-23T10:03:08';
  
```

Figure 32. Step 2: Query statement for retrieving image data

In step 2, traffic-monitoring images are retrieved to observe the vehicles that affect the bridge response retrieved in step 1. Figure 32 shows the query statement for retrieving the image data from the camera ID “telegraph2”, date “20160823” and the period between 2016-08-23 10:01:54 and 2016-08-23 10:03:08. The time period is extended slightly to capture the vehicles that went over the bridge before the sensor data period since the traffic flow may affect the initial vibration of the bridge. The images are retrieved as binary data stored in binary large object (BLOB) format and converted to an image file format, such as JPEG, in step 3.



(a) Retrieved acceleration data



(b) Retrieved traffic-monitoring images

Figure 33. Acceleration response of the TRB from 2016-08-23 10:02:09 to 2016-08-23 10:03:08 collected by “u131ch0” and corresponding traffic-monitoring images.

In step 3, the binary data is converted to image, for example, using Python's StringIO library (Python 2016c). Figure 33 shows the retrieved images with the corresponding sensor data. The images shown in Figure 33(b) are trimmed to show only the northbound lane that we are interested in. In this figure, the sensor data is divided into twelve segments as labeled from 1 to 12 for matching with the corresponding images. The image 0 shows the vehicle that crosses the bridge 11 seconds before the data acquisition began. The initial acceleration (segments 1 and 2 in Figure 33) ranges from -4.48mg to 7.96mg due to the vehicles captured in image 0. As shown in images 2, 3, 5 and 6 and the corresponding sensor data, the compact cars and midsize cars increase acceleration only up to 14.82mg, even with cars crossing the bridge at the same time. On the other hand, as shown in images 4, 8, 10 and 12 and the corresponding segments of sensor data, trucks and trailers increase the acceleration level significantly and up to 44.57mg.

5. SUMMARY AND CONCLUSION

Bridge monitoring and management involves a wide variety of information from different data sources, including geometric modeling and engineering analysis tools, bridge management system (BMS) and structural health monitoring (SHM) system. While the different types of information are related to each other, current practice of bridge management typically handles them using isolated systems, followed by manual data conversion. In order to address the data sharing issue, we discuss a bridge information modeling framework for supporting bridge monitoring and management applications. We use the OpenBrIM data model (OpenBrIM 2016) as the base model, and augment and extend the base model to capture engineering model and sensor description, by examining relevant software tools and data modeling standards. Specifically, we investigate the data entities of CSI Bridge (Computers & Structures, Inc. 2016), a structural analysis software tool, and SensorML (Open Geospatial Consortium 2014), a standard for sensor description.

To facilitate storage, sharing and utilization of bridge information model, a NoSQL database system is employed. Since bridge monitoring and management applications will potentially involve a large volume of data with various data types, an appropriate database system that can guarantee scalability and flexibility is important. In this study, we employ Apache Cassandra, one of the most commonly used column family NoSQL databases because of its high scalability and flexible data schema. Data schema for Cassandra database is defined following the bridge and sensor information models.

The data management framework is demonstrated using the bridge information and sensor data collected from the Telegraph Road Bridge (TRB) located in Monroe, Michigan. The bridge information model was built based on 2-dimensional drawings, 3-dimensional engineering model and sensor description of the TRB. The created bridge information model is stored to the Cassandra database using data mapping scripts that convert the bridge information model expressed in XML-based tree structure to the column-oriented data model of Cassandra database. In addition, sensor data and traffic monitoring images are stored in the Cassandra database. We have presented demonstrative example scenarios that involve different types of bridge information, which are typically managed by isolated systems in the current practice of bridge monitoring and management. The first example scenario describes the integrated data retrieval for sensor information, sensor data and FE model. The second example describes the retrieval of sensor data with the corresponding image data. The demonstration results show that the bridge information modeling framework is able to integrate different types of bridge information by linking related data entities.

As a research prototype, the bridge information model, in its current state, considers only a few standards and applications. Many data entities, which are necessary to fully support other bridge monitoring and management applications, are lacking. However, the bridge information model can be easily extended by defining new object type definitions and updating Cassandra database schema as new data entities are required by new applications. The flexible data structure of Cassandra database system is able to efficiently handle the dynamic schema modification. Another shortcoming is the limited query possibilities of Cassandra database, which requires creating data retrieval scripts. For prototyping purpose, performing complex operations, as illustrated in the scenario example, requires combining and manipulating data from different data sources. Our current and future study includes developing standardized interfaces and composition tools. Furthermore, our current work attempts to implement standardized web service for commonly used data

queries in bridge monitoring and management applications to establish a collection of reusable application services.

ACKNOWLEDGEMENT

This research is supported by a Grant No. 13SCIPA01 from Smart Civil Infrastructure Research Program funded by Ministry of Land, Infrastructure and Transport (MOLIT) of Korea government and Korea Agency for Infrastructure Technology Advancement (KAIA). The research is also partially supported by a collaborative project funded by the US National Science Foundation (Grant No. ECCS-1446330 to Stanford University and Grant No. CMMI-1362513 and ECCS-1446521 to the University of Michigan). The authors thank the Michigan Department of Transportation (MDOT) for access to the Telegraph Road Bridge and for offering support during installation of the wireless monitoring system. The in-kind support by Computers and Structures, Inc. for providing the CSI Bridge software to the research team at Stanford University is gratefully appreciated. Any opinions, findings, conclusions or recommendations expressed in this paper are solely those of the authors and do not necessarily reflect the views of NSF, MOLIT, MDOT, KAIA or any other organizations and collaborators.

REFERENCE

- Bartholomew, M. (2015). "FHWA Bridge Information Modeling Update," [Online article], Retrieved from: <http://bridges.transportation.org/Documents/2015%20SCOBS%20presentations/Technical%20Committee/T-19-Mike%20Bartholomew-FHWA%20Bridge%20Information%20Modeling.pdf> (accessed on 20 April 2016)
- Bartholomew, M., Blasen, B. and Koc, A. (2015). "Bridge Information Modeling (BrIM) Using Open Parametric Objects," Report No. FHWA-HI F -16-010, Federal Highway Administration.
- buildingSMART. "Industry Foundation Classes Version 4 - Addendum 1," [Online article], Retrieved from: <http://www.buildingsmart-tech.org/ifc/IFC4/Add1/html/> (accessed on 20 May 2016)
- Chen S. S. (2013). "Bridge Data Protocols for Interoperability Local Failure Bridge Data Protocols for Interoperability and Life Cycle Management," [Online article], Retrieved from: <http://iug.buildingsmart.org/resources/itm-and-iug-meetings-2013-munich/infra-room/bridge-data-protocols-for-interoperability-and-life-cycle-management> (accessed on 20 April 2016)
- Chen, S. S. and Shirolé, A. M. (2006). "Integration of information and automation technologies in bridge engineering and management: Extending the state of the art," *Transportation Research Record*, 1976(1), pp. 3-12.
- Chen, S. S. and Shirolé, A. M. (2013). "Implementation Roadmap for Bridge Information Modeling (BrIM) Data Exchange Protocols," SubTask 12.2 Multi-Year Implementation Roadmap Revised Draft documenting updates to July 2013.
- Computers & Structures, Inc. "Structural Bridge Design Software | CSiBridge," [Online article], Retrieved from: <https://www.csiamerica.com/products/csibridge> (accessed on 20 April 2016)
- DataStax. "Introduction to Cassandra Query Language," [Online article], Retrieved from: http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html (accessed on 20 April 2016a)
- DataStax. "Getting Started with Time Series Data Modeling," [Online article], Retrieved from: <https://academy.datastax.com/demos/getting-started-time-series-data-modeling> (accessed on 20 April 2016b)
- DataStax. "Python Cassandra Driver," [Online article], Retrieved from: <https://datastax.github.io/python-driver/> (accessed on 20 April 2016c)
- Eastman, C. M. (1999). "Building product models: computer environments, supporting design and construction," CRC press.
- Eastman, C., Teicholz, P., Sacks, R. and Liston, K. (2011). "BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors," John Wiley & Sons, Inc., Hoboken, NJ.

- Gazoni E. and Clark C. (2016). "openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files," [Online article], Retrieved from: <http://openpyxl.readthedocs.org/en/default/> (accessed on 20 April 2016)
- Grolinger, K., Higashino, W. A., Tiwari, A. and Capretz, M. A. (2013). "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications*, 2(1), pp. 22–41.
- Hecht, R. and Jablonski, S. (2011). "NoSQL Evaluation: A use case oriented survey," *Proceedings of CSC 2011*, pp. 336-341.
- Hewitt, E. (2010) "Cassandra: the definitive guide," O'Reilly Media, Inc., [Safari Book Online], Retrieved from: <http://proquest.safaribooksonline.com/9781449399764> (accessed on 20 April 2016)
- Hou, R., Zhang, Y., O'Connor, S., Hong, Y. and Lynch, J. P. (2015). "Monitoring and Identification of Vehicle-Bridge Interaction using Mobile Truck-based Wireless Sensors," *Proceedings of 11th International Workshop on Advanced Smart Materials and Smart Structures Technology*, August 1-2, 2015, University of Illinois, Urbana-Champaign, United States.
- ISO. (1994). "ISO 10303:1994 – Industrial Automation Systems and Integration – Product Data Representation and Exchange," International Organization for Standardization (ISO), Geneva, Switzerland.
- ISO. (2000). "ISO 10303-104:2000 – Industrial Automation Systems and Integration – Product Data Representation and Exchange -- Part 104: Integrated application resource: Finite element analysis," International Organization for Standardization (ISO), Geneva, Switzerland.
- Jeong, S., Zhang, Y., O'Connor, S. M., Lynch, J. P., Sohn, H. and Law, K. H. (2016). "A NoSQL Data Management Infrastructure for Bridge Monitoring," *Smart Structures and Systems*, 17(4), pp. 669-690.
- Lakshman, A. and Malik, P. (2010). "Cassandra: a decentralized structured storage system," *Operating Systems Review (ACM)*, 44(2), pp. 35-40.
- Lee, K. (2000). "IEEE 1451: A standard in support of smart transducer networking," *Proceedings of IEEE Instrumentation and Measurement Technology Conference*, 2, pp. 525-528.
- Lee, S.-H. and Jeong, Y.-S. (2006). "A system integration framework through development of ISO 10303-based product model for steel bridges," *Automation in Construction*, 15(2), pp. 212–228.
- Li, H., Ou, J., Zhao, X., Zhou, W., Li, H., Zhou, Z. and Yang, Y. (2006). "Structural health monitoring system for the Shandong Binzhou Yellow River Highway Bridge," *Computer-Aided Civil and Infrastructure Engineering*, 21(4), pp. 306-317.
- Li, Y. and Manoharan, S. (2013). "A performance comparison of SQL and NoSQL databases," *Proceedings of IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing*, art. no. 6625441, pp. 15-19.
- Marzouk, M. M. and Hisham, M. (2011). "Bridge information modeling in sustainable bridge management," *Proceedings of the International Conference on Sustainable Design and Construction 2011: Integrating Sustainability Practices in the Construction Industry*, pp. 457-466.
- Nagel, R. N., Braithwaite, W. W. and Kennicott, P. R. (1980). "Initial Graphics Exchange Specification IGES Version 1.0," Washington DC: National Bureau of Standards, NBSIR 80-1978.
- Open Geospatial Consortium (2014). "OGC® SensorML: Model and XML Encoding Standard" [Online article], Retrieved from: <http://www.opengeospatial.org/standards/sensorml> (accessed on 20 April 2016)
- OpenBrIM, "OpenBrIM V3," [Online resource], Retrieved from: <https://openbrim.appspot.com/schema.xsd?for=openbrim&v=3&format=xsd&version=1.1> (accessed on 10 November 2016)
- ParamML, "ParamML Author's Guide," [Online article], Retrieved from: <https://sites.google.com/a/redeqn.com/paramml-author-s-guide/system/app/pages/recentChanges> (accessed on 20 April 2016)
- Pschorr, J., Henson, C. A., Patni, H. K. and Sheth, A. P. (2010). "Sensor Discovery on Linked Data," Retrieved from: <http://corescholar.libraries.wright.edu/knoesis/780> (accessed 2 March, 2016).
- Python Software Foundation, "19.7. xml.etree.ElementTree - The ElementTree XML API," [Online article], Retrieved from: <https://docs.python.org/2/library/xml.etree.elementtree.html> (accessed on 20 April 2016a)
- Python Software Foundation, "pywin32 214 : Python Package Index," [Online article], Retrieved from: <https://pypi.python.org/pypi/pywin32> (accessed on 7 October 2016b)
- Python Software Foundation, "7.5. StringIO — Read and write strings as files," [Online article], Retrieved from: <https://docs.python.org/2/library/stringio.html> (accessed on 7 October 2016c)

- Robert, W. E., Marshall, A. R., Shepard, R., and Aldayuz, J. (2003). "The Pontis Bridge management system: State-of-the-practice in implementation and development," *Proceedings of the 9th International Bridge Management Conference*, pp. 49-60.
- Samec, V., Stamper, J., Sorsky, H. and Gilmore, T. W. (2014). "Long span suspension bridges – bridge information modeling," *Proceedings of 7th International Conference of Bridge Maintenance, Safety and Management*, pp. 1005-1010.
- Stonebraker, M., Madden, S., Abadi, D. J., Harizopoulos, S., Hachem, N. and Helland, P. (2007). "The end of an architectural era (it's time for a complete rewrite)," *Proceedings of the 33rd International Conference on Very large data bases*, pp. 1150–1160.
- Wang, L., Tang, A., Cui, Y., Yang, S. and Zhan, Z. (2009). "Study on the Development of Sihui Bridge Management System," *Proceedings of 2009 WRI World Congress on Software Engineering, IEEE*. pp. 487–491.
- Yabuki, N., Lebegue, E., Gual, J. and Shitani, T. (2006). "International collaboration for developing the bridge product model IFC-Bridge," *Proceeding of the 11th International Conference on Computing in Civil and Building Engineering*, pp. 1927-1936.
- Zhang, Y., O'Connor, S. M., van der Linden, G., Prakash, A. and Lynch, J. P. (2016). "SenStore: A Scalable Cyberinfrastructure Platform for Implementation of Data-to-Decision Frameworks for Infrastructure Health Management," *Journal of Computing in Civil Engineering*, 04016012
- Zhou, H. F., Chan, T. K., Wang, J. Y. and Ni, Y. Q. (2006). "A structural health monitoring data management system for instrumented cable-supported bridges," *Proceedings of the Asia-Pacific Workshop on Structural Health Monitoring*, pp. 514-522.