A SCALABLE AND INTEROPERABLE CYBERINFRASTRUCTURE PLATFORM FOR CIVIL INFRASTRUCTURE MONITORING

A DISSERTATION

SUBMITTED TO THE DEPARMENT OF

CIVIL AND ENVIRONMENTAL ENGINEERING

AND THE COMMITTEE ON GRATDUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Seongwoon Jeong

March 2019

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Kincho Law, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Anne Kiremidjian

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Jerome Lynch

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

Abstract

Civil infrastructure monitoring is an important technology that provides accurate and objective data on the health condition of a structure by leveraging sensor technologies. Together with routine maintenance and inspection, civil infrastructure monitoring enables the diagnosis of potential structural problems and the prognosis for the need of structural strengthening and repairs. As sensor technologies mature and become economically affordable, their deployment for civil infrastructure monitoring will continue to grow to collect more detailed data about the structures. The data collected from civil infrastructure monitoring systems offers promising opportunities to find meaningful information, knowledge and insight that can improve decision making processes. Furthermore, advances in sensing and communication technologies will eventually realize the concept of cyber-physical systems (CPS) that tightly integrate physical systems and cyber systems to monitor, analyze, coordinate and control the operations of physical systems. Nevertheless, the increasing use of sensors will also lead to significant data management issues. Civil infrastructure monitoring systems instrumented with dense sensor networks will be inundated with unprecedented volume and diverse types of data that need to be processed, interpreted and brought forth to support system operations. The utilization of such "big data" will be significantly limited unless a proper data management platform, which can efficiently store, manage, retrieve, share, interface, link and integrate data, is developed.

This thesis describes a cyberinfrastructure platform for civil infrastructure monitoring with an emphasis on system scalability and interoperability. The cyberinfrastructure platform brings together information and communication technologies (ICT), including information modeling, NoSQL database, cloud computing and web services, for effective data management. An information modeling framework with application to bridge monitoring is designed to facilitate data interoperability and data integration. A NoSQL-based data management system is developed to enable scalable, flexible and fault-tolerant management of monitoring data. Cloud computing is adopted as a scalable, reliable and accessible computing infrastructure. Platform-neutral web services are developed to enable easy access to the cloud resources and data involved in engineering systems via standard communication protocols. For demonstration, the cyberinfrastructure platform is implemented for the monitoring of bridges along the I-275 corridor in the State of Michigan. The results show that the cyberinfrastructure platform can effectively manage the sensor data and domain-specific information and facilitate data sharing, integration and utilization.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor and life mentor, Professor Kincho H. Law, who always encourages me to move outside of my comfort zone and provides me deep insights on this interdisciplinary research. His patient guidance has helped me find a field of intellectual pursuit that fits my interest and ability. I am forever indebted to him.

Special thanks are owed to Professor Hoon Sohn for allowing me the opportunity to conduct collaborative research, as well as to providing me access to his laboratory and research facilities. Many thanks are owed to Professor Jerome P. Lynch for sharing with me his insights on structural health monitoring, as well as for providing valuable data from his testbeds to validate my research. I would also like to thank Professor Hector Garcia-Molina for chairing my defense. I am also grateful to Professor Anne Kiremidjian and Professor Michael Lepech for serving as my defense committee. Their comments and valuable questions have helped me improve the contents of my dissertation.

I would like to express my sincere thanks to the members of the Engineering Informatics Group, particularly Dr. Zan Chu, Professor Jinkyoo Park and Max Ferguson, for helping me in various aspects of my research. I would like to thank my colleagues at University of Michigan, particularly, Dr. Yilan Zhang and Rui Hou. They are responsible for providing access to the bridges on I-275 Corridor and helping me use various data collected from the bridges to validate my research. I would also like to thank colleagues at KAIST, particularly, Dr. Hyung Jin Lim, Dr. Ji-Min Kim and Suyoung Yang for helping me use their laboratory and conduct field tests on their testbeds, including the Youngjong Bridge and the Yeondae Bridge.

I would like to extend my thanks to many friends for their continuous encouragement and support throughout this work. My special thanks go to Justin Smith, Wendy Smith, Grant Wells, Christine Wells, Jared Honeycutt, Wendy Quay, Tom Curtis and Christina Curtis. I would also like to thank Jung In Kim, Jinhyun Choo, Changhyun Noh, Hyewon Shin and Yongchae Cho.

Finally, I am grateful for the constant support from my family. I would like to especially acknowledge my parents and brother for their unwavering love and support. I would also like to thank my father-in-law for his unconditional trust and encouragement. Last but not least, I would like to express my deepest gratitude to my beloved wife, Junghwa Han, who is my best friend and lifetime companion. My venture to study abroad at Stanford would not have been conceivable without her selfless support and continuous encouragement.

I gratefully acknowledge the financial support provided by a Grant Number 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). I also acknowledge the financial support provided by the US National Science Foundation (NSF), Grant Number ECCS-1446330. Any opinions, findings, conclusions or recommendations expressed in this paper are solely those of the authors and do not necessarily reflect the views of MOLIT, KAIA, NSF or any other organizations and collaborators.

Table of Contents

Abstract iv
Acknowledgements vi
Table of Contents viii
List of Tables xiii
List of Figures xiv
1 Introduction 1
1.1 Problem Statement 1
1.2 Data Management Requirements for Cyberinfrastructure Platform 3
1.2.1 Scalability 4
1.2.2 Data integration and interoperability 4
1.2.3 Standardized interface 5
1.2.4 Flexibility 5
1.3 Current Sensor Data Management Practice 6
1.3.1 Internet of Things (IoT) platform 6
1.3.2 Civil infrastructure monitoring systems 7
1.4 Information and Communication Technologies for Effective Data Management 9
1.4.1 Information modeling 9
1.4.2 Cloud computing 9
1.4.3 NoSQL database 10
1.4.4 Web service 11

1.5 Research Objectives 11
1.6 Thesis Outline 13
2 Information Modeling Framework for Bridge Monitoring 15
2.1 Introduction 15
2.2 Bridge Information Modeling Schema Definition 17
2.2.1 Base model: OpenBrIM 17
2.2.2 Finite element modeling 21
2.2.2.1 Data entities for representing finite element model 22
2.2.2.2 Data entities for representing load and analysis conditions 26
2.2.3 Sensor description 28
2.3 Case Example 34
2.3.1 Geometric model 35
2.3.2 Finite element model 36
2.3.3 Sensor description 40
2.4 Summary 42
3 A NoSQL-based Scalable Data Management Framework for Civil Infrastructure Monitoring 44
3.1 Introduction 44
3.2 Selection of NoSQL Data Management Tools 46
3.2.1 Data management requirements in civil infrastructure monitoring system 46
3.2.2 Selection of NoSQL database tools 49
3.2.2.1 Apache Cassandra: column family database for supporting persistent data archiving 50
3.2.2.2 MongoDB: document store for supporting efficient data retrieval 52
3.3 A NoSQL-based Data Management Framework for Civil Infrastructure Monitoring 53
3.3.1 Onsite computer 55
3.3.1.1 Data scheme for document database on onsite computer 55
3.3.1.2 Data store and retrieval processes on onsite computer 57
3.3.2 Main server 58

3.3.2.1 Data scheme for column family database on main server 59
3.3.2.1.1 Sensor data schema 59
3.3.2.1.2 Bridge information model schema 62
3.3.2.2 Data store and retrieval processes on main server 64
3.3.2.2.1 Data store and retrieval processes for sensor data 64
3.3.2.2.2 Data store and retrieval processes for bridge information model 65
3.3.3 Local computer 67
3.4 Case Example 68
3.4.1 Data store and retrieval 69
3.4.2 Influence line analysis using sensor data and bridge engineering model 72
3.4.3 Comparison of sensor data and analytically computed bridge response........75
3.4.4 Retrieval of sensor data along with traffic image data...................................81
3.5 Summary 83
4 A Cloud-based Cyberinfrastructure Platform for Civil Infrastructure Monitoring 86
4.1 Introduction 86
4.2 Overview of Cyberinfrastructure Platform 88
4.3 Data Store Process 90
4.3.1 Communication layer 90
4.3.1.1 Web server 90
4.3.1.2 Message broker 93
4.3.2 Mapping layer 94
4.3.3 Storage layer 94
4.4 Data Retrieval Process 95
4.4.1 Data retrieval using web services 96
4.4.2 Service composition 98
4.5 Cloud-based Implementation 103
4.5.1 Cloud computing environment 104
4.5.2 Hybrid cloud-based decentralized data management 107
4.6 Case Scenario: Civil Infrastructure Monitoring Application 108

4.6.1 Automated data store and retrieval 109
4.6.2 Data integration and utilization 112
4.6.3 Web and mobile user interfaces 115
4.7 Summary 118
5 Implementation of Data-driven Sensor Data Reconstruction Procedure utilizing the Cyberinfrastructure Platform 121
5.1 Introduction 121
5.2 BRNN-based Sensor Data Reconstruction Method 124
5.2.1 Data preprocessing and normalization 125
5.2.2 Architecture of the BRNN model 126
5.2.3 Training of BRNN model 129
5.2.4 Sensor data reconstruction 131
5.3 Demonstration of Sensor Data Reconstruction Method with Numerical Simulation 133
5.3.1 Effects of training parameters 136
5.3.2 Comparison of BRNN-based sensor data reconstruction with other methods 139
5.3.3 Sensor validation 142
5.4 Data Analysis Pipeline based on Cyberinfrastructure Platform 145
5.4.1 Training of BRNN model on high-performance computer 146
5.4.2 Storing the trained model on cyberinfrastructure platform 150
5.4.3 Sensor data reconstruction on local computer 151
5.5 Application on a Bridge in Service 154
5.5.1 Comparison of BRNN-based sensor data reconstruction with other methods 155
5.5.2 Effects of using different set of input sensors 156
5.5.3 Environmental effect 159
5.5.4 Reconstruction of missing sensor data 160
5.6 Summary 162
6 Conclusions 165

6.1 Summary of Results and Contributions 165
6.2 Future Research Recommendation 168
Bibliography 170

List of Tables

Table 2.1 Objects added to the OpenBrIM model for representing structural elements ...26

Table 2.2 Objects added to the base OpenBrIM model for representing load and analysis conditions 29

Table 2.3 Data entities included in the Sensor subtype objects. 34

Table 3.1 Test truck load description (unit: pound) [112] 73

Table 4.1 RESTful data store web services currently implemented on the cyberinfrastructure platform 91

Table 4.2 RESTful data retrieval web services currently implemented on the cyberinfrastructure platform 96

Table 4.3 Specification and role of cloud virtual machines and private server composing cyberinfrastructure platform 109

Table 5.1 Random variables composing randomized traffic 134

Table 5.2 Data-driven sensor data reconstruction models for comparison 140

Table 5.3 Simulation: computing time and reconstruction error 141

Table 5.4 Bridge structure: testing error 155

Table 5.5 Testing error for different input combinations 158

List of Figures

Figure 2.1 Definition of a fundamental Object and Parameter entities [78] 18
Figure 2.2 Data schema of the Shape object [78]...20
Figure 2.3 Entity: Node...23
Figure 2.4 Entity: FELine 23
Figure 2.5 Entity: FESurface 24
Figure 2.6 Entity: Material..25
Figure 2.7 New entity: FELoadPattern 27
Figure 2.8 New entity: FEAnalysisCase 28
Figure 2.9 Importing SensorML's schema 30
Figure 2.10 New entity: Sensor 32
Figure 2.11 New entities: Sensor subtype object 33
Figure 2.12 Telegraph Road Bridge (Monroe, Michigan)..35
Figure 2.13 High-level structure of geometric model written in ParamML.......................36
Figure 2.14 Geometric model visualized using OpenBrIM Viewer 37
Figure 2.15 TRB's FE model created using CSiBridge (1st natural frequency: 2.324 Hz) 37
Figure 2.16 Data mapping from Excel Spreadsheet to OpenBrIM object..........................38
Figure 2.17 Pseudo code for data mapping from CSiBridge' data schema to BrIM data schema 38
Figure 2.18 High-level structure of the finite element model represented in BrIM data schema 39

Figure 2.19 Pseudo code for data mapping from BrIM data schema to CSiBridge' data schema 40
Figure 2.20 Modal analysis with the re-generated finite element model (1st natural frequency: 2.324 Hz) 40
Figure 2.21 Sensor description represented using BrIM data schema and SensorML data schema 41
Figure 3.1 Overall structure of data management system for civil infrastructure monitoring 47
Figure 3.2 Four categories of NoSQL database systems 49
Figure 3.3 Ring topology of Cassandra database 51
Figure 3.4 Data structure of Cassandra database..52
Figure 3.5 Data structure of MongoDB 53
Figure 3.6 Data management framework for civil infrastructure monitoring based on NoSQL database 54
Figure 3.7 Data schema of sensor data in MongoDB 56
Figure 3.8 Data schema of sensor information in MongoDB..57
Figure 3.9 Time-series data stored in a distributed system..59
Figure 3.10 Database schema for time-series numeric sensor data....................................60
Figure 3.11 Database schema for time-series image data..61
Figure 3.12 Data mapping between BrIM schema FELine and corresponding Cassandra column family 63
Figure 3.13 Database schema for BrIM objects 64
Figure 3.14 Data mapping from BrIM schema to Cassandra database schema 66
Figure 3.15 Data mapping from Cassandra database schema to BrIM schema 67
Figure 3.16 Type and location of sensors installed on the Telegraph Road Bridge [110] 69
Figure 3.17 onsite.py: Python script storing sensor data to MongoDB in onsite computer..70
Figure 3.18 tomain.py: Python script storing sensor data to Cassandra database in main server 70

Figure 3.19 Select query for sensor data retrieval and query result in a tabular format 71
Figure 3.20 Bridge information model retrieved from Cassandra database in main server 72
Figure 3.21 Sensor information retrieved from Cassandra database in main server 72
Figure 3.22 Test truck dimension 73
Figure 3.23 Location of strain gauges 73
Figure 3.24 Test truck defined in FE model 74
Figure 3.25 Influence line analysis result 75
Figure 3.26 Workflow of example scenario for comparing sensor data and analytically computed bridge reponse 76
Figure 3.27 Step 1: Retrieving sensor information 77
Figure 3.28 Step 2: Retrieving sensor data 77
Figure 3.29 Step 3: Pseudo code of a recursive function for FE model retrieval 78
Figure 3.30 Step 3: Data mapping from query result to hierarchical BrIM model 78
Figure 3.31 Step 4: Pseudo code for FE model mapping from BrIM to Excel spreadsheet 79
Figure 3.32 Step 4: Mapping dictionary from BrIM to Excel spreadsheet 79
Figure 3.33 Step 5: Running FE analysis using CSiBridge and its APIs 80
Figure 3.34 Step 6: Plotting retrieved sensor measurement (u131ch0) along with the maximum and minimum values of the response obtained from FE simulation 81
Figure 3.35 Workflow of example scenario for retrieving sensor data along with traffic-image data 81
Figure 3.36 Step 1: Query statement for retrieving sensor data 82
Figure 3.37 Step 2: Query statement for retrieving image data 82
Figure 3.38 Acceleration response of the TRB from 2016-08-23T10:02:09 to 2016-08-23T10:03:08 collected by “u131ch0” and corresponding traffic-monitoring images. 84
Figure 4.1 Overall architecture of the cyberinfrastructure platform for engineering 88

Figure 4.2 Web server hosting web services for engineering model and sensor data store 92
Figure 4.3 Message broker including a topic for sensor data exchange 93
Figure 4.4 Engineering model retrieval from the cyberinfrastructure platform 97
Figure 4.5 Sensor data retrieval from the cyberinfrastructure platform 97
Figure 4.6 A composite application DataRetrievalByLocation: data flow 99
Figure 4.7 A composite application DataRetrievalByLocation: Execution example 101
Figure 4.8 A composite application SensorInfoOnMap: Data flow 102
Figure 4.9 A composite application SensorInfoOnMap: Execution example 103
Figure 4.10 A model of cloud computing for SHM 105
Figure 4.11 Virtual machine created on Microsoft Azure cloud platform 106
Figure 4.12 A framework of hybrid cloud-based implementation of cyberinfrastructure platform for decentralized data management 108
Figure 4.13 Workflow: application for data store automation 110
Figure 4.14 Sensor data file 110
Figure 4.15 Workflow: traffic video image collecting application 111
Figure 4.16 Data retrieval using web services 112
Figure 4.17 A workflow for relating structural behavior with temperature data 113
Figure 4.18 HTTP requests and corresponding CQL queries for sensor list and data retrieval 114
Figure 4.19 Patterns of modal frequency of the Telegraph Road Bridge 116
Figure 4.20 Prototype web-based user interface 117
Figure 4.21 Telegraph Road Bridge model downloaded from the web-based user interface 117
Figure 4.22 Prototype mobile interface 118
Figure 5.1 Overview of sensor data reconstruction 124
Figure 5.2 Structure of bidirectional recurrent neural network with a single hidden layer and the computation flow of a forward pass 127
Figure 5.3 Organization of BRNN layers at a time step t 127

Figure 5.4 Expanded view of interconnection between time steps t and $t+1$ in the BRNN structure 128
Figure 5.5 (a) Finite element model and (b) sensor layout of the Telegraph Road Bridge 134
Figure 5.6 Vehicle load definition....................134
Figure 5.7 Simulated vertical acceleration measurements of the Telegraph Road Bridge (Monroe, MI) 135
Figure 5.8 Reconstruction error, training time and reconstruction time with respect to different batch sizes 137
Figure 5.9 Reconstruction error, training time and reconstruction time with respect to different training data sizes 138
Figure 5.10 Reconstruction error and testing time with respect to different testing data sizes 139
Figure 5.11 Simulation: sensor data reconstruction with the different methods....................141
Figure 5.12 Simulation: anomaly detection and identification for the anomaly due to noise at Sensor 4 and 6....................144
Figure 5.13 Overview of data analysis process for sensor data reconstruction....................146
Figure 5.14 Sensor data (JSON format) retrieved from the cyberinfrastructure platform by invoking sensor data retrieval service....................148
Figure 5.15 Trained BRNN model and metadata encoded in JSON format 149
Figure 5.16 Data schema for storing sensor data reconstruction model 151
Figure 5.17 Data mapping from the retrieved BRNN parameters to a BRNN model object created using PyTorch [153]....................152
Figure 5.18 Vertical acceleration measurement of the Telegraph Road Bridge (July 2014)....................154
Figure 5.19 Bridge structure: sensor data reconstruction with the different methods 156
Figure 5.20 Covariance matrix of sensor data collected from the Telegraph Road Bridge 157
Figure 5.21 Sensor data reconstruction with different inputs....................158

Figure 5.22 Temperature change measured on the Telegraph Road Bridge from October 2014 to May 2015 (Months selected for analysis are highlighted with boxes) 159
Figure 5.23 Reconstruction errors for testing sets collected from different months: 10 BRNN models trained with monthly data and one BRNN model trained with yearly data are used 160
Figure 5.24 Reconstruction of missing data using BRNN models 161
Figure 5.25 First five modal frequencies and operational mode shapes of the Telegraph Road Bridge 162

Chapter 1

Introduction

1.1 Problem Statement

Advances in sensor and communication network technologies have led to an increasing deployment of sensors for civil infrastructure monitoring [1, 2]. Data collected from sensor networks offers promising opportunities to enhance the operation of civil infrastructure systems. Research efforts for sensor data utilization, ranging from short-term anomaly detection to long-term trend investigation, have been widely reported [3, 4, 5]. Furthermore, advanced data analytics and machine learning methodologies can potentially enable finding hidden patterns, meaningful information and insights regarding the target systems and their operations from the sensor data. Growth in sensor deployment, however, will also give rise to data management issues in civil infrastructure monitoring. The amount and complexity of data involved in civil infrastructure monitoring make data management a very difficult task, which would hinder the utilization of sensor data. Data issues are of significant importance that need to be handled before sensing technologies can truly be useful for civil infrastructure management [6].

Successful data management can be achieved by using appropriate computing technologies with consideration of domain-specific characteristics. Until now, however, there is a huge gap between the computing and engineering communities. In the computing community, on the one hand, various technologies that can be used to cope with today's data issues have emerged [7]. For example, cloud computing has become a new computing paradigm where scalable, reliable, cost-effective and easy-to-manage computing resources can be rapidly provisioned and accessed over advanced communication networks. Another example is NoSQL (Not-only-SQL) database systems which have been proposed as alternatives to traditional database systems to meet today's data management requirements, such as scalability, flexibility and fast query performance [8, 9]. Furthermore, as sensing and Internet of Things (IoT) technologies mature and become increasingly prevalent, many cloud-based IoT platforms have been developed to facilitate the connection of numerous physical devices, as well as data exchange among the devices and applications [10]. Nonetheless, there is no one-size-fit-all technology: such contemporary computing technologies have to be tailored to meet domain-specific requirements [11]. Engineering applications often involve diverse types of information ranging from heterogeneous sensor data (e.g., high-frequency time-series data, video and camera images, etc.) to domain-specific engineering information (e.g., geometric models, engineering simulation models, etc.), imposing additional data management requirements [12, 13]. For example, sensor data needs to be integrated with engineering information to enable effective data retrieval and utilization. In addition, efficient information sharing and data exchange are required because engineering projects typically involve a wide variety of software tools, as well as ad hoc analysis modules [14]. A data management platform needs to be designed to meet these requirements in order to support engineering applications effectively.

In the engineering community, on the other hand, information modeling has gained enormous attention as a vehicle to support integrated project delivery processes [14]. Information modeling enables information sharing and integration, as well as seamless data exchange among software based on interoperability standards [15]. However, the engineering community seldom pays attention to data issues involving scalable big data management and standardized communications with client devices. As the use of sensors in engineering applications continues to grow, appropriate data management tools for handling a large amount of sensor data together with engineering domain information will become very important. For effective data management in engineering applications, it is essential to investigate state-of-the-art information technologies and adopt them properly.

To overcome the imminent data issues in civil infrastructure monitoring, this thesis aims to propose a cyberinfrastructure platform that can efficiently store, manage, retrieve, share, interface, link and integrate data. This thesis brings together various information and communication technologies (ICT), including information modeling, NoSQL database systems, cloud computing and web services, for effective data management, particularly, for civil infrastructure monitoring.

1.2 Data Management Requirements for Cyberinfrastructure Platform

"Big data" is typically characterized by three "V"'s, which are volume, velocity and variety [16]. Civil infrastructures instrumented with sensors also need to handle data with the same characteristics. First, the volume of data collected from civil infrastructure will grow significantly as an increasing number of sensors are deployed. The huge data volume requires a highly scalable data management system that can handle a large and increasing amount of data. Second, the velocity (or the rate) of data acquired means that data collection and analysis need to be conducted in a timely manner [17]. To this end, civil infrastructure monitoring systems need to be able to handle, manage and analyze data effectively. Third, civil infrastructure monitoring systems involve a high variety of data, including, but not limited to, time-series data, video, image, geometric models and physical models. Data of heterogeneous types needs to be linked and integrated in order to facilitate data retrieval and utilization. A data management system that satisfies such big data requirements needs to be developed. Based on the big data characteristics and the domain requirements, the following sections summarize the desirable characteristics of a cyberinfrastructure platform for effective data management.

1.2.1 Scalability

As the deployment of sensors increases, the volume of data collected and processed will continue to grow. Civil infrastructure monitoring systems typically collect hundreds of gigabytes of data every year [18]. With the decreasing cost of sensors, the trend of civil infrastructure monitoring involves the instrumentation of hundreds and thousands of sensors to collect extensive information about the target systems [19]. Furthermore, the adoption of advanced sensors with very high sampling rates, such as piezo lead zirconate titanate (PZT) and fiber Bragg grating (FBG) sensors [3, 20, 21], and the use of image and video data [13] lead to a significant increase in data volume. To cope with such a voluminous amount of data, a cyberinfrastructure platform needs to be easily scaled according to processing and storage demands.

1.2.2 Data integration and interoperability

Civil infrastructure monitoring systems involve a wide variety of information, including geometry, engineering model, inspection report, sensor information and sensor data, collected from different data sources. Current practice typically employs isolated systems to manage and process different types of information wherein information managed in one system is neither integrated nor shared among other systems. The isolated data management makes it difficult to compare and combine heterogeneous data; instead, different types of data are manually converted, mapped and compared, which is error-prone and time-consuming [15, 22]. Effective sharing and integration of data would facilitate data utilization and, thus, enhance operation and management of engineering systems [23, 24].

To this end, a cyberinfrastructure platform needs to be designed to support data integration and interoperability.

1.2.3 Standardized interface

Data managed by cyberinfrastructure needs to be easily accessed by various applications (e.g., visualization tools, finite element analysis tools and machine-learning modules) on different systems (e.g., cloud computing platforms, desktop computers and mobile devices). In current civil infrastructure monitoring systems, however, data management tools typically lack standardized interfaces, resulting in manual data download and application execution. To facilitate data access and utilization, standardized interfaces are essential. By providing standardized interfaces, machine-to-machine data exchange among cyberinfrastructure and different applications can be enabled. Furthermore, standardized interfaces can enable easy prototyping and rapid deployment of applications in a plug and play manner. Therefore, cyberinfrastructure needs to support standardized interface.

1.2.4 Flexibility

Over the lifecycle of an engineering system, the users' (e.g., system operators and engineers) requirements are constantly changing. For example, new types of sensors may be deployed with an existing cyberinfrastructure platform to collect additional information about the physical system. Needs for managing previously-unmanaged information can arise as knowledge and insight about system operations grow. In addition, as new data analysis modules are developed, new services need to be deployed to interface the cyberinfrastructure platform with the new analysis modules. A cyberinfrastructure platform should be able to adapt to the changing environment in a timely and cost-effective manner. Therefore, a cyberinfrastructure platform has to be designed with consideration of system flexibility.

1.3 Current Sensor Data Management Practice

1.3.1 Internet of Things (IoT) platform

With the increasing adoption of sensor technology, many IoT platforms have been developed. Cloud computing service vendors offer generic IoT platforms, including AWS IoT by Amazon AWS [25], IoT Hub by Microsoft Azure [26], Cloud IoT Core by Google Cloud [27], Watson IoT Platform by IBM cloud [28], and AT&T IoT Platform by AT&T [29]. These generic IoT platforms support device connectivity via standard communication protocols with high scalability by leveraging cloud computing technology. These platforms also provide many tools, such as device management tools, rule engines, event processing modules, security tools and software development kits (SDKs). While these generic IoT platforms provide basic services, they lack the supporting services for domain-specific applications and data management tools. Instead, application services need to be developed and added by customers or partner companies. There have been some domain-driven IoT platforms, such as PTC ThingWorx [30] and AutoDesk Fusion Connect [31], particularly for industrial IoT (IIoT). While IIoT platforms offer some industrial applications and sophisticated functions, such as augmented reality (AR)-enabled user interfaces, the IIoT platforms are not designed to manage engineering information models and do not support data and software interoperability.

Research efforts have been spent on the development of IoT platforms for specific application areas, such as healthcare [32, 33], smart cities [34, 35, 36] and agriculture [37]. A domain-specific platform handles not only sensor data, but also other relevant information. For example, Lea and Blackstock [35] describe an IoT platform for smart city applications to manage a wide array of data, from real-time (e.g. traffic data) to static data (e.g., asset lists). However, this work does not address data integration for linking heterogeneous sensor data and domain information. To allow software agents to easily discover relevant information and to perform analysis, domain information and sensor data need to be properly linked and integrated. Jayaraman *et al*. [37] describe a semantic-driven

IoT platform to link sensor data and domain concepts based on ontology definitions. An IoT platform proposed in [34] enables interoperability among heterogeneous information models, such as building information modeling (BIM) and system information models (SIM), based on sematic web technology. However, previous and current studies do not address the interoperability problem which is critical in engineering projects involving various software tools, each of which may have its own interface and data model. Engineering information needs to be exposed in platform-neutral and standardized data formats that can be easily parsed and used by different software agents, ranging from engineering simulation tools to data-driven analysis modules. Therefore, a data management platform that can deal with both data integration and data interoperability problems needs to be developed to facilitate data management and utilization in civil infrastructure monitoring.

1.3.2 Civil infrastructure monitoring systems

Until now, research efforts on civil infrastructure monitoring have been mostly focused on the development of new sensor technologies and data analysis techniques. Very little efforts have been devoted to the fundamental issues associated with data management. Early civil infrastructure monitoring systems typically collect and store measurement data in files on local computers [38, 39, 40]. File-based systems, however, do not directly support queries, which often makes data access a tedious and manual task. For systematic data management, relational database management systems (RDBMSs) have been used as centralized data storage [41, 42, 43, 44]. RDBMSs support queries with structured query language (SQL) to facilitate data access. However, research studies have suggested that current RDBMSs, which were architected decades ago when the characteristics of hardware and data processing requirements were very different, are often not as effective in meeting the data needs of today’s applications which often involve text, time-series data, image files and video data [45, 8]. Some fundamental limitations of RDBMSs [46, 47, 48] which can impact civil infrastructure monitoring applications include:

- *Reading and writing speed* for processing a large amount of time-series monitoring data;
- *Scalability* for handling voluminous and ever-increasing amount of monitoring data;
- *Schema flexibility* for managing engineering data that include semi- and un-structured information (e.g., information models).

Furthermore, civil infrastructure monitoring systems are typically designed to keep data in storage systems without considering how to effectively retrieve, utilize and integrate data. Ineffective data management often results in poor utilization of data. Some example problems from practical civil infrastructure monitoring systems include the following:

- *Limited data store* that would lead to removing old data before utilizing it and require averaging time-series data to reduce data volume, resulting in the loss of potentially important information;
- *Limited query capability* due to storing data un-systematically (which poses restrictions on data retrieval and query) and managing data using a data management tool that may not offer a good interface (requiring significant labor to manually download and map the data);
- *Limited data utilization* due to considering sensor data only as subsidiary information that does not directly affect decision making process.

In summary, current civil infrastructure monitoring systems are not designed to handle today's ever-increasing heterogeneous monitoring data, resulting in data solutions not meeting the data management requirements demanded by the application. This thesis describes a cyberinfrastructure platform that satisfies the data management requirements of infrastructure applications by leveraging state-of-the-art information and communication technologies.

1.4 Information and Communication Technologies for Effective Data Management

1.4.1 Information modeling

Much research has been conducted in developing data exchanges and interoperability standards in many industry domains to avoid error-prone and time-consuming manual data conversion, as well as to facilitate the automated exchange of information and machine-to-machine interaction [49, 50]. In the architecture, engineering and construction (AEC) industry, for example, BIM has been widely adopted as a means to support an integrated project delivery process and data exchange throughout the project lifecycle of buildings [14]. One of the *de facto* BIM standard data models is the Industry Foundation Classes (IFC) [51]. The IFC standard specifies a 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 an XML (eXtensible Markup Language) format, a commonly used representation of industry standards [51]. Given the success of BIM, bridge information modeling (BrIM) has been developed for bridges [22, 52, 53, 54, 55]. These efforts have so far been focused mostly on the 3-dimensional geometric representation of engineering systems. As such, current information modeling practice often lacks the data entities needed for representing the information pertinent to sensor networks and the data links to connect information models with sensor data stores. Therefore, information modeling schemas have to be extended to include data entities that capture information about sensor networks and engineering models.

1.4.2 Cloud computing

Advances in cyber physical systems and cloud computing services share many significant components that can be deployed for the management of infrastructure monitoring data. Cloud computing, as defined by the National Institute of Standards and Technology

(NIST), is a "model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [56]." Cloud computing can reduce the cost and lessen the burdens on the deployment, operation, maintenance and management of computational resources. Furthermore, cloud computing can provide a highly scalable and accessible computing environment that is cost-effective [7, 57, 58, 59]. Many state-of-the-art data management platforms take advantage of cloud computing to allow communication and data sharing among the physical systems, sensors, software applications and users. Using cloud computing, a civil infrastructure monitoring system can be easily scaled based on demand with computing and storage resources optimized.

1.4.3 NoSQL database

For data management on cloud computing and distributed computing environments, many alternative database management systems (DBMS) have been developed. Driven by the need for storing, managing and retrieving large online data records with heterogeneous formats, research has been devoted to develop non-relational database and non-traditional file management systems. Examples of open source database systems that have been deployed by cloud service providers include Apache Cassandra, Apache H-Base and MongoDB [9]. These non-traditional database systems are noted as NoSQL (Not only SQL) database systems which are designed to handle semi-structured and unstructured data. Recent studies have shown that NoSQL database systems have significant advantages over RDBMSs in terms of flexibility and scalability [46, 47, 48]. For example, Le *et al.* [60] proposed an Internet of Things (IoT) platform and concluded that NoSQL database systems, such as Apache Cassandra, consistently have better performance than RDBMSs for managing sensor data because of the flexible data structure suitable to query time series data. Furthermore, NoSQL database systems have been shown to have better scalability in handling massive IoT data and have better query performance for sensor network data [61, 62].

1.4.4 Web service

To take advantage of cloud computing, the software framework should be designed with consideration of the useful features provided by cloud services (e.g., dynamic provisioning, distributed computing and on-demand commodity hardware), as well as domain-specific application requirements (e.g., information model, application, interface, etc.). In contrast to traditional proprietary servers, the real value of cloud computing relies upon interoperability among systems and engineering services [63]. For service interoperability, engineering services on cloud platforms need to be exposed via standard interfaces. There are two main web service paradigms: namely, Service-Oriented Architecture (SOA) and Resource-Oriented Architecture (ROA). SOA is built upon standard web service protocols, such as Simple Object Access Protocol (SOAP), Web Service Description Language (WSDL) and Business Process Execution Language (BPEL), etc. [64]. While SOA's reliability and message-level security benefit enterprise-level applications, the complexity of the protocols makes them less attractive for basic, *ad hoc* integration of services [65]. ROA, on the other hand, is based on a simple set of Representational State Transfer (REST) protocols [66]. REST has become a preferable approach because of its simple and lightweight architecture, easy accessibility and scalability [67, 68, 69]. Through web services, a data management platform can enable easy access to data, information exchanges and integration of software services for civil infrastructure monitoring applications.

1.5 Research Objectives

As the use of sensors increases, data management issues have already started to present challenges to civil infrastructure monitoring practice. While there have been various information technologies developed to meet today's data management requirements, current research efforts rarely pay attention to using such technologies to address data issues in civil infrastructure monitoring. The goal of this thesis is thus to propose a cyberinfrastructure platform to deal with the data issues in civil infrastructure monitoring. The cyberinfrastructure platform is designed according to the data management requirements of civil infrastructure monitoring applications, which are: (1) scalability, (2) data integration and interoperability, (3) standardized interface, and (4) flexibility. The platform brings together state-of-the-art information and communication technologies for effective data management. Specifically, an information modeling framework is developed to represent information involved in civil infrastructure monitoring in an integrated manner, as well as to facilitate data interoperability based on open standards. A prototype NoSQL database system is designed and deployed for the scalable management of massive sensor data and flexible management of semi- and un-structured engineering information with fast query performance on a distributed computing environment. The cyberinfrastructure platform is built upon a cloud computing platform for scalability, reliability and minimal maintenance effort of computing resources. Leveraging standardized web-based interfaces, the cyberinfrastructure platform offers easy-to-use data management services that client systems can invoke via standard communication protocols to store, retrieve and share different types of data involved in civil infrastructure monitoring.

By orchestrating various data management technologies, the proposed cyberinfrastructure platform will enable scalable management of voluminous and heterogeneous data involved in civil infrastructure monitoring. Unlike current sensor data management systems, the cyberinfrastructure platform allows different data sources (e.g., sensors, cameras and modeling tools) and analysis modules (e.g., structural analysis tools and machine learning modules) to easily connect to the platform to store and retrieve various information involved in various monitoring applications. The demonstration results presented in this work show that the cyberinfrastructure platform can effectively and scalably manage sensor data and domain-specific information and facilitate data sharing, integration and utilization.

1.6 Thesis Outline

The thesis presents an information modeling schema for civil infrastructure monitoring applications. The cyberinfrastructure platform consisting of a cloud computing platform, NoSQL database and web server is then presented. Finally, the implementation of a data analysis pipeline leveraging the cyberinfrastructure platform is then described. The developed cyberinfrastructure platform is demonstrated with the different types of data obtained from the I-275 corridor monitoring system. The thesis is organized into the six chapters as follows:

- Chapter 2 presents an information modeling framework to facilitate data integration and interoperability for supporting civil infrastructure monitoring applications [70]. The framework augments and extends the prior BrIM (Bridge Information Modeling) standards to further capture the information relevant to engineering analysis and sensor network. For the representation of engineering analysis information, the framework draws on the data entities of one of the widely used commercial bridge modeling and analysis tools. For the representation of sensor network information, the framework adopts data entities defined by an open standard for describing sensor systems.
- Chapter 3 describes a scalable and flexible data management framework for civil infrastructure monitoring [71, 72]. The data management framework chooses NoSQL database systems suitable to meet the data management requirements. Specifically, Apache Cassandra database is employed for scalable data management, whereas MongoDB is employed for efficient data retrieval. A database schema is defined based on the BrIM schema to enable data mapping between the BrIM schema and the database schema.
- Chapter 4 presents a cloud-based cyberinfrastructure platform for civil infrastructure monitoring to offer scalable and interoperable data management services [73, 74, 75, 76]. Using the services of this cloud platform, different client

systems (e.g., data source and applications) can easily connect to store and retrieve data via standard communication protocols. Furthermore, the platform-neutral data management services can facilitate service composition and data utilization. This chapter then discusses the adoption of cloud computing platform for the development of a scalable and easy-to-manage cyberinfrastructure platform. Data privacy and security concerns under cloud computing environments are also addressed.

- Chapter 5 presents a sensor data reconstruction method using the cyberinfrastructure platform [77]. Specifically, this study develops a data-driven sensor data reconstruction method based on bidirectional recurrent neural networks, which can improve reconstruction accuracy by considering spatiotemporal correlation among the sensor data. An automated data analysis pipeline is built upon the cyberinfrastructure platform to enable efficient data reconstruction by training the neural network on a high-performing computer and sharing the trained model to local computing platforms through the cyberinfrastructure platform.
- Chapter 6 summarizes the development of the cyberinfrastructure platform for civil infrastructure monitoring for facilitating scalability, flexibility and interoperability. This chapter provides the contributions of this thesis and then suggests potential future research directions to continue advancing the field of informatics in infrastructure.

Chapter 2

Information Modeling Framework for Bridge Monitoring

2.1 Introduction

Bridge management involves copious and diverse information, ranging from various semi-structured or unstructured data (e.g., geometric model, engineering model, inspection report, sensor metadata, etc.) to a large amount of heterogeneous sensor data (e.g., acceleration, displacement, photo, video image, etc.). Current practice of bridge management typically 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, the demand for efficient information sharing and data exchange will grow. Sharing and integration of such information would enable integrated data retrieval and utilization, improve bridge management services and, thus, enhance bridge operation, maintenance and public safety.

Given the success of building information modeling (BIM), research efforts have been initiated to develop frameworks and standards for bridge information modeling (BrIM) [15]. The main goals of BrIM are twofold: enabling integrated bridge data repository and developing electronic data exchange standards for bridge application [15]. For example, there have been research efforts aimed towards developing an information modeling framework for the integration of bridge management information and 3-dimensional bridge models [52, 53]. To facilitate information interoperability in the bridge domain, an extension of IFC, namely IFC-Bridge, has been proposed with emphasis on the spatial and physical entities of bridge structures [54]. As another example of BrIM standards, the OpenBrIM standards, which is supported by the US Federal Highway Administration (FHWA), have been proposed as a "bridge industry consensus standard for engineering data description, modeling, and interoperability for integrated structural design, construction, and lifecycle management of bridges [22, 55]." Efforts to advance BrIM standards so far have been focused on the 3-dimensional geometric representation of bridge structures, while the standards lack the data entities needed for representing the information pertinent to bridge monitoring applications.

This chapter presents a BrIM framework for bridge monitoring applications [70]. The framework aims to facilitate the exchange and integration of information involved in bridge management applications. The BrIM framework adopts and extends the data schema of the OpenBrIM standards to support data interoperability between bridge monitoring and management applications. New data entities are defined to capture information associated with bridge engineering analyses, sensor descriptions and bridge monitoring systems. The framework also provides a data link to the time-series sensor data so as to allow users to locate the data via the information model.

This chapter is organized as follows. Section 2.2 presents the bridge information modeling framework for bridge monitoring. Specifically, the BrIM schema of the OpenBrIM standards is employed as a base model and enriched to include data entities for the representation of finite element models and sensor information. Section 2.3 demonstrates the bridge information modeling framework with the bridge information collected from the Telegraph Road Bridge (TRB) located in Monroe, Michigan. This chapter is concluded with a summary in Section 2.4.

2.2 Bridge Information Modeling Schema Definition

This section describes the design of a bridge information modeling schema for bridge monitoring applications [70]. Engineering information modeling, such as BIM, typically adopts an object-based approach that describes a target system (e.g., building) using objects and their attributes [23]. 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 the 3-dimensional geometry of bridge structures [78]. The BrIM schema developed in this study extends the data schema of the OpenBrIM standards with newly defined objects for representing engineering analysis models and sensor information. New objects are identified based on relevant standards and software tools to ensure that the BrIM schema is capable of supporting typical applications in bridge engineering. Specifically, CSiBridge (a structural modeling and analysis software tool) [79] and SensorML (an open standard for describing sensors) [80] are examined for the definition of engineering analysis models and sensor information, respectively.

2.2.1 Base model: OpenBrIM

The OpenBrIM standards describe a bridge structure as a collection of hierarchical objects and their parameters [78]. 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, whereas each parameter either represents a property (e.g., length, width and thickness) of an object or refers to another object. Figure 2.1(a) and (b) show the schema definitions of a basic `Object` entity and a `Parameter` entity, respectively, in OpenBrIM [81]. 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.

```
<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) Definition of Object entity

```
<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) Definition of Parameter entity

Figure 2.1 Definition of a fundamental Object and Parameter entities [78]

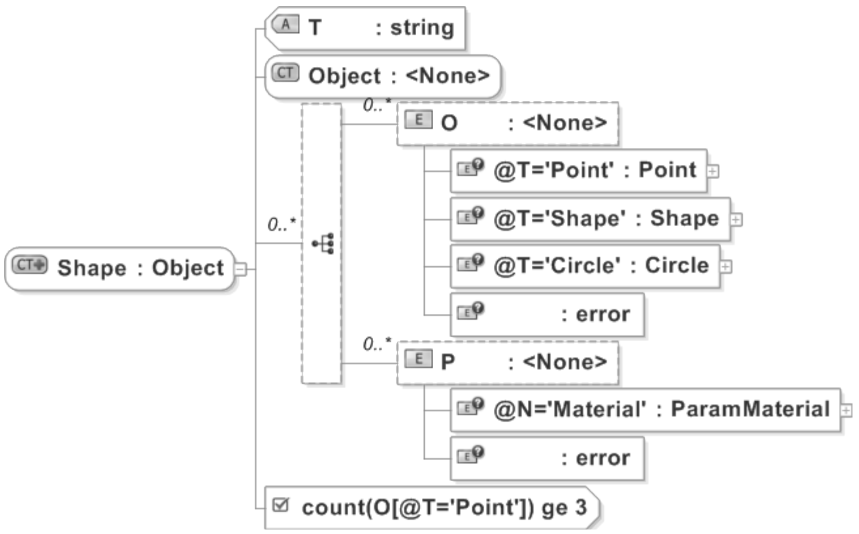
To encode bridge information, OpenBrIM standards use ParamML [82], an XML-based mark-up language for engineering applications. For example, Figure 2.1(a) shows the OpenBrIM schema definition for the Shape object written in the XML schema definition (XSD) format [81]. In the schema definition, `xs` refers to the XML schema namespace [83]. The definitions of the elements from the XML schema namespace are as follows [84].

- `complexType` is an element that contains other elements and/or attributes.
- `complexContent` specifies extensions or restrictions on a `complexType` element.
- `extension` extends an existing `complexType` element.
- `sequence` defines the child elements that can occur.
- `element` defines an XML element.
- `alternative` dynamically assigns the type of its parent element based on the specified test condition. (available from XSD 1.1)
- `attribute` contains data related to its parent entity.
- `assert` specifies the condition used to validate XML data entity.

Given the definitions of the XML elements, the schema definition for a `Shape` object as shown in Figure 2.2(a) specifies the following. The `extension` element indicates that `Shape` is a subtype of the `Object` entity. The `sequence` element specifies the eligible child objects and parameters using `O` and `P` tags, respectively. As shown in the 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 the attribute element enforces that every `Shape` object must include the `T` (type) attribute, which has value `Shape`. Another example is the `assert` element that specifies the condition that the "`Shape` object must contain at least 3 Point objects."

```
<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 2.2 Data schema of the Shape object [78]

The XML schema written in XSD can be displayed as an XSD diagram using visualization tools, such as Liquid XML Studio [85] and XMLSpy [86]. For example, the schema structure of the `Shape` object as shown in Figure 2.2(a) can be displayed by an XSD diagram as shown in Figure 2.2(b). It should be noted that the names of the data components are abbreviated in the diagram. The numbers written at the left side of the XML components represent the possible numbers of the components. For example, in Figure 2.2(b), the numbers `0..*` at the left side of the `O` element indicate that the parent component (i.e., `Shape` object) can have zero to any number of child `O` elements (i.e., child objects). In the following sections, XML data schema will be described using the XSD diagrams for readability purpose.

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 the engineering model and material for structural analysis and structural monitoring.

2.2.2 Finite element modeling

The OpenBrIM standards currently include only a few basic objects for finite element (FE) modeling [81]. 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 CSiBridge, one of the widely used commercial bridge modeling and analysis tools [79], as an example, an FE model of 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 CSiBridge software to OpenBrIM standards' data schema definition.

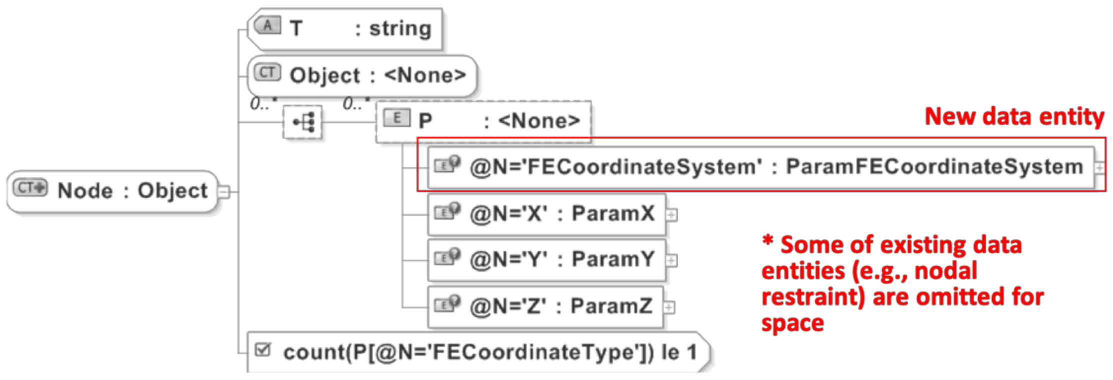
While information models related to finite element analysis exist (such as STEP Part 104 [87] and Industry Foundation Classes [51]), the existing models usually lack the data entities to represent complex load and analysis conditions (such as time-variant vehicle loads) required in bridge engineering 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 the bridge structure information; however, 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 that represent a bridge structure, as well as on defining new objects for representing load and analysis conditions. It should be noted that the data schema is extended solely based on the CSiBridge so far and, thus, additional data entity definitions will be required to enable data mapping between other FE software packages.

2.2.2.1 Data entities for representing finite element model

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. For instance, a `Node` object, which is the most fundamental data entity in FE modeling for specifying the nodal coordinates and restraints, currently defined in the OpenBrIM standards, do not have the parameters necessary for defining a reference coordinate system to conveniently create models using multiple coordinate systems. To augment the `Node` object in OpenBrIM, we create a new object `FECoordniateSystem` 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 `FECoordniateSystem` as shown in the XSD diagram in Figure 2.3.

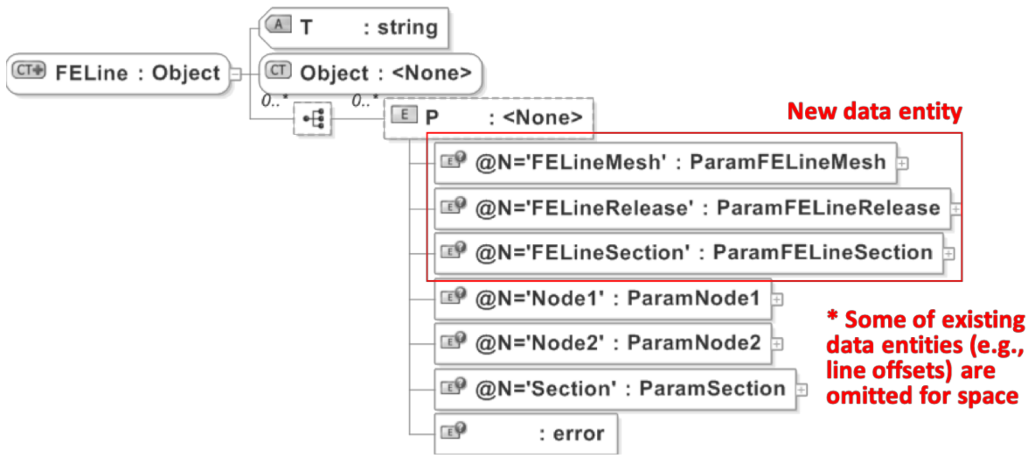
An `FELine` object represents an element that consists of two nodes and section information in an FE model. The current definition of `FELine` object in OpenBrIM includes data entities for describing the two `Nodes` and `Section`, but the object definition does not include data entities to represent information about discretization and member-end-release. In addition, the `Section` object, while it is suitable to represent a user-defined section shape composed of many section points, it does not allow for 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 2.4.

Similarly, the current definition of `FESurface` object in OpenBrIM represents an element consisting of vertices, thickness and material types, such as shell and wall, but the schema



T : string
Object : <None>
0..*
0..*
P : <None>
New data entity
@N='FECoordinateSystem' : ParamFECoordinateSystem
Node : Object
@N='X' : ParamX
@N='Y' : ParamY
@N='Z' : ParamZ
* Some of existing data entities (e.g., nodal restraint) are omitted for space
count(P[@N='FECoordinateType']) le 1

Figure 2.3 Entity: Node



T : string
FELine : Object
Object : <None>
0..*
0..*
P : <None>
New data entity
@N='FELineMesh' : ParamFELineMesh
@N='FELineRelease' : ParamFELineRelease
@N='FELineSection' : ParamFELineSection
@N='Node1' : ParamNode1
@N='Node2' : ParamNode2
@N='Section' : ParamSection
: error
* Some of existing data entities (e.g., line offsets) are omitted for space

Figure 2.4 Entity: FELine

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 2.5.

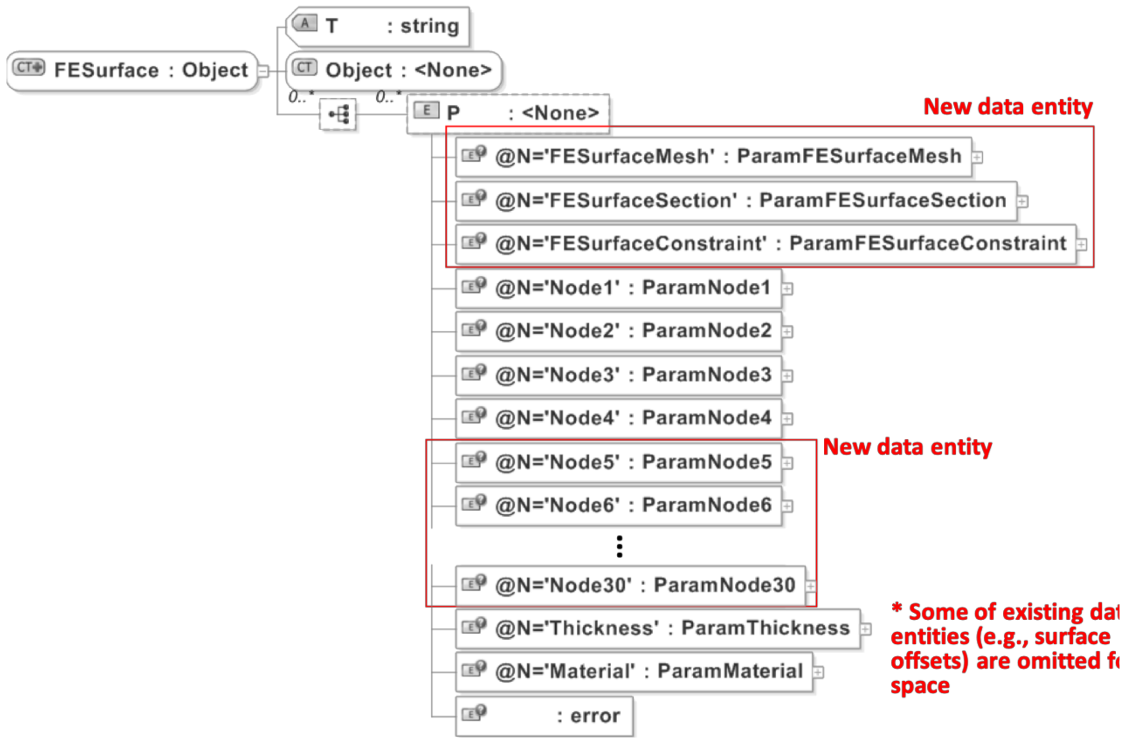


Figure 2.5 Entity: FESurface

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 compressive strength. To further enhance the definition of the Material object for structural analysis purposes, new parameters, such as `Symmetricity`, `TemperatureDependency`, `ShearModulus` and various damping properties, are added. The enhanced the `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 2.6 shows the data schema diagram of the enhanced `Material` object.

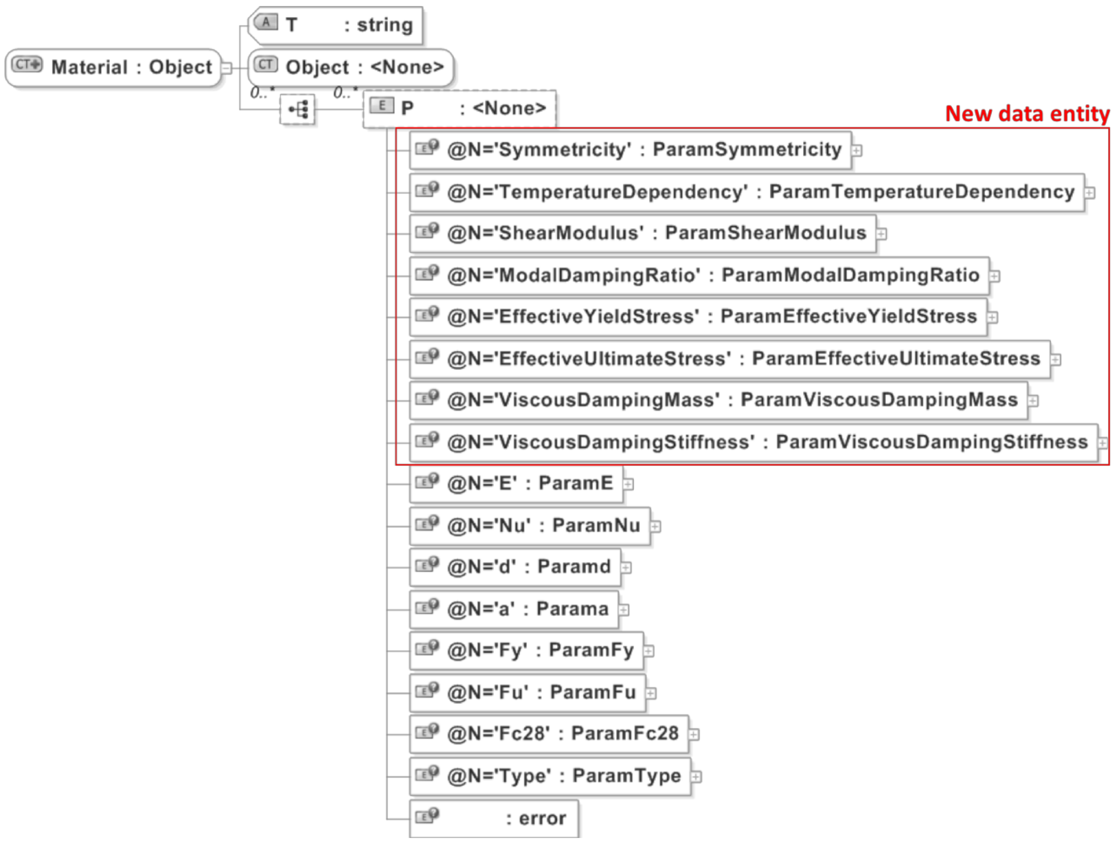

T : string
Material : Object
Object : <None>
0..*
0..*
P : <None>
New data entity
@N='Symmetricity' : ParamSymmetricity
@N='TemperatureDependency' : ParamTemperatureDependency
@N='ShearModulus' : ParamShearModulus
@N='ModalDampingRatio' : ParamModalDampingRatio
@N='EffectiveYieldStress' : ParamEffectiveYieldStress
@N='EffectiveUltimateStress' : ParamEffectiveUltimateStress
@N='ViscousDampingMass' : ParamViscousDampingMass
@N='ViscousDampingStiffness' : ParamViscousDampingStiffness
@N='E' : ParamE
@N='Nu' : ParamNu
@N='d' : Paramd
@N='a' : Parama
@N='Fy' : ParamFy
@N='Fu' : ParamFu
@N='Fc28' : ParamFc28
@N='Type' : ParamType
: error

Figure 2.6 Entity: Material

In addition to the entities described above, new data entities and their parameters are defined, as summarized in Table 2.1, to complete the schema definitions of the `Node`, `FELine`, `FESurface` and `Material` objects in OpenBrIM.

2.2.2.2 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 [81]. 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). CSiBridge, for example, describes load conditions and analysis conditions using “load patterns” and “load cases”, respectively [79]. The load patterns are the spatial distribution and magnitude of forces and other effects acting on a structure, while the load cases are the analysis options that include applied load

Table 2.1 Objects added to the OpenBrIM model for representing structural elements

Object	**Parameters**
`FECoordinateSystem`	`FECoordinateType`, `OriginX`, `OriginY`, `OriginZ`, `OriginRX`, `OriginRY`, `OriginRZ`
`FELineMesh`	`AutoMesh`, `MeshAtJoints`, `MeshAtFrames`, `NumberOfSegments`, `MaxMeshLength`, `MaxMeshDegree`
`FELineSection`	`Material`, `Shape`, `Width`, `Height`, `WebThickness`, `FlangeThickness`
`FELineRelease`	`NodeV1`, `Node1V2`, `Node1V3`, `Node1M1`, `Node1M2`, `Node1M3`, `NodeV1`, `Node2V2`, `Node2V3`, `Node2M1`, `Node2M2`, `Node2M3`
`FESurfaceMesh`	`Meshtype`, `MeshGroup`, `NumberOfObject1`, `NumberOfObject2`, `MaxSize1`, `MaxSize2`, `MeshFromSelectedLine`, `MeshFromSelectedPoint`, `ConstraintEdge`, `ConstraintFace`
`FESurfaceSection`	`Material`, `MaterialAngle`, `SurfaceType`, `Thickness`, `BendThickness`
`FESurfaceConstraint`	`EdgeConstraint`

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 CSiBridge 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 2.7. 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.

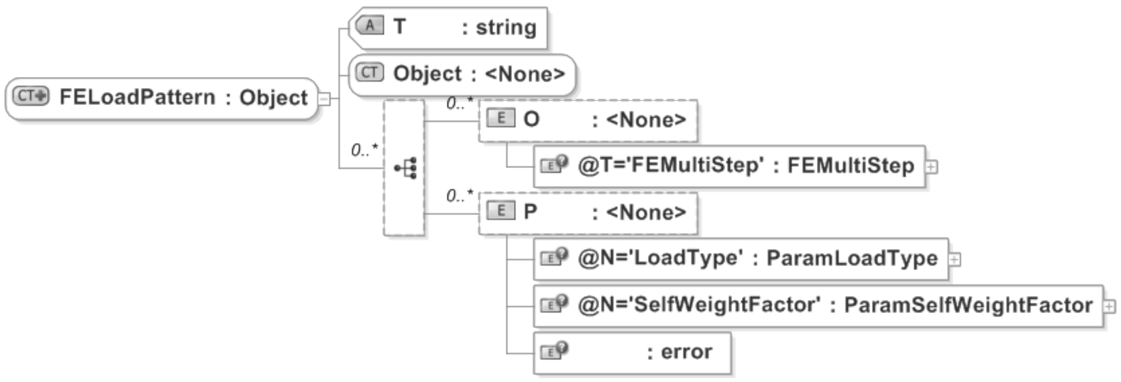

A T : string
CT Object : <None>
CT FELoadPattern : Object
0..*
0..*
E O : <None>
@T='FEMultiStep' : FEMultiStep
0..*
E P : <None>
@N='LoadType' : ParamLoadType
@N='SelfWeightFactor' : ParamSelfWeightFactor
: error

Figure 2.7 New entity: FELoadPattern

For the representation of analysis conditions, we create a new object `FEAnalysisCase`. Figure 2.8 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, `FEStatic` 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 2.2 summarizes the new data entities created under the `FELoadPattern` and `FEAnalysisCase` objects to represent load and analysis conditions.

2.2.3 Sensor description

There have been several standards developed to describe sensor information and measurement data [88, 89]. The Sensor Web Enablement (SWE) standards by the Open

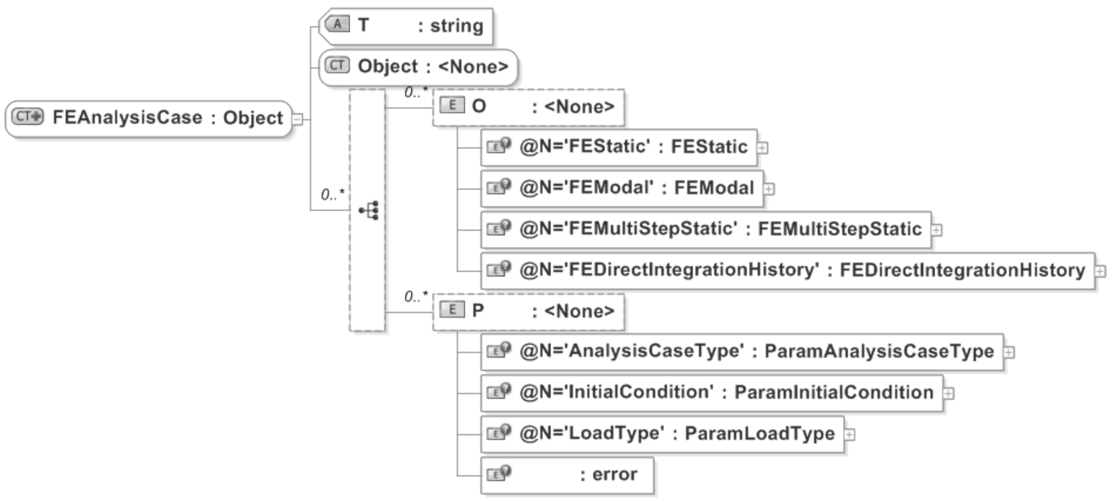

T : string
Object : <None>
FEAnalysisCase : Object
0..*
O : <None>
@N='FEStatic' : FEStatic
@N='FEModal' : FEModal
@N='FEMultiStepStatic' : FEMultiStepStatic
@N='FEDirectIntegrationHistory' : FEDirectIntegrationHistory
0..*
P : <None>
@N='AnalysisCaseType' : ParamAnalysisCaseType
@N='InitialCondition' : ParamInitialCondition
@N='LoadType' : ParamLoadType
: error

Figure 2.8 New entity: FEAnalysisCase

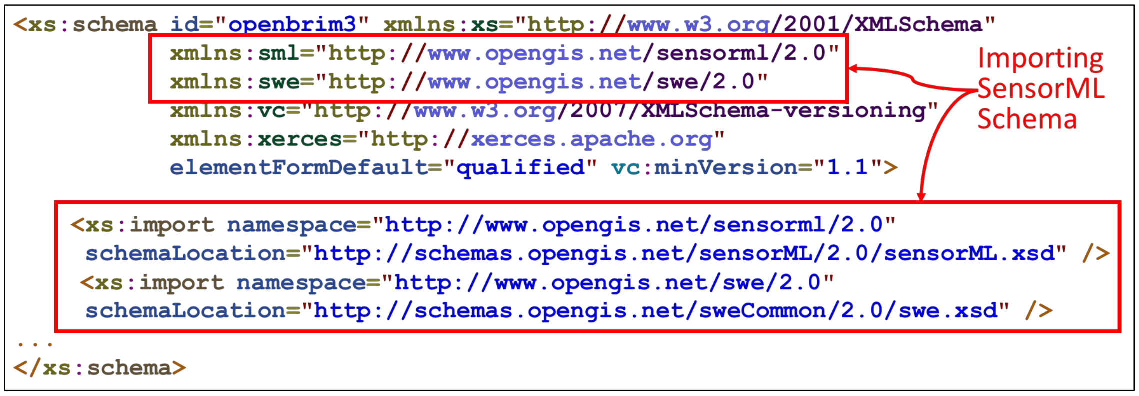
Table 2.2 Objects added to the base OpenBrIM model for representing load and analysis conditions

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

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 an XML-based data format for the description of sensor metadata as well as processes and processing components associated with the sensors [80]. 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 2.9, where the `xmlns:sml` prefix in the figure refers to the namespace of SensorML standards.

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 [80]:

- `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.



```
<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 2.9 Importing SensorML's schema

- `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 a string to refer to the ID of a geometric element that the sensor is attached to. Another entity added is the `FENode`, whose value type is also a string, to include the ID of the 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 2.10.

As shown in Figure 2.11, 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 2.3 summarizes the new data entities for the different sensor types. The data entities (as denoted as grandchild elements in Table 2.3) 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.

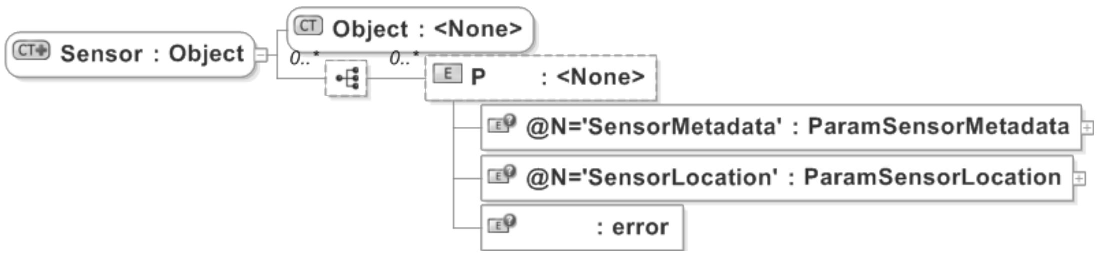

Sensor : Object
Object : <None>
0..*
0..*
P : <None>
@N='SensorMetadata' : ParamSensorMetadata
@N='SensorLocation' : ParamSensorLocation
: error

Figure 2.10 New entity: Sensor

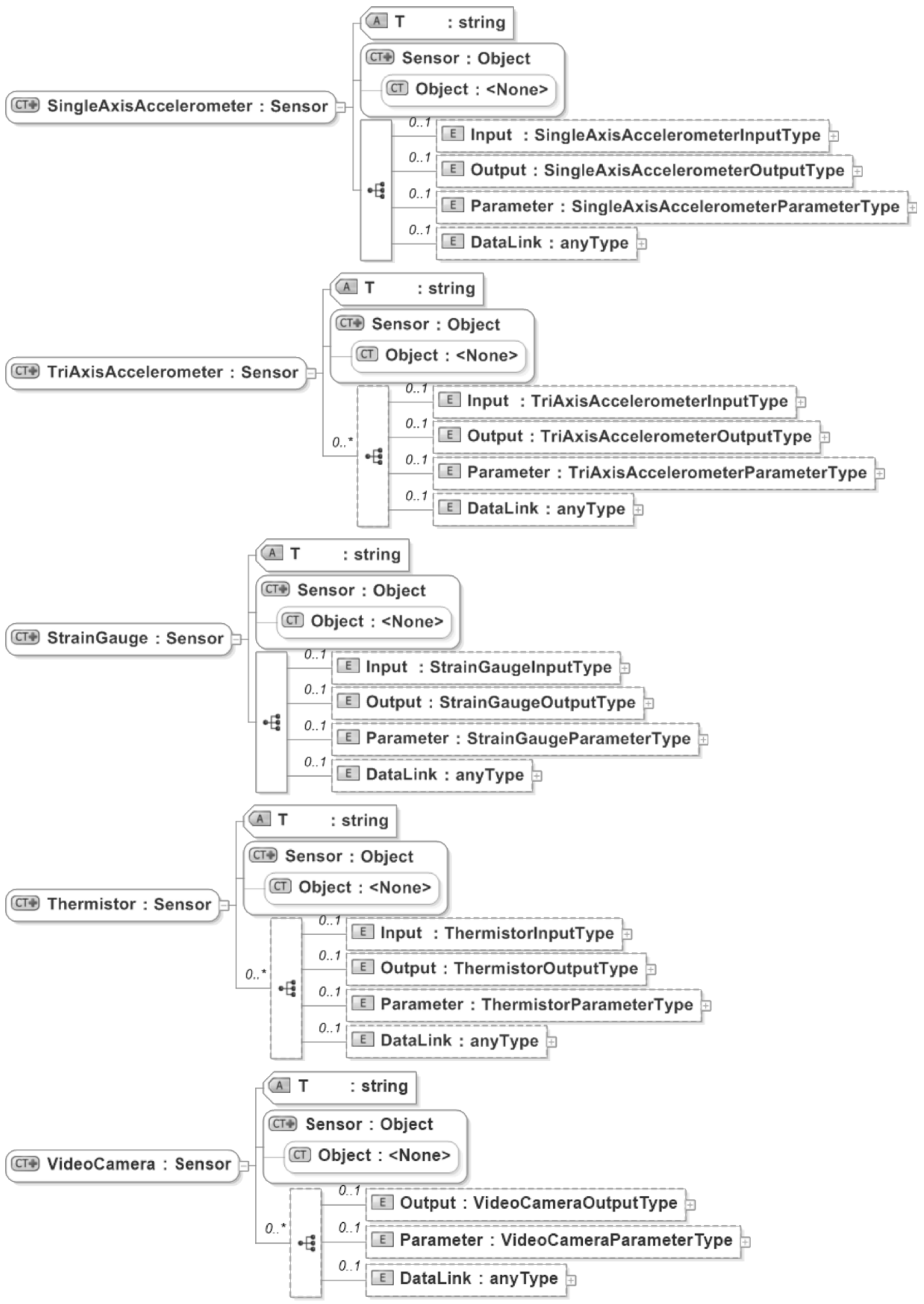

T : string
Sensor : Object
Object : <None>
SingleAxisAccelerometer : Sensor
0..1 Input : SingleAxisAccelerometerInputType
0..1 Output : SingleAxisAccelerometerOutputType
0..1 Parameter : SingleAxisAccelerometerParameterType
0..1 DataLink : anyType
T : string
Sensor : Object
Object : <None>
TriAxisAccelerometer : Sensor
0..*
0..1 Input : TriAxisAccelerometerInputType
0..1 Output : TriAxisAccelerometerOutputType
0..1 Parameter : TriAxisAccelerometerParameterType
0..1 DataLink : anyType
T : string
Sensor : Object
Object : <None>
StrainGauge : Sensor
0..1 Input : StrainGaugeInputType
0..1 Output : StrainGaugeOutputType
0..1 Parameter : StrainGaugeParameterType
0..1 DataLink : anyType
T : string
Sensor : Object
Object : <None>
Thermistor : Sensor
0..*
0..1 Input : ThermistorInputType
0..1 Output : ThermistorOutputType
0..1 Parameter : ThermistorParameterType
0..1 DataLink : anyType
T : string
Sensor : Object
Object : <None>
VideoCamera : Sensor
0..*
0..1 Output : VideoCameraOutputType
0..1 Parameter : VideoCameraParameterType
0..1 DataLink : anyType

Figure 2.11 New entities: Sensor subtype object

Table 2.3 Data entities included in the Sensor subtype objects.

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

2.3 Case Example

This section demonstrates the use of the proposed BrIM schema for the representation of bridge information. For demonstration, the bridge information of the Telegraph Road Bridge (TRB) is employed. The TRB is a 68-meter long highway overpass located at Monroe, Michigan, as shown in Figure 2.12. The TRB has been monitored with a structural

Figure 2.12 Telegraph Road Bridge (Monroe, Michigan)

monitoring system installed and operated by a research team at University of Michigan since 2011 [90, 91]. The information considered in this case example includes the geometry and finite element model of the bridge, as well as sensor description. The geometry of the bridge is currently described using 2-dimensional drawings. The finite element model of the bridge is created using CSiBridge [79] to perform numerical simulation of structural behavior of the bridge. The sensor information, including sensor ID, sampling rate, physical characteristic, electrical characteristics, etc., is currently recorded in a Microsoft Excel Spreadsheet.

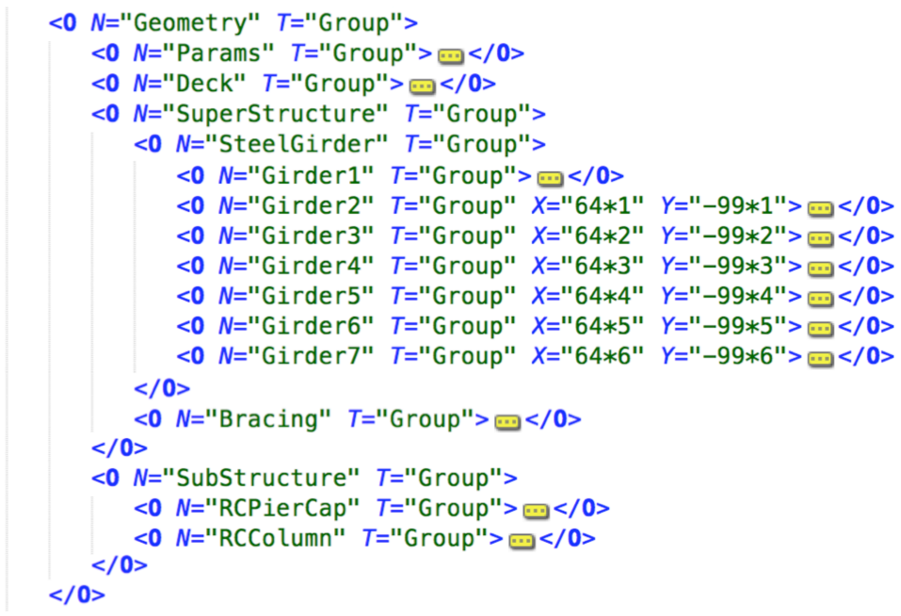
The following sections describe the representation of geometric model, engineering model and sensor information of the TRB using the proposed BrIM schema. While the TRB also involves sensor data, it is not efficient to represent sensor data using the object-oriented BrIM schema; instead, sensor data is stored in a database system (which will be discussed in Chapter 3) and linked via uniform resource identifiers (which will be discussed in Chapter 4).

2.3.1 Geometric model

The 3-dimensional geometry of the TRB can be represented using the OpenBrIM's original data entities. In this study, the geometric model of the TRB is created by writing a ParamML script based on the 2-dimensional drawings of the bridge. The script has a hierarchical object-oriented structure which can describe the hierarchical relationship among the bridge elements. For example, Figure 2.13 shows a high-level structure of the ParamML script. In this figure, the geometric model (i.e., `geometry` object) is composed of its child objects, such as `Deck`, `Superstructure` and `Substructure`. This figure also shows that the `Superstructure` object consists of child objects `SteelGirder` and `bracing`, where the `SteelGirder` object has child objects from `girder1` to `girder7`. In addition, the `Substructure` object has child objects `RCPierCap` and `RCColumn`. The created geometric model can be visualized using applications that support OpenBrIM standards. For example, Figure 2.14 shows the geometric model of TRB visualized using OpenBrIM Viewer [81].

2.3.2 Finite element model

The BrIM data schema proposed in this study can be used for the representation of finite element models. To validate the BrIM data schema, data mapping between the CSiBridge's data schema and the BrIM data schema is performed using the TRB's finite element model. Figure 2.15 shows the TRB's finite element model which is created using the CSiBridge.



```
<O N="Geometry" T="Group">
   <O N="Params" T="Group">…</O>
   <O N="Deck" T="Group">…</O>
   <O N="SuperStructure" T="Group">
      <O N="SteelGirder" T="Group">
         <O N="Girder1" T="Group">…</O>
         <O N="Girder2" T="Group" X="64*1" Y="-99*1">…</O>
         <O N="Girder3" T="Group" X="64*2" Y="-99*2">…</O>
         <O N="Girder4" T="Group" X="64*3" Y="-99*3">…</O>
         <O N="Girder5" T="Group" X="64*4" Y="-99*4">…</O>
         <O N="Girder6" T="Group" X="64*5" Y="-99*5">…</O>
         <O N="Girder7" T="Group" X="64*6" Y="-99*6">…</O>
      </O>
      <O N="Bracing" T="Group">…</O>
   </O>
   <O N="SubStructure" T="Group">
      <O N="RCPierCap" T="Group">…</O>
      <O N="RCColumn" T="Group">…</O>
   </O>
</O>
```

Figure 2.13 High-level structure of geometric model written in ParamML

For data mapping from CSiBridge's data schema to the BrIM data schema, the finite element model is first exported to an Excel Spreadsheet file using CSiBridge's "export" function to parse the data. The exported file consists of over 40 tables, each of which contains a number of tuples in a tabular format. Here, each table is related to an object type in the BrIM data schema, whereas each tuple contains data that can be represented with an instance of the object type. Figure 2.16, for example, shows the data mapping from a tuple to a BrIM object. In this example, the type of the object is `FECoordinateSystem`, which is determined based on the name of the table "Coordinate Systems". The name of

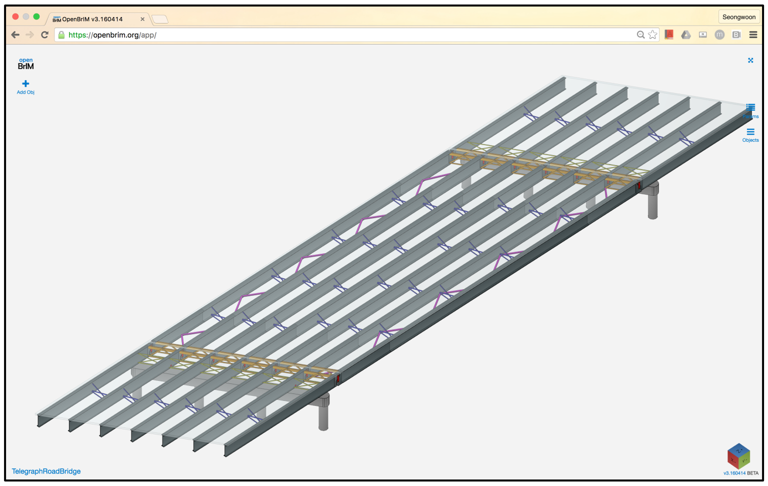


Figure 2.14 Geometric model visualized using OpenBrIM Viewer

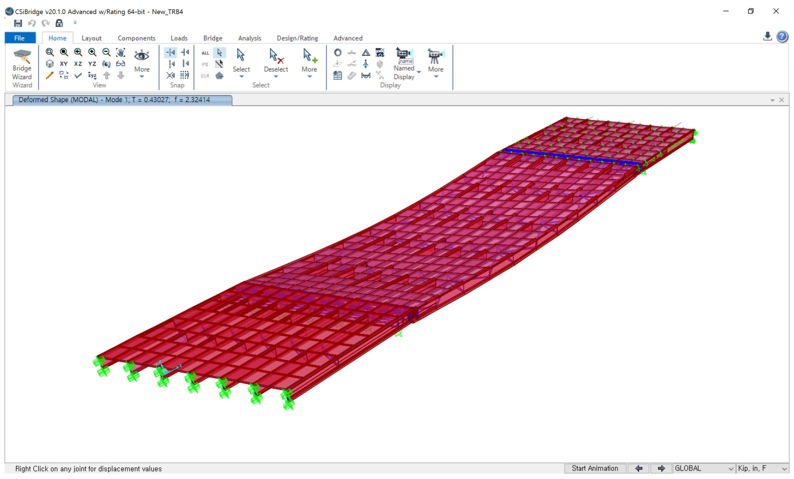
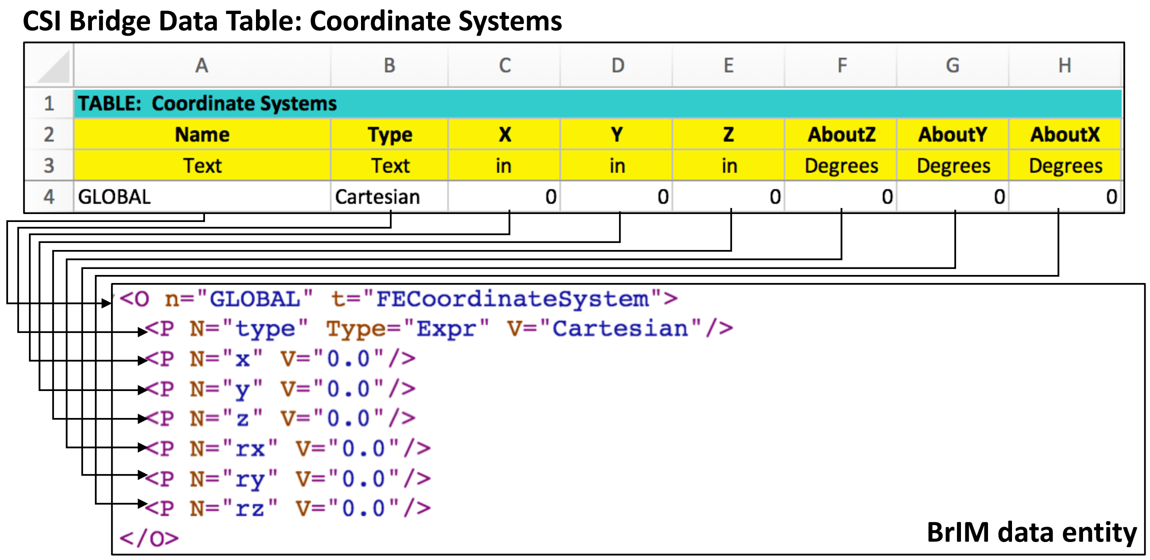

CSiBridge v20.1.0 Advanced w/Rating 64-bit - New_TRB4
File
Home
Layout
Components
Loads
Bridge
Analysis
Design/Rating
Advanced
Bridge Wizard
View
Snap
Select
Deselect
More
Named Display
Display
Deformed Shape (MODAL) - Mode 1; T = 0.43027; f = 2.32414
Right Click on any joint for displacement values
Start Animation
GLOBAL
Kip, in, F

Figure 2.15 TRB's FE model created using CSiBridge (1st natural frequency: 2.324 Hz)

the object is GLOBAL which corresponds to the “Name” attribute of the tuple. In addition, the figure shows that the parameters of the BrIM object correspond to the attribute of the tuple.

For convenient data mapping from the CSiBridge’s data schema to the BrIM data schema, an automation script (written in Python) is developed. The script performs data mapping as described in the pseudo code shown in Figure 2.17. This script reads each tuple in each table, determines the corresponding BrIM object type based on the table name, creates (or



	A	B	C	D	E	F	G	H
1	TABLE: Coordinate Systems							
2	Name	Type	X	Y	Z	AboutZ	AboutY	AboutX
3	Text	Text	in	in	in	Degrees	Degrees	Degrees
4	GLOBAL	Cartesian	0	0	0	0	0	0

```
<O n="GLOBAL" t="FECoordinateSystem">
  <P N="type" Type="Expr" V="Cartesian"/>
  <P N="x" V="0.0"/>
  <P N="y" V="0.0"/>
  <P N="z" V="0.0"/>
  <P N="rx" V="0.0"/>
  <P N="ry" V="0.0"/>
  <P N="rz" V="0.0"/>
</O>
```

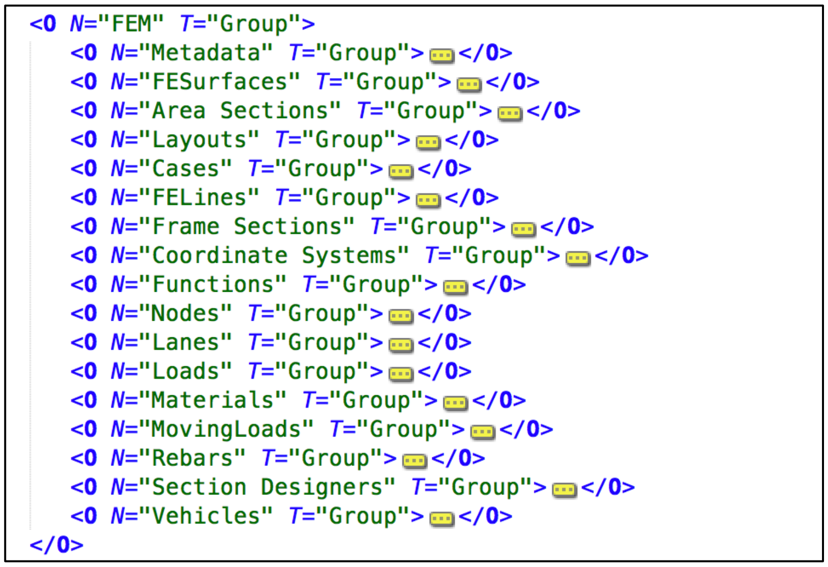
Figure 2.16 Data mapping from Excel Spreadsheet to OpenBrIM object

```
Create an object tree
For each table:
  Determine object type based on the name of the table
  For each tuple:
      Read the name of the tuple and assign it as a name of object
      If there is an object with the same type and name in the object tree:
        Load the object
      Else:
        create a new object with the object type in the object tree
      For each attribute:
        Determine the parameter name based on the attribute name
        Determine the parameter value based on the value of the attribute
        Attach the parameter to the object
```

Figure 2.17 Pseudo code for data mapping from CSiBridge’ data schema to BrIM data schema

load) BrIM object and records the attribute data to the parameter of the BrIM object. The high-level structure of the resulting file is shown in Figure 2.18. In this file, the objects with the same types are grouped together for the convenient data retrieval.

An FE model represented in BrIM data schema can potentially be used by FE analysis tools by mapping BrIM data entities into the FE analysis tools' data entities. As an example, this work develops an automation script (also written in Python) for data mapping from BrIM to CSiBridge. Figure 2.19 shows the pseudo code of the automation script. This script reads each of the objects from the data represented in BrIM data schema and select related tables based on the object type. For each of selected tables, the script determines the parameters relevant to the table and then creates a new tuple containing the name of the object and the selected parameters' data. The created Excel Spreadsheet can be imported by CSiBridge and then be used to perform structural analysis. For example, Figure 2.20 shows the result of the modal analysis using the re-generated finite element model. The result shows that the first natural frequency computed using the re-generated model match with the first natural frequency computed using the original finite element model (see Figure 2.15),

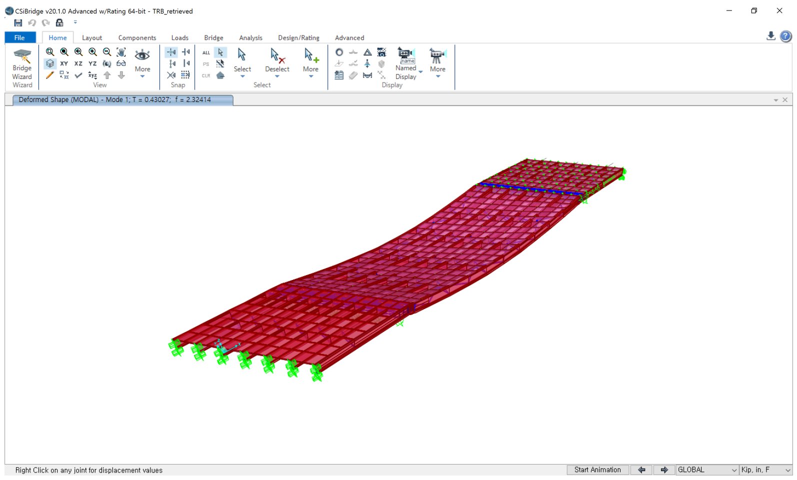


```
<O N="FEM" T="Group">
    <O N="Metadata" T="Group">…</O>
    <O N="FESurfaces" T="Group">…</O>
    <O N="Area Sections" T="Group">…</O>
    <O N="Layouts" T="Group">…</O>
    <O N="Cases" T="Group">…</O>
    <O N="FELines" T="Group">…</O>
    <O N="Frame Sections" T="Group">…</O>
    <O N="Coordinate Systems" T="Group">…</O>
    <O N="Functions" T="Group">…</O>
    <O N="Nodes" T="Group">…</O>
    <O N="Lanes" T="Group">…</O>
    <O N="Loads" T="Group">…</O>
    <O N="Materials" T="Group">…</O>
    <O N="MovingLoads" T="Group">…</O>
    <O N="Rebars" T="Group">…</O>
    <O N="Section Designers" T="Group">…</O>
    <O N="Vehicles" T="Group">…</O>
</O>
```

Figure 2.18 High-level structure of the finite element model represented in BrIM data schema

```
Create a spreadsheet file
For each object:
  Find related tables based on the object type
  For each table:
    Select the parameters relevant to the table
    Write a tuple containing the object name and the selected parameters' data
```

Figure 2.19 Pseudo code for data mapping from BrIM data schema to CSiBridge' data schema



Deformed Shape (MODAL) - Mode 1; T = 0.43027; f = 2.32414

Figure 2.20 Modal analysis with the re-generated finite element model (1st natural frequency: 2.324 Hz)

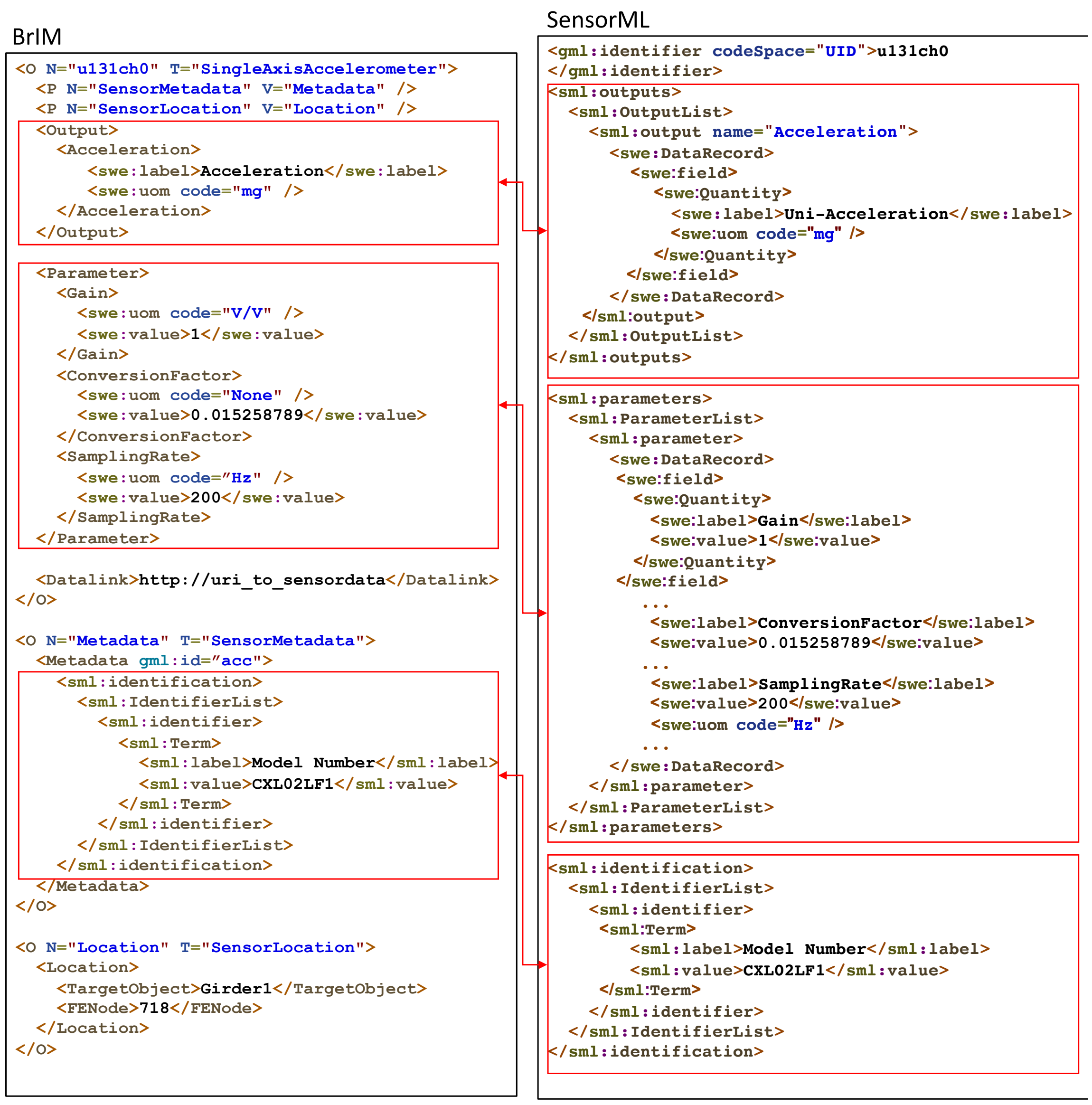
which indicates that the data mapping process is performed without missing data or unintended modification of data.

2.3.3 Sensor description

The proposed BrIM data schema includes object definitions for the representation of sensors. To validate the sensor data schema, the sensor description for the structural monitoring system of the TRB is represented using the BrIM schema. Sensor objects are created based on the sensor information which is recorded in an Excel Spreadsheet file. For example, Figure 2.21 shows a `SingleAxisAccelerometer` object written in ParamML. This object includes information, such as sensor ID (`u131ch0`), output type (`Acceleration`) and parameters (`Gain`, `ConversionFactor`, `Sampling rate`).

In addition, the object includes `SensorMetadata` object, as well as `SensorLocation` object.

The BrIM data schema can describe the relation between sensor objects and other objects (e.g., geometric element and finite element nodes). For example, a sensor object can be



```
<O N="u131ch0" T="SingleAxisAccelerometer">
  <P N="SensorMetadata" V="Metadata" />
  <P N="SensorLocation" V="Location" />
  <Output>
    <Acceleration>
       <swe:label>Acceleration</swe:label>
       <swe:uom code="mg" />
    </Acceleration>
  </Output>

  <Parameter>
    <Gain>
      <swe:uom code="V/V" />
      <swe:value>1</swe:value>
    </Gain>
    <ConversionFactor>
      <swe:uom code="None" />
      <swe:value>0.015258789</swe:value>
    </ConversionFactor>
    <SamplingRate>
      <swe:uom code="Hz" />
      <swe:value>200</swe:value>
    </SamplingRate>
  </Parameter>

  <Datalink>http://uri_to_sensordata</Datalink>
</O>

<O N="Metadata" T="SensorMetadata">
  <Metadata gml:id="acc">
    <sml:identification>
      <sml:IdentifierList>
        <sml:identifier>
          <sml:Term>
            <sml:label>Model Number</sml:label>
            <sml:value>CXL02LF1</sml:value>
          </sml:Term>
        </sml:identifier>
      </sml:IdentifierList>
    </sml:identification>
  </Metadata>
</O>

<O N="Location" T="SensorLocation">
  <Location>
    <TargetObject>Girder1</TargetObject>
    <FENode>718</FENode>
  </Location>
</O>
```

```
<gml:identifier codeSpace="UID">u131ch0
</gml:identifier>
<sml:outputs>
  <sml:OutputList>
    <sml:output name="Acceleration">
      <swe:DataRecord>
        <swe:field>
          <swe:Quantity>
            <swe:label>Uni-Acceleration</swe:label>
             <swe:uom code="mg" />
          </swe:Quantity>
        </swe:field>
      </swe:DataRecord>
     </sml:output>
  </sml:OutputList>
</sml:outputs>

<sml:parameters>
  <sml:ParameterList>
    <sml:parameter>
      <swe:DataRecord>
       <swe:field>
         <swe:Quantity>
           <swe:label>Gain</swe:label>
           <swe:value>1</swe:value>
         </swe:Quantity>
       </swe:field>
          ...
           <swe:label>ConversionFactor</swe:label>
           <swe:value>0.015258789</swe:value>
          ...
           <swe:label>SamplingRate</swe:label>
           <swe:value>200</swe:value>
           <swe:uom code="Hz" />
          ...
      </swe:DataRecord>
    </sml:parameter>
  </sml:ParameterList>
</sml:parameters>

<sml:identification>
  <sml:IdentifierList>
    <sml:identifier>
     <sml:Term>
        <sml:label>Model Number</sml:label>
        <sml:value>CXL02LF1</sml:value>
     </sml:Term>
    </sml:identifier>
  </sml:IdentifierList>
</sml:identification>
```

Figure 2.21 Sensor description represented using BrIM data schema and SensorML data schema

assigned as a child object of a geometric object, which indicates that the sensor object is instrumented on that geometric object. This relation can also be described using the `TargetObject` data in the `SensorLocation` object of the sensor object. Furthermore, the `SensorLocation` object can include the ID of an `FENode` object to indicate that the sensor's location corresponds to the `FENode` in the finite element model.

Since the BrIM data schema is defined based on SensorML, the sensor description in BrIM data schema can be mapped to SensorML data schema. Figure 2.21 shows that the `SingleAxisAccelerometer` object represented in BrIM data schema can be mapped to a SensorML instance. Therefore, the sensor description represented in the BrIM data schema can be parsed, mapped, read and utilized by applications that support SensorML standard.

2.4 Summary

This chapter presents an information modeling framework to facilitate data interoperability and integration for bridge monitoring applications. Bridge monitoring involves a wide variety of information collected from different data sources, including geometric modeling and engineering analysis tools, bridge management systems (BMS) and structural health monitoring (SHM) systems. While the different types of information are related to each other, current practice of bridge management typically handles them using isolated systems, followed by inefficient and error-prone manual data conversion. Using the OpenBrIM data schema [81] as the base model, the proposed BrIM data schema is designed to capture the engineering model and sensor description by examining relevant software tools and data modeling standards. Specifically, data entities for the representation of finite element models are defined based on the data entities of CSiBridge [79] a structural analysis software tool. In addition, data entities for sensor description are defined based on the data entities of SensorML [80], an open standard for sensor description.

The proposed BrIM data schema is demonstrated using bridge information, including geometry, finite element model and sensor information, of the Telegraph Road Bridge (Monroe, MI). The 3-dimensional geometric model of the TRB is created using the OpenBrIM's original object definitions. Data mapping between the CSiBridge native data schema and the proposed BrIM data schema is demonstrated by developing automated mapping scripts. The demonstration also describes the representation of sensors using the proposed BrIM data schema. The results show that the proposed BrIM data schema can effectively represent various types of information involved in bridge monitoring and their relationship.

As a research prototype, the BrIM data schema considers a limited number of standards and applications. For practical use of the BrIM, its data schema needs to be extended to support different standards and applications. Furthermore, the proposed BrIM schema does not consider inspection information which is critical in decision making process for bridge management. The future work will need to investigate additional data sources, such as bridge inspection data, bridge management system (BMS) data, etc., involved in bridge monitoring and management and extend the BrIM data schema accordingly.

Chapter 3

A NoSQL-based Scalable Data Management Framework for Civil Infrastructure Monitoring

3.1 Introduction

As sensor technologies mature, there have been increasing interests in the deployment of sensors for the structural health monitoring (SHM) of large-scale civil infrastructures. The advent of wireless sensor technologies has led to significant reduction in the installation cost of sensor network on civil infrastructures [1]. Many civil infrastructures are now instrumented with dense sensor network to collect valuable information for management purposes [12, 92]. With the permanent installation of sensors, recent research efforts have been attempted to extract statistically meaningful information and to apply data-driven predictive analysis with the collected long-term sensor data [4, 93]. Furthermore, the developments of advanced nondestructive evaluation technologies have facilitated the assessment of the integrity and health of a structure by enabling the detection of the onset of damages [94]. Advances in sensor technologies and increasing deployment of sensors result in the tremendous amount of data that needs to be handled in civil infrastructure monitoring systems. While current SHM research continues to develop and explore new sensor technologies, very little efforts have been spent to investigate proper data management tools. The data issues are of fundamental importance that need to be dealt with before sensing technologies can truly find useful for civil infrastructure lifecycle assessment and management.

Selecting an appropriate database tool for specific application is key to successful deployment of a data management system. Different database tools have different strength and properties. Given the potentially enormous quantity and diversity of sensing data and complexity of civil infrastructure model, it would be desirable that the database tools employed for civil infrastructure monitoring and management system are highly scalable and flexible. Traditional relational database management systems (RDBMS) have the strict table-type data structure and explicit relationships defined among the data. Recent studies have shown that RDBMS do not perform well when dealing with large volume of unstructured data [46, 95]. NoSQL database systems, which are highly scalable and support flexible data schema, have been proposed as an alternative to RDBMS [47, 96]. It has been reported that NoSQL database system can achieve better performance than RDBMS in terms of scalability, flexibility, and low latency by relaxing the rigid data consistency and strict data schema definition of RDBMS [46, 47]. This chapter thus focus on the use of NoSQL database system for data management in civil infrastructure monitoring [71, 72]. The described NoSQL-based data management framework is designed to support not only the scalability to manage a large amount of sensor data, but also the flexibility to manage semi- and un-structured civil infrastructure information.

This chapter is organized as follows. Section 3.2 investigates the data management requirements in civil infrastructure monitoring and describes the selection of NoSQL database tools based on the requirements. Section 3.3 presents a NoSQL-based data management framework and describes BrIM-based database schema that supports the management of various data, including sensor data, image data, sensor information, geometry and engineering model, involved in civil infrastructure monitoring. Section 3.4 demonstrates the NoSQL-based data management through simple data store and retrieval examples, as well as complex data integration examples based on the information model and the monitoring data of the Telegraph Road Bridge (TRB) in Monroe, Michigan. This chapter is concluded with a summary in Section 3.5.

3.2 Selection of NoSQL Data Management Tools

This section discusses the data management requirement of civil infrastructure monitoring systems and the selection of the data management tools [71, 72]. There exist many NoSQL database systems, each has its own pros and cons. The successful development of a data management system depends on the understanding of target systems' requirements, as well as choosing appropriate NoSQL database systems satisfying the requirements. This section describes the overall organization of a typical civil infrastructure monitoring system and discusses the data management requirement of each component in a civil infrastructure monitoring system. NoSQL database tools are then selected based on the defined requirements.

3.2.1 Data management requirements in civil infrastructure monitoring system

Figure 3.1 depicts an overview of a typical data management system structure for SHM systems [6, 72, 97]. In this figure, the shaded boxes refer to the components of the data management system, whereas the arrows describe the data flow. The data management system consists of four main components: (1) onsite computers, (2) main (data repository) server, (3) local (desktop) computer and (4) end-user devices. The roles and data management requirements of the four components are as follows:

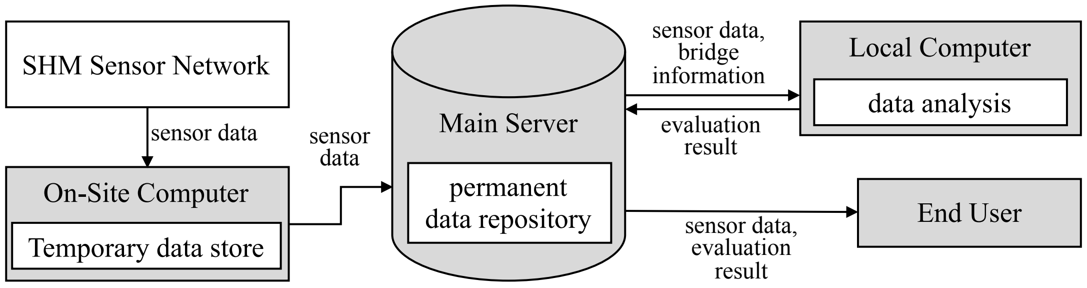


Figure 3.1 Overall structure of data management system for civil infrastructure monitoring

- An onsite computer, which is an autonomous *in situ* computing system, serves the role of an intermediary between the sensor network and the main server for data acquisition (DAQ). An onsite computer interacts with DAQ as follows. First, the onsite computer sends messages to the sensor nodes via local communication network to begin a sampling process. Next, once the sampling process has finished by the sensor nodes, the onsite computer receives the collected data from the sensor nodes and temporarily stores the data in its storage (e.g., a file system or a database system). If necessary, the onsite computer pre-processes and converts the sampled signals into physical values or performs analysis for extracting meaningful information from the collected data. Finally, the onsite computer then transmits the collected data to the main server through a communication network, such as the Internet. Since onsite computers only need to store a limited amount of data temporarily, onsite computers can use a file system (particularly, when the computing capacity of an onsite computer is limited) or a lightweight database system to support query for analysis and pre-processing of the data.
- The main server, which can be implemented on private servers or cloud computing environments, plays a pivotal role for the data management in civil infrastructure monitoring system. Specifically, the main server stores and manages heterogeneous data, including sensor data, sensor information, geometric models, engineering models and analysis results. Furthermore, the main server allows other components (e.g., onsite computer, local computers and end-user

devices) to access the database via communication network (e.g., the Internet and local area network) and to store and retrieve data using database query languages. Given the voluminous and increasing amount of sensor data and semi- and un-structured information, the main server will potentially handle significant amount of data records, which are not necessarily homogeneous or of the same data types. Therefore, the backend database for the main server needs to be highly flexible and scalable to allow long term data archival and extendibility.

- A local (desktop) computer serves as a computing platform that engineers employ to conduct computational tasks (e.g., structural analysis and data-driven analysis) involved in civil infrastructure monitoring and management. For example, the local computer may periodically retrieve sensor data and relevant engineering model from the main server and performs analysis to determine the integrity of the target structure. The local computer can also store some of the data (e.g., recently collected sensor data) in a local database system to prevent unnecessarily repetitive data retrieval from the main server and to enable efficient local data query. Upon finishing data analysis, the local computer sends the analysis results back to the main server to share the results with other components and project participants. Since a local computer needs to store a limited amount of data temporarily, the focuses of the database system for a local desktop computer are not necessarily related to the long-term archiving of large amount of data but should be on efficient data retrieval to support data parsing and analysis.
- An end-user device, such as a laptop computer or a personal device, allow users direct, real-time and ubiquitous access to the data residing in the main server via web and mobile applications. Furthermore, with the development of appropriate interfaces, the end-user devices can access computational tools (e.g., local computer) to remotely conduct engineering analysis. Since an end-user device focuses mainly on information retrieval, no database system is necessary.

3.2.2 Selection of NoSQL database tools

There have been a number of NoSQL database systems with different features and properties. Selecting an appropriate database tool for specific application is very important for successful deployment of data management system [46]. Based on the data model, current NoSQL database tools can be categorized into column family stores, document-oriented stores, key-value stores, and graph databases [46, 47, 48], as shown in Figure 3.2. The features and properties of the four categories of NoSQL database systems can be summarized as follows:

- Column family stores (or wide column stores), which are originated from Google Bigtable [98], aim to handle a large amount of data in a distributed manner. The column family data model has ability to handle a large number of dynamic columns (e.g., billions of columns), which enables flexible data schema.
- Document-oriented stores (or document stores) offer schema-free data model in which different documents (i.e., unit data entities) may have different set of key-value pairs without having a pre-defined data schema. Furthermore, document-oriented database systems typically support convenient queries for heterogeneous data and hierarchical data.

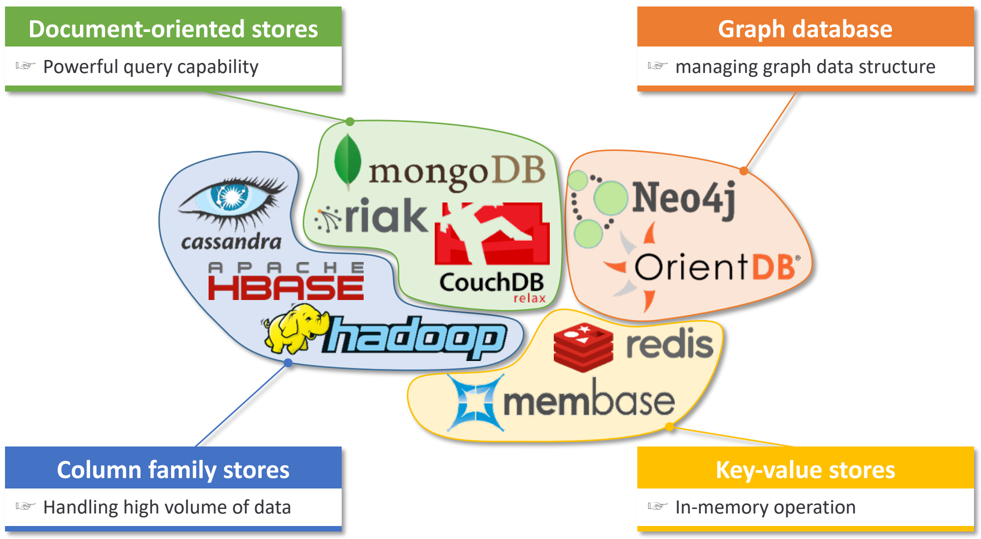


Figure 3.2 Four categories of NoSQL database systems

- Key-value stores offer simple data model that consists of key and value, where the value is opaque to system and cannot be used for data query (i.e., only the key-based query is supported). Based on a simple data model, as well as in-memory operation, key-value stores guarantees fast read and write performance.
- Graph databases use graph structure consisting of nodes and edges to represent the data. Based on the graph structure, graph databases are optimized to manage data records with complex relationships. Furthermore, graph databases enable efficient query performance, in particular, for recursive join operations involving complex relationship among data entities.

In this study, we employ Apache Cassandra [99], one of the most widely used column family stores, to satisfy the scalability and flexibility requirements of the main server, and MongoDB [100], one of the most widely used document-oriented stores, for convenient and efficient queries on the onsite computer and the local computer. Key-value stores, while suitable for fast data store and retrieval, are ruled out in this study, because of their limitation in terms of the data capacity. Lastly, the data schemas, to be described in the latter section, do not lend themselves suitable for the graph database.

3.2.2.1 Apache Cassandra: column family database for supporting persistent data archiving

Given the scalability requirements of the main server, Apache Cassandra database [99], which was originally developed to meet the reliability and scalability needs of Facebook [101], is selected. Cassandra database is designed to handle a very large volume of data based on a peer-to-peer (P2P) architecture which is a preferable approach for a highly available and scalable distributed database [45]. Specifically, in the P2P architecture, each node (i.e., a database instance) is self-sufficient and all nodes have an identical role, which ensures that, in the worst-case scenario, the failure of some nodes results in degradation of the database operation but remains able to guarantee a high possibility of availability. Furthermore, the P2P architecture ensures high scalability in that the number of nodes can be easily modified without causing operational downtime and the database performance is linearly scaled as new nodes are added to an existing Cassandra database cluster.

To maintain the consistency of the database and to process requests in a decentralized manner, nodes in a Cassandra database cluster communicate among one another according to a "ring" topology as shown in Figure 3.3. Data is replicated and distributed over multiple nodes to ensure high availability and fault-tolerance, as well as to maintain efficient reading and writing performances. Figure 3.3, for instance, illustrates how sensor measurement data is stored in a Cassandra database cluster. In this example, the incoming data `R` has two records `r1` and `r2` which could be sensor measurement data collected by different sensors. The replication factor (i.e., the number of replicas in a cluster) is two. Any node (say, node `N5` in the example) can accept the write request. The incoming sensor data is partitioned into two pieces and then copied twice over the nodes. Since the sensor data is replicated over the cluster, writing and reading the data can still be performed even when a node is down, as long as other nodes remain available for processing the requests.

Another advantage of using the Cassandra database is the flexible data schema. Figure 3.4 depicts the Cassandra database's data structure consisting of "keyspace," "column family,"

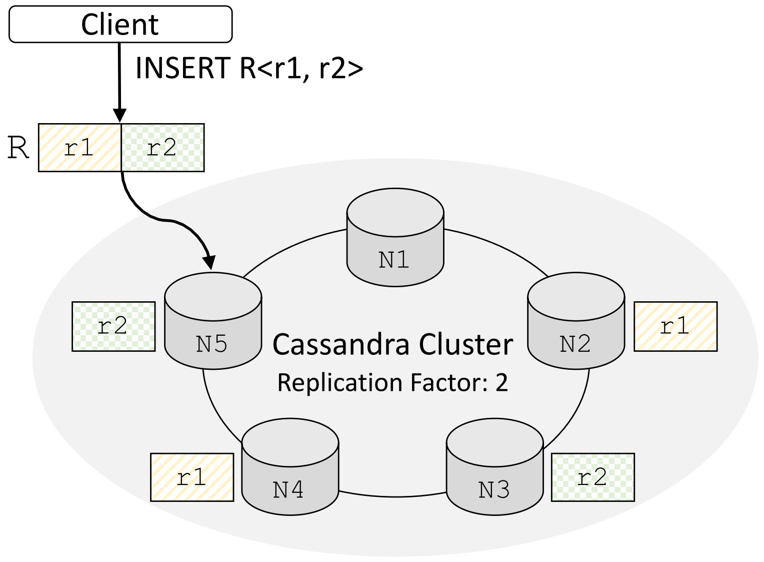


Figure 3.3 Ring topology of Cassandra database

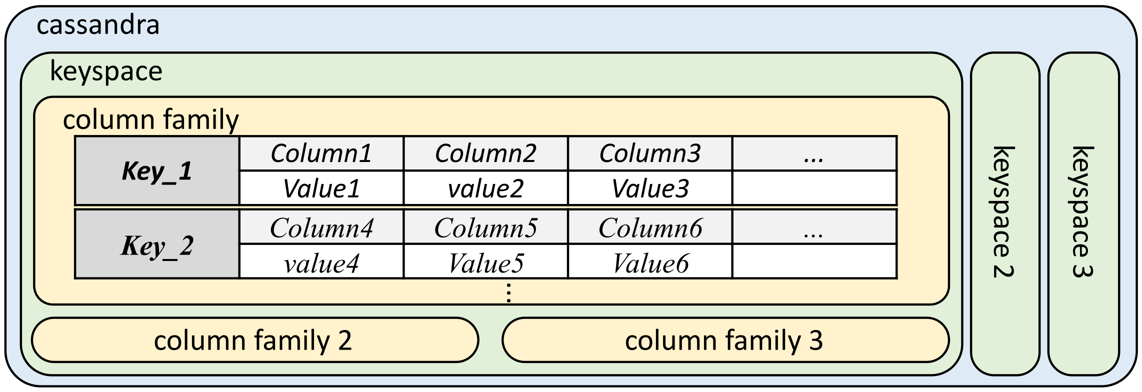


Figure 3.4 Data structure of Cassandra database

"row" and "column," which are analogous to "database," "table," "tuple" and "attribute" of relational database, respectively. The top-level keyspace is defined for a specific project. The column family consists an array of rows where each row consists of a set of columns. Each column represents a basic element in the Cassandra database and is assigned a name and value pair. One of the most significant differences from relational databases is that the column family stores allow different rows to store different combinations of columns. The flexible data structure has advantages on storing semi-structured data (e.g., bridge information models) by allowing different attribute sets for different records. Furthermore, Cassandra's dynamic column feature, which allows new records to be attached at the end of an existing row, and supports effective query performance for time-series data by storing contiguous time-series data in contiguous disk locations [60]. Last but not least, Cassandra database supports a variety of data types (e.g., number, array, dictionary, binary data, etc.), which can be an advantage for SHM data management which often involves time-series sensor data and video image data.

3.2.2.2 MongoDB: document store for supporting efficient data retrieval

MongoDB features schema-free data structure and powerful query capability. The data structure of MongoDB consists of the database, collection, and binary JSON (BSON) schema-less document, as shown in Figure 3.5 [102]. The JSON document enables easy change or extension of the data model and human-understandable data structure such as object-oriented data format. MongoDB also has the advantage of representing complex data structure by enabling relationships between documents and supporting hierarchical data structure. Moreover, MongoDB offers fast read and write performance [102]. Although MongoDB does not support the "join" query, it still supports a rich set of query operations including indexing, range query, and aggregation operations. With the flexible schema and high performance, MongoDB is particularly suitable when complex SQL-like queries and transactions are not required.

Based on its flexibility and speed, MongoDB has been widely used in many fields including Internet of Things (IoT) applications and real-time analysis [102, 103]. In the proposed framework, MongoDB is employed for onsite computers and local (desktop) computers to support temporary data store with convenient and efficient queries.

3.3 A NoSQL-based Data Management Framework for Civil Infrastructure Monitoring

This section describes the data management framework for civil infrastructure monitoring based on NoSQL database tools [71]. Specifically, data schema definition for NoSQL

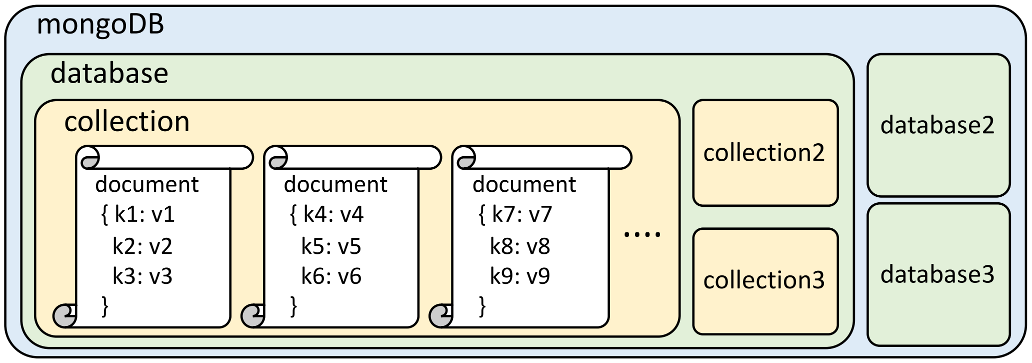


Figure 3.5 Data structure of MongoDB

database systems and data store and retrieval processes are discussed. The design and implementation of the framework is described with a bridge monitoring system as an example. Figure 3.6 depicts the overall data management framework employing NoSQL database systems. The onsite computer and the local computer employ MongoDB for convenient and efficient querying, while the main server employs Cassandra database for the long-term archiving of a large amount of data. To facilitate system automation and improve data management efficiency, data schemas are defined considering the data requirements in civil infrastructure monitoring. Data involved in civil infrastructure monitoring applications can be divided into sensor data (typically time-series data) and civil infrastructure information (typically object-oriented data). Sensor data schema is defined in a way that can facilitate range query, whereas civil infrastructure information

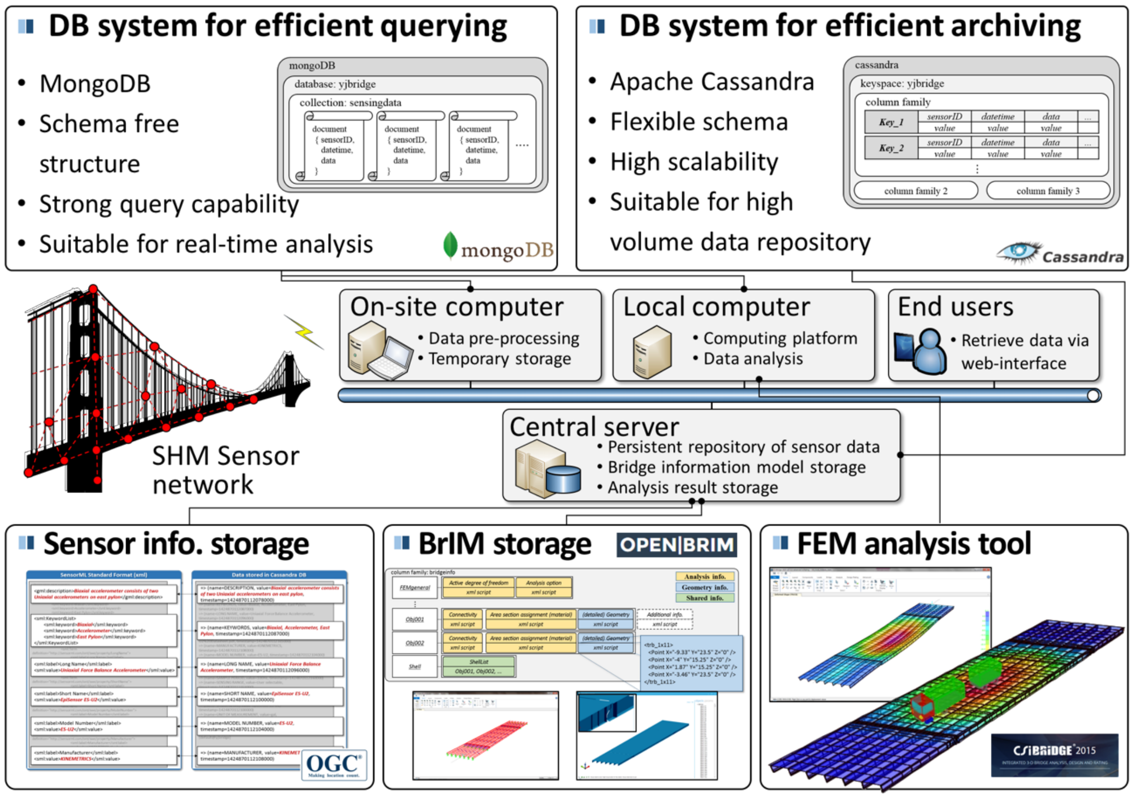


Figure 3.6 Data management framework for civil infrastructure monitoring based on NoSQL database

(including geometry, finite element model and sensor information) schema is defined based on the BrIM schema, as discussed in Chapter 2, to facilitate data interoperability.

3.3.1 Onsite computer

An onsite computer receives sensor measurement data from sensor networks, stores the data temporarily, and sends the data to the main server for permanent archiving. In case when the onsite computer has enough computational power to run a local database and is required to support data queries for data analyses, a document database can be employed to support efficient querying. For this study, a version of MongoDB (version 2.0.6) is employed because some onsite computers in practice use older versions of the Microsoft operating systems (OSs) which is not compatible with recent versions (i.e., version 2.2 or higher) of the MongoDB.

3.3.1.1 Data scheme for document database on onsite computer

In the current design of the data management framework, an onsite computer stores sensor data and sensor information but not bridge information, because an onsite computer performs typically simple data analyses which do not involve bridge information. In MongoDB, a database, which is equivalent to the database of RDBMS, can be created (if not already exists) and selected using MongoDB's `use` command as follows:

```
use mybridgedb
```

which creates a new database named `mybridgedb` (if not already exists) and accesses to the database. A collection, which is equivalent to the database and table of RDBMS, in MongoDB can be created without defining data schema using MongoDB's `createCollection` command as follows:

```
db.createCollection("sensor_data")
```

which creates a new collection named `sensor_data` in the current database. Data schema is created dynamically as a new document, which is equivalent to a tuple of RDBMS, is inserted to a collection. It should be noted that documents in a collection do not need to have the same data schema; they can have different set of key-value pairs.

Figure 3.7 shows the data schema of sensor data collection, named `sensor_data`, in MongoDB. Each document stores sensor data in a list along with the `sensor_id` and `event_time` of data acquisition. In the current implementation, each document collects a list of measured data over a period of a second. For example, as shown in Figure 3.7, if the sampling rate is 5Hz, the sampled data is divided into buckets; each bucket has five consecutive data and is stored in a single document. Since a document in MongoDB can have up to 16MB of storage [100], the strategy to partition the data according to sampling rate and the data storage is necessary to prevent data overflow which can be caused by sensors that have high sampling rate.

In addition to sensor data collection, data schema for sensor information collections are defined as shown in Figure 3.8. The schema takes advantage of MongoDB's hierarchical data structure to categorize sensors for ease of sensor information retrieval. For example,

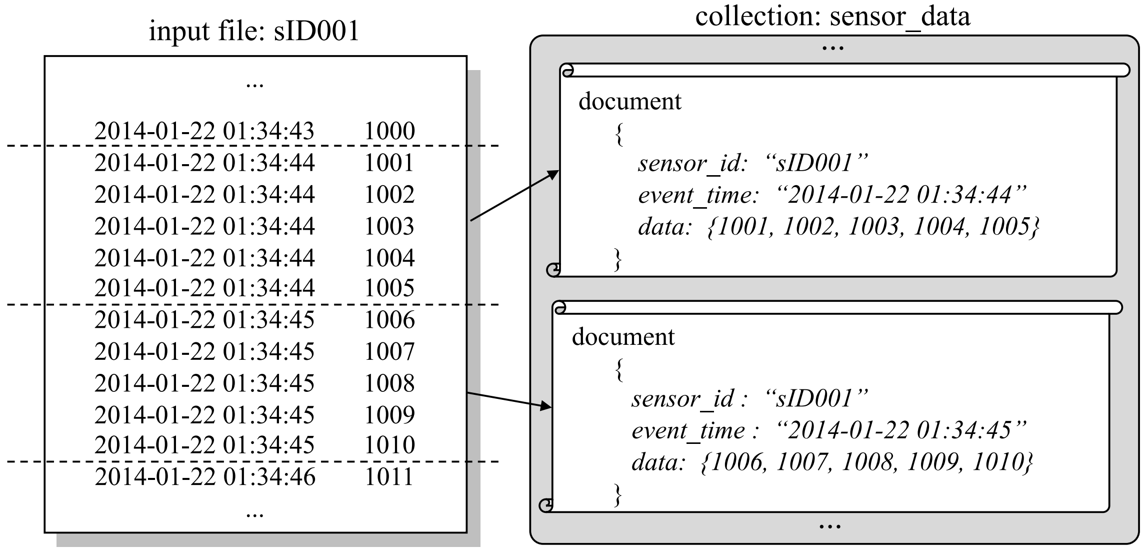


Figure 3.7 Data schema of sensor data in MongoDB

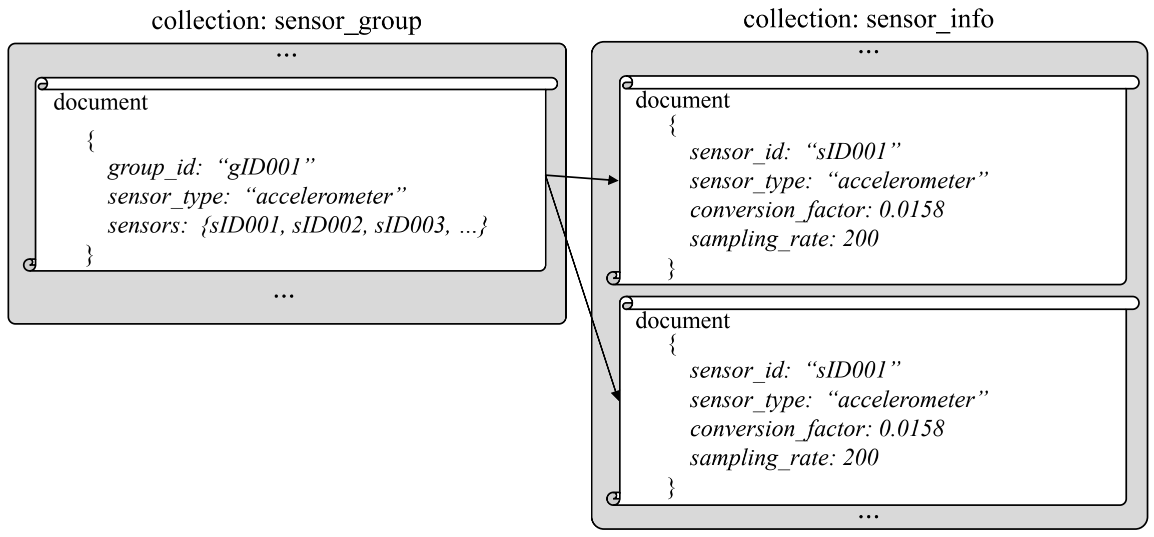

collection: sensor_group
document
{
group_id: "gID001"
sensor_type: "accelerometer"
sensors: {sID001, sID002, sID003, ...}
}
collection: sensor_info
document
{
sensor_id: "sID001"
sensor_type: "accelerometer"
conversion_factor: 0.0158
sampling_rate: 200
}
document
{
sensor_id: "sID001"
sensor_type: "accelerometer"
conversion_factor: 0.0158
sampling_rate: 200
}

Figure 3.8 Data schema of sensor information in MongoDB

the `sensor_group` collection is defined to store documents containing the list of `sensor_ids` of certain types of sensors (e.g., accelerometer, strain gauge and thermistor). In addition, the `sensor_info` collection is defined to store documents containing sensor metadata (e.g., `sensor_id`, `sensor_type`, `conversion_factor`, `sampling_rate`, etc.). With the schema definition, data store to and data retrieval from the onsite computer can be performed with MongoDB's query language.

3.3.1.2 Data store and retrieval processes on onsite computer

An onsite computer needs to not only store incoming sensor data stream, but also allow data retrieval for data analyses and for uploading data to the main server. MongoDB offers data manipulation language (DML) to support data store and retrieval operations. Furthermore, MongoDB supports application programming interfaces (API) in different scripting languages. After selecting a database using the `use` statement, data store process can be performed using MongoDB's `save` command. For example, a sensor data store process can be performed using the save command as follows:

```
db.sensor_data.save({sensor_id: "sID001", event_time:
ISODate("2014-01-22T01:34:44"), data: [1001, 1002, 1003,
1004, 1005]})
```

where the document (i.e., the contents inside the curly brackets) containing the `sensor_id` (i.e., `sID001`), `event_time` (i.e., `2014-01-22T01:34:44`) and the data (i.e., `[1001, 1002, 1003, 1004, 1005]`) is stored to the collection `sensor_data`.

Similarly, the data retrieval process can be performed using another MongoDB command `find`. For example, application programs, such as data analysis module and data transmission module, can retrieve sensor data collected from certain time period by a certain sensor as follows:

```
db.sensor_data.find({sensor_id: "sID001", event_time:
{$gte: ISODate("2014-01-22"), $lte: ISODate("2014-01-
23")}})
```

which retrieves sensor data collected by sensor `sID001` during the time period from `2014-01-22` to `2014-01-23` from the collection `sensor_data`. MongoDB returns all corresponding documents according to the specified data schema as follows:

```
{sensor_id: "sID001", event_time: ISODate("2014-01-
22T01:34:44"), data: [1001, 1002, 1003, 1004, 1005]}
```

The module that runs the `find` operation can use the retrieved document similar to using a JSON document.

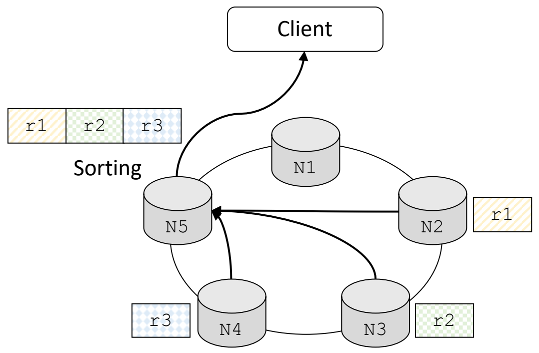
3.3.2 Main server

The main server serves as a central data repository of the bridge monitoring system. Apache Cassandra database is implemented as the backend database for the main server to support efficient long-term archiving of data. Cassandra database in the main server is designed to store data of different types, including sensor data, image data, sensor information, bridge information model (including geometry and engineering model), and analysis result. Cassandra database offers Cassandra Query Language (CQL) [104] which includes DDL and DML. Using CQL and Cassandra Driver APIs [105], client systems, such as the onsite computer, local computer and user devices, can access the Cassandra database to store and retrieve data. Since the size of sensor data is usually quite large, it is desirable to tune the Cassandra database to use as much as memory as possible for efficient data processing.

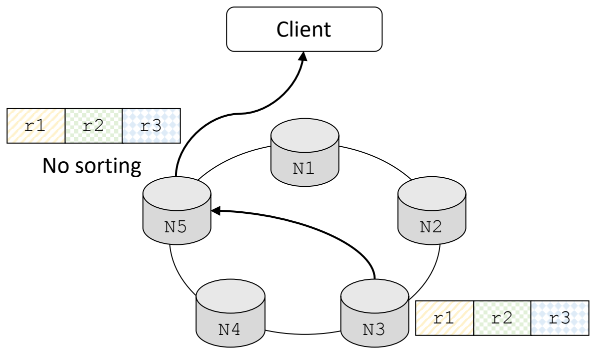
3.3.2.1 Data scheme for column family database on main server

3.3.2.1.1 Sensor data schema

In the proposed data management framework, Cassandra database stores a large volume of sensor data. Since bridge monitoring often utilize continuous time-series data collected within a certain period, efficient range query performance for time-series data needs to be supported. The partitioning feature of Cassandra, however, may result in distribution of the time-series data to many different database nodes, resulting in excessive disk seek time. Figure 3.9(a) shows an example where the continuous time-series data `r1`, `r2` and `r3` are



(a) Time-series data distributed over multiple nodes



(b) Time-series data stored in the same node in sorted order

Figure 3.9 Time-series data stored in a distributed system

distributed over multiple database nodes. In this case, a database query not only needs to retrieve the data from all three nodes, but also needs to sort the data according to their timestamps, which ends up with poor performance for retrieving time-series data.

To enable effective range query performance, a dynamic column strategy for handling time-series data in Cassandra database is employed, as shown in Figure 3.10. The basic idea of this strategy is to store time-series data contiguously in a sequentially sorted order to minimize disk seek time. The Cassandra data schema for time-series sensor data is defined as follows. A column family called `sensor_data` is created for sensor data store using `CREATE` query statement of CQL as follows:

```
CREATE TABLE IF NOT EXISTS sensor_data
(sensor_id TEXT,
 year TEXT,
 event_time TIMESTAMP,
 data LIST<DOUBLE>,
 PRIMARY KEY ((sensor_id, year), event_time));
```

In this schema, the data entities `sensor_id`, `year` and `event_time` compose the primary key of the column family, where `sensor_id` and `year` are the row key and `event_time` is the clustering key. Based on the row key definition, sensor data collected from a sensor is stored in a single row. Furthermore, sensor data is stored in a sorted order by assigning the `event_time` of the data as a clustering key. Since sensor data collected from civil infrastructure monitoring systems usually has very high sampling rate with same interval period between data points, it is redundant to record timestamp for every data point. Instead, the proposed data schema encodes sensor data as a numeric array type `list<double>` that stores data collected during a specified time period (e.g., 1 second)

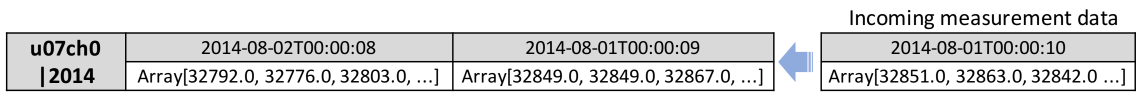

Incoming measurement data
u07ch0 |2014
2014-08-02T00:00:08
Array[32792.0, 32776.0, 32803.0, ...]
2014-08-01T00:00:09
Array[32849.0, 32849.0, 32867.0, ...]
2014-08-01T00:00:10
Array[32851.0, 32863.0, 32842.0 ...]

Figure 3.10 Database schema for time-series numeric sensor data

in a sorted order. The timestamp records can be regenerated, if needed, based on the sampling rate. With the data schema definition, a new incoming data is stored to the existing row corresponding to `sensor_id` by dynamically adding new columns at the end of the row (see Figure 3.10**)**. This data schema improves the range query performance comparing to RDBMSs by enforcing the consecutive time-series sensor data to be stored in a contiguous physical disk location in the same node, as shown in Figure 3.9(b) [60]. In this study, a part of the timestamp (e.g., year) is added to the row key to prevent a single row from becoming too lengthy. In this way, the data from a sensor can be partitioned to several rows based on a specific time period (e.g., year) of data acquisition.

Similarly, a column family called `image_data` is created for image data store using `CREATE` query statement of CQL as follows:

```
CREATE TABLE IF NOT EXISTS image_data
(camera_id TEXT,
 month TEXT,
 event_time TIMESTAMP,
 image BLOB,
 PRIMARY KEY ((camera_id, month), event_time));
```

In this schema, the data entities `camera_id`, `month` and `event_time` compose the primary key of the column family, where `camera_id` and `month` are the row key and `event_time` is the clustering key. Figure 3.11 shows that sequential image files collected from a traffic video camera are stored in a row by assigning event time (e.g., `2016-08-23T10:02:08`) as a clustering key. In addition, part of the timestamp (e.g., year and month) is added to the row key to partition image data to several rows based on the year and month of its acquisition. Each image file is encoded in a binary large object (BLOB)

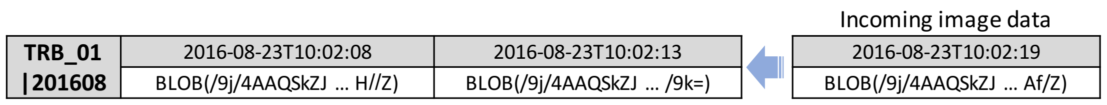


Figure 3.11 Database schema for time-series image data

data (e.g., `/9j/4AAQSkZj … H//Z`) and stored in a single column. The BLOB data can be converted back to the original image file using imaging libraries, such as Python Imaging Library [106].

3.3.2.1.2 Bridge information model schema

One salient feature for adapting a NoSQL database system is the ease of mapping a hierarchical object-oriented bridge information model onto the database schema. An extensible database schema is needed to effectively manage the complex bridge information. Cassandra database offers flexible data structure that can elegantly handle complex data [107]. Using the flexible data structure, a database schema that follows closely the BrIM schema (which was discussed in Section 2) is designed. Figure 3.12, for example, shows data mapping between the BrIM schema of the `FELine` object and the corresponding column family schema `FELine` created using a CQL `CREATE` query statement as follows:

```
CREATE TABLE IF NOT EXISTS FELine
(uid uuid,
 N text,
 T text,
 node1 text,
 node2 text,
 felinesection text,
 felinemesh text,
 felinerelease text,
 parent map<uuid,text>,
 child map<uuid,text>,
 PRIMARY KEY(uid));
```

The database schema contains the data entities of `FELine` object, as well as child and parent entities to record the hierarchical relation between the objects. As such, bridge

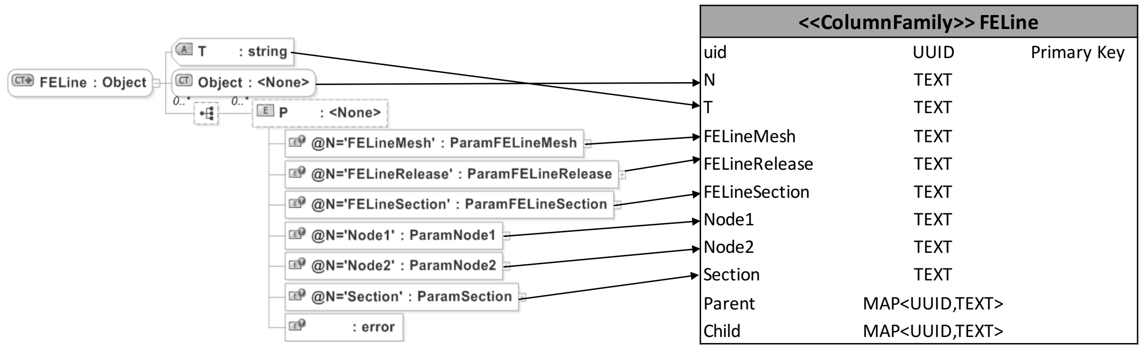
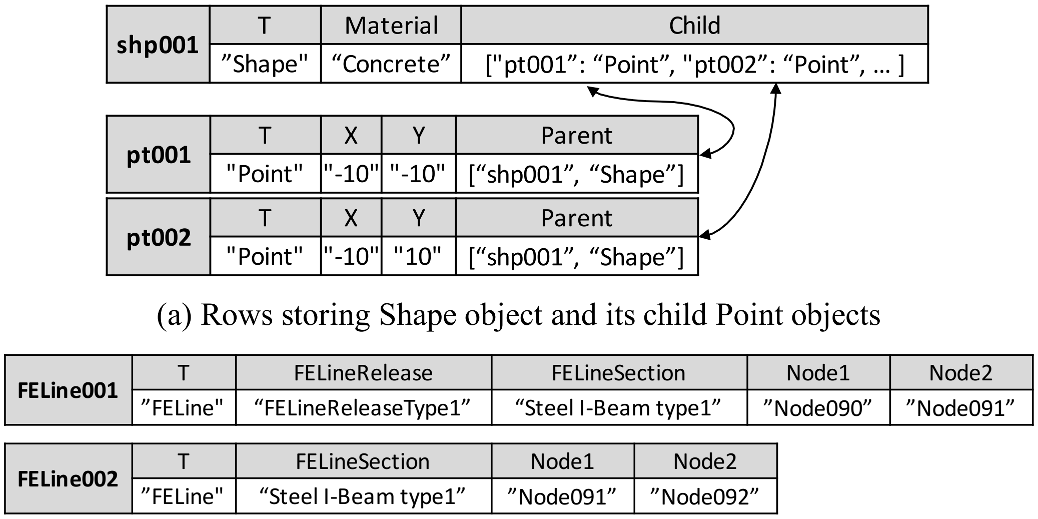


Figure 3.12 Data mapping between BrIM schema FELine and corresponding Cassandra column family

information stored in the column-oriented database can be mapped to hierarchical BrIM objects.

Figure 3.13 shows examples of the rows of the column-oriented database for storing BrIM objects where a single object is stored in a row. Each row has a mandatory partition key (e.g., `shp001` of the first row in Figure 3.13(a)). A row has columns for storing attributes and parameters, as well as the list of child and parent objects. Since the Cassandra database supports collection types, any number of child objects can be recorded in the child column. In Figure 3.13(a), for example, the child column of the shape object contains the ID and types of child objects (i.e., `[pt001: Point`, `pt002: Point`, …`]`). One issue in managing hierarchical object data is that each object may have different sets of attributes. This data irregularity can be efficiently handled by the Cassandra database with its flexible data structure. Specifically, the Cassandra database allows rows in the same column family to contain different sets of columns. For instance, Figure 3.13(b) shows that the two rows in the column family `FELine` have different column sets: the first row has the `FELineRelease` column, while the second one does not. In fact, BrIM objects with the same type often have different sets of attributes and child objects. As such, the flexible data structure of Cassandra database is suitable to handle the heterogeneous BrIM object entities without enforcing every row to have the same set of columns.



shp001	T	Material	Child
	”Shape"	“Concrete”	["pt001”: “Point”, "pt002”: “Point”, …]

pt001	T	X	Y	Parent
	"Point"	"-10"	"-10"	[“shp001”, “Shape”]

pt002	T	X	Y	Parent
	"Point"	"-10"	"10"	[“shp001”, “Shape”]

(a) Rows storing Shape object and its child Point objects

FELine001	T	FELineRelease	FELineSection	Node1	Node2
	”FELine"	“FELineReleaseType1”	“Steel I-Beam type1”	”Node090”	”Node091”

FELine002	T	FELineSection	Node1	Node2
	”FELine"	“Steel I-Beam type1”	”Node091”	”Node092”

(b) Rows storing heterogeneous FELine objects

Figure 3.13 Database schema for BrIM objects

3.3.2.2 Data store and retrieval processes on main server

3.3.2.2.1 Data store and retrieval processes for sensor data

Once the data schema is defined, bridge monitoring data can be stored and retrieved using CQL [104], which is similar to the structural query language (SQL) of relational database. Specifically, CQL uses `INSERT-INTO-VALUE` statement for data insertion and `SELECT-FROM-WHERE` statement for data retrieval. For example, sensor data can be stored to Cassandra database by using an `INSERT` query as follows:

```
INSERT INTO sensor_data (sensor_id, year, event_time,
data)
VALUES ("sID001", "2014", "2014-01-22T01:34:44", [1001,
1002, 1003, 1004, 1005])
```

This query statement stores a row, which has `sensor_id`, `year`, `event_time` and `data` values, to the `sensor_data` column family. Once stored, data can be retrieved using a `SELECT` query. For example, client systems, such as local computers, can retrieve consecutive sensor data using a `SELECT` query as follows:

```
SELECT data
FROM sensor_data
WHERE sensor_id="sID001"
      AND year="2014"
      AND event_time>= "2014-01-22T00:00:00"
      AND event_time<="2014-01-23T00:00:00"
```

which retrieves sensor data collected by sensor `sID001` during the time period from `2014-01-22T00:00:00` to `2014-01-23T00:00:00` from the column family `sensor_data`. 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, it should be noted that CQL has limited query operations 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.

3.3.2.2.2 Data store and retrieval processes for bridge information model

The bridge information described in BrIM data model needs to be mapped to the data schema of the Cassandra database, and vice versa. For this purpose, we develop mapping scripts that maps data between the BrIM schema and the database schema. The scripts is written based on Cassandra driver API [105] for interacting with Cassandra database and an XML parser (such as xml.etree.ElementTree package [108]) for parsing and modifying the information written in XML. Figure 3.14 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

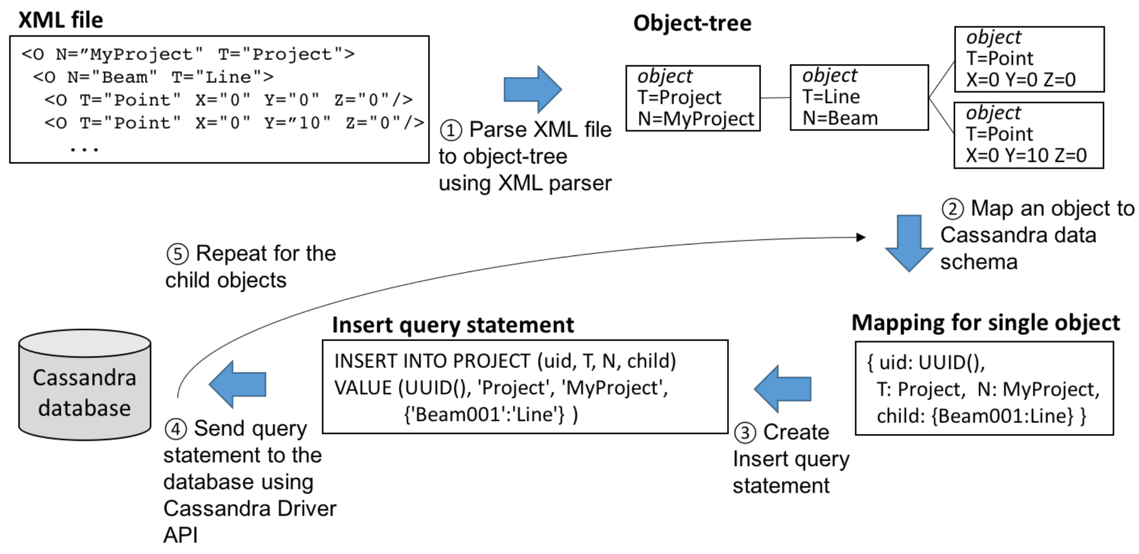

XML file
<O N="MyProject" T="Project">
<O N="Beam" T="Line">
<O T="Point" X="0" Y="0" Z="0"/>
<O T="Point" X="0" Y="10" Z="0"/>
...
① Parse XML file to object-tree using XML parser
Object-tree
object T=Project N=MyProject
object T=Line N=Beam
object T=Point X=0 Y=0 Z=0
object T=Point X=0 Y=10 Z=0
② Map an object to Cassandra data schema
⑤ Repeat for the child objects
Mapping for single object
{ uid: UUID(), T: Project, N: MyProject, child: {Beam001:Line} }
③ Create Insert query statement
Insert query statement
INSERT INTO PROJECT (uid, T, N, child) VALUE (UUID(), 'Project', 'MyProject', {'Beam001':'Line'})
④ Send query statement to the database using Cassandra Driver API
Cassandra database

Figure 3.14 Data mapping from BrIM schema to Cassandra database schema

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 all the 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 3.15 shows an example that retrieves the `Project` object and its child objects, which are stored through the mapping process shown in Figure 3.14. 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.

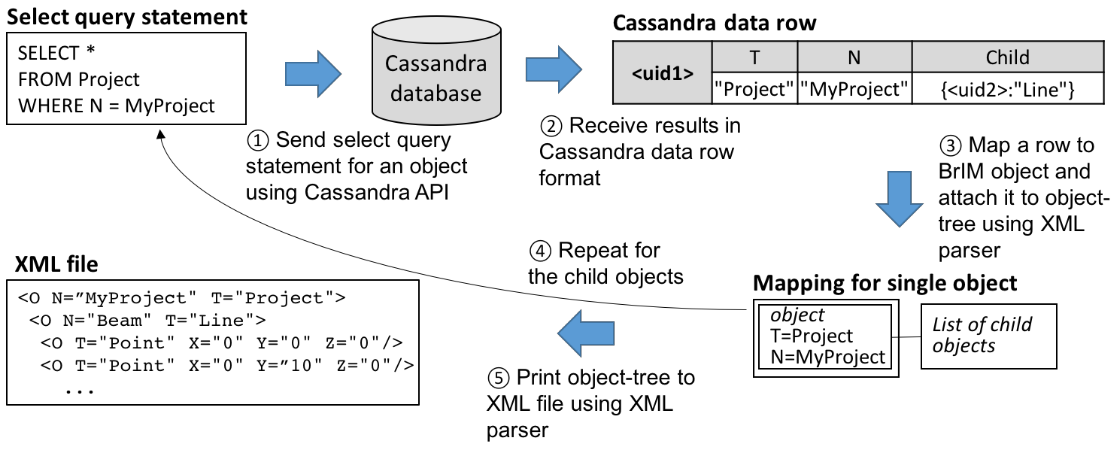

Select query statement
SELECT *
FROM Project
WHERE N = MyProject
Cassandra database
① Send select query statement for an object using Cassandra API
Cassandra data row
<uid1>
T
N
Child
"Project"
"MyProject"
{<uid2>:"Line"}
② Receive results in Cassandra data row format
③ Map a row to BrIM object and attach it to object-tree using XML parser
④ Repeat for the child objects
Mapping for single object
object
T=Project
N=MyProject
List of child objects
⑤ Print object-tree to XML file using XML parser
XML file
<O N="MyProject" T="Project">
<O N="Beam" T="Line">
<O T="Point" X="0" Y="0" Z="0"/>
<O T="Point" X="0" Y="10" Z="0"/>
...

Figure 3.15 Data mapping from Cassandra database schema to BrIM schema

3.3.3 Local computer

A local computer is essentially a desktop-based computing platform that retrieves data from the main server using CQL, performs analysis, and pushes the analysis results back to the main server. Since some data analysis modules require very expensive computational costs, the decentralized strategy helps the main server to be isolated from such operations and to maintain its performance as the central data repository. Nevertheless, it may not be efficient to retrieve sensor data from the main server every time the local computer performs analysis because (1) the sensor data retrieval takes a long time due to the volume of data, and (2) some of the data can possibly be used repeatedly. Instead, the local computer in the current implementation is designed to retrieve sensor data periodically from the main server and to store the data of certain period (e.g., data collected during the last one month) in a local database. This approach can reduce data transmission time as well as traffic between the local computer and the main server. In the current implementation, MongoDB is used as the local database on the local computer. Data schema design follows the same schema designed for the MongoDB on the onsite computer.

Data retrieved from MongoDB on the local computer and from Cassandra database on the main server can be utilized by different analysis modules on the local computer. To interface the database systems and the analysis modules written in scripting languages (e.g., Python), database APIs [105, 109] are used. Furthermore, the analysis modules employ various tools, including MATLAB Engine (MATLAB API for Python [110]), scikit-learn (a package for machine learning in Python [111]) and rpy2 (R API for Python [112]), to connect database systems with different data analysis modules. Demonstrative examples of analysis modules implemented on the local computer will be discussed next in Section 3.4.

3.4 Case Example

This section demonstrates the utilization of the NoSQL-based data management framework [70, 71]. To test the data management framework, this study uses the data sets collected from the Telegraph Road Bridge (TRB). In addition to static bridge information (e.g., geometry, finite element model and sensor description) the structural monitoring system of the TRB collects dynamic sensing data. More specifically,

- Sensor data: The TRB is instrumented with 60 sensors, including 14 accelerometers, 40 strain gauges, and 6 thermistors [93, 113]. Figure 3.16 shows the layout of the sensor network [113]. These sensors collect data every two hours for one-minute duration at sampling frequency 200 Hz (accelerometer) or 100 Hz (strain gauges and thermistors). The sensors acquire data for a one-minute time duration on every two hours interval.
- Traffic video image: The traffic monitoring system operated by Michigan Department of Transportation (MDOT) collects traffic video images at the TRB every two seconds [114].

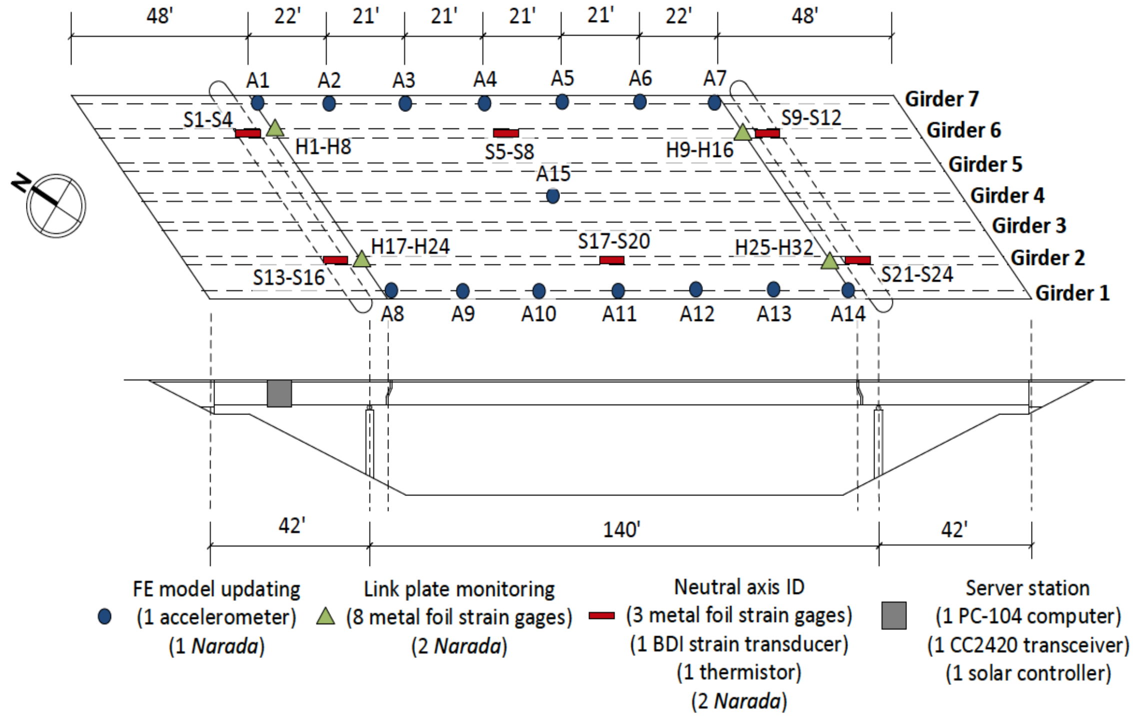

48'
22'
21'
21'
21'
21'
22'
48'
A1
A2
A3
A4
A5
A6
A7
Girder 7
Girder 6
Girder 5
Girder 4
Girder 3
Girder 2
Girder 1
S1-S4
H1-H8
S5-S8
A15
H9-H16
S9-S12
H17-H24
S17-S20
H25-H32
S13-S16
S21-S24
A8
A9
A10
A11
A12
A13
A14
N
42'
140'
42'
FE model updating (1 accelerometer) (1 Narada)
Link plate monitoring (8 metal foil strain gages) (2 Narada)
Neutral axis ID (3 metal foil strain gages) (1 BDI strain transducer) (1 thermistor) (2 Narada)
Server station (1 PC-104 computer) (1 CC2420 transceiver) (1 solar controller)

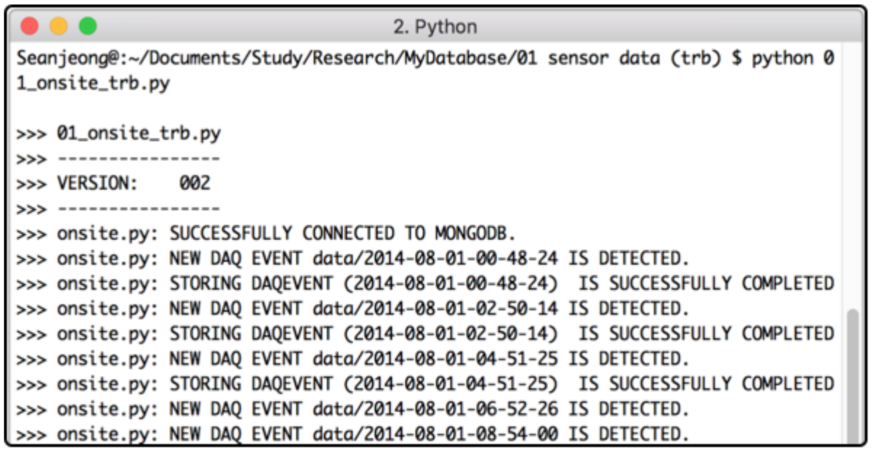
Figure 3.16 Type and location of sensors installed on the Telegraph Road Bridge [110]

For demonstration, a laptop computer (MacBook Pro mid 2014), a server computer (Dell PowerEdge T620) and a desktop computer (A custom desktop computer with Windows 7) are used as an onsite computer, a main server and a local computer, respectively.

3.4.1 Data store and retrieval

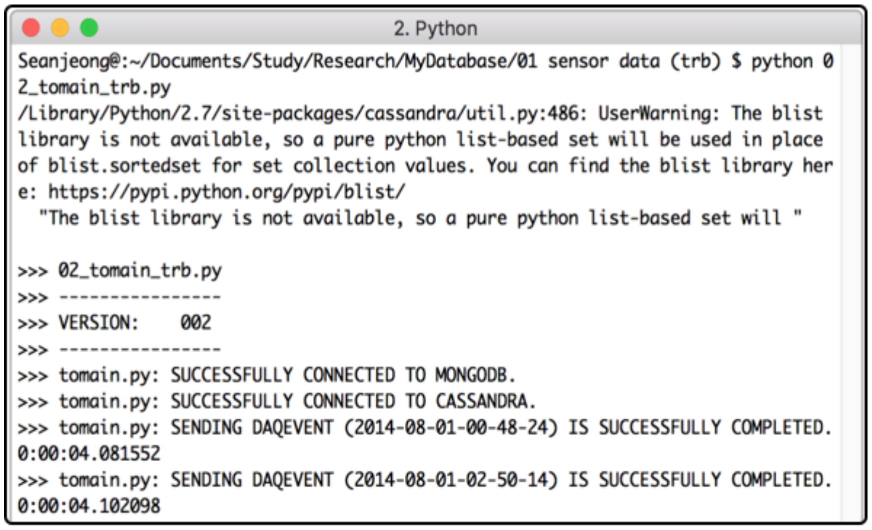
To simulate the in-situ bridge monitoring scenario, a script (written in Python) that periodically sends the sensor data sets to the onsite computer is developed. Once the sensor data is delivered to the onsite computer, the interface program (namely, `onsite.py`) on the onsite computer automatically re-structures the raw data according to the defined data schema for MongoDB and stores the parsed data to MongoDB. Figure 3.17 shows a screenshot of the `onsite.py` in operation. Once a data set for a single data acquisition event is stored in MongoDB on the onsite computer, another interface program (namely, `tomain.py`) on the onsite computer parses the data set stored in MongoDB to the defined data schema for Cassandra data and uploads the data to the Cassandra database in the main server. Figure 3.18 shows a screenshot of the `tomain.py` in operation.

Sensor data stored in the Cassandra database can be retrieved using CQL queries. Figure 3.19 shows an example of sensor data retrieval using a `SELECT` query. This query specifies sensor ID (`TRB_u07_ch0`), year (`2014`) and time range (from `2014-08-02T00:00:00` to `2014-08-02T01:00:00`) to retrieve sensor data. Once the query is



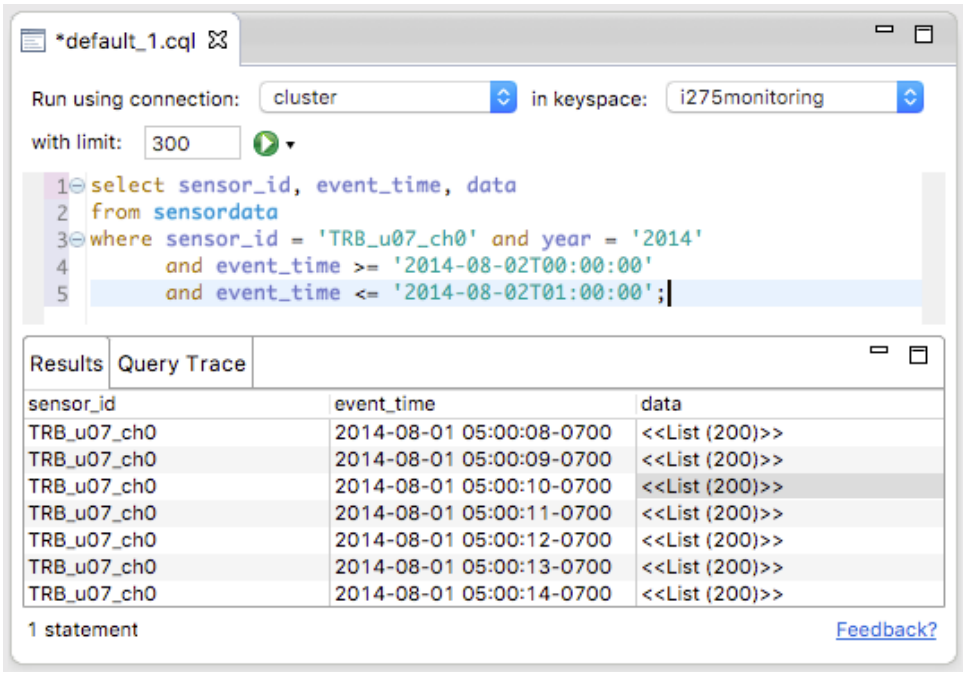
```
Seanjeong@:~/Documents/Study/Research/MyDatabase/01 sensor data (trb) $ python 0
1_onsite_trb.py

>>> 01_onsite_trb.py
>>> ----------------
>>> VERSION:    002
>>> ----------------
>>> onsite.py: SUCCESSFULLY CONNECTED TO MONGODB.
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-00-48-24 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-00-48-24)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-02-50-14 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-02-50-14)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-04-51-25 IS DETECTED.
>>> onsite.py: STORING DAQEVENT (2014-08-01-04-51-25)  IS SUCCESSFULLY COMPLETED
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-06-52-26 IS DETECTED.
>>> onsite.py: NEW DAQ EVENT data/2014-08-01-08-54-00 IS DETECTED.
```

Figure 3.17 onsite.py: Python script storing sensor data to MongoDB in onsite computer



```
Seanjeong@:~/Documents/Study/Research/MyDatabase/01 sensor data (trb) $ python 0
2_tomain_trb.py
/Library/Python/2.7/site-packages/cassandra/util.py:486: UserWarning: The blist
library is not available, so a pure python list-based set will be used in place
of blist.sortedset for set collection values. You can find the blist library her
e: https://pypi.python.org/pypi/blist/
  "The blist library is not available, so a pure python list-based set will "

>>> 02_tomain_trb.py
>>> ----------------
>>> VERSION:    002
>>> ----------------
>>> tomain.py: SUCCESSFULLY CONNECTED TO MONGODB.
>>> tomain.py: SUCCESSFULLY CONNECTED TO CASSANDRA.
>>> tomain.py: SENDING DAQEVENT (2014-08-01-00-48-24) IS SUCCESSFULLY COMPLETED.
0:00:04.081552
>>> tomain.py: SENDING DAQEVENT (2014-08-01-02-50-14) IS SUCCESSFULLY COMPLETED.
0:00:04.102098
```

Figure 3.18 tomain.py: Python script storing sensor data to Cassandra database in main server



```
select sensor_id, event_time, data
from sensordata
where sensor_id = 'TRB_u07_ch0' and year = '2014'
      and event_time >= '2014-08-02T00:00:00'
      and event_time <= '2014-08-02T01:00:00';
```

sensor_id	event_time	data
TRB_u07_ch0	2014-08-01 05:00:08-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:09-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:10-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:11-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:12-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:13-0700	<<List (200)>>
TRB_u07_ch0	2014-08-01 05:00:14-0700	<<List (200)>>

Figure 3.19 Select query for sensor data retrieval and query result in a tabular format

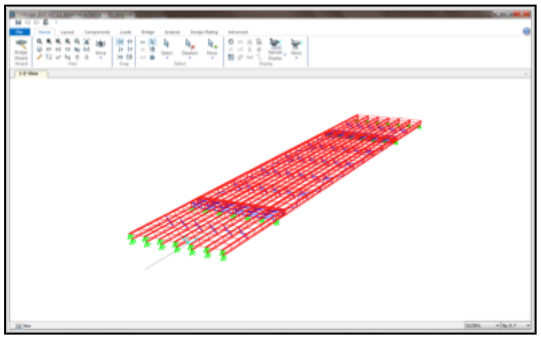
submitted, Cassandra database processes the query request and returns query results in a tabular structure, as shown in Figure 3.19.

Bridge engineering model is also stored in the Cassandra database in the main server. For storing the bridge engineering model, we export the FE model into Microsoft Excel format using CSiBridge's exporting function. The exported engineering model is first mapped into BrIM schema encoded in XML. The engineering model encoded in BrIM schema is then mapped into the database schema and stored in the Cassandra database in the main server using the data mapping scripts. Once stored, the bridge engineering model can be retrieved in different file formats. Figure 3.20(a) and (b) show the retrieved FE model (Excel file format) visualized using CSiBridge and the retrieved BrIM geometry model (XML file format) visualized using the OpenBrIM Viewer, respectively.

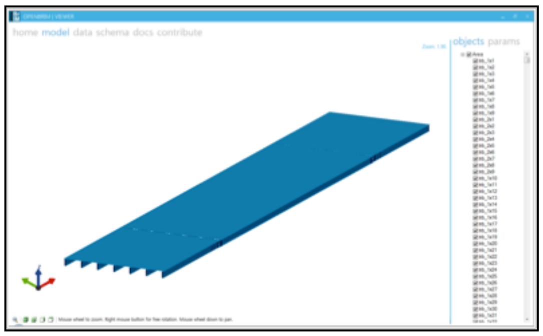
Similarly, sensor information is also stored in the Cassandra database in the main server. To conduct this task, we develop a Python script to parse the sensor information stored in Microsoft Excel Spreadsheet into the database schema and to send parsed sensor information to the main server. Figure 3.21 shows the sensor information retrieved using CQLSH, a command line client interface of Cassandra database.

3.4.2 Influence line analysis using sensor data and bridge engineering model

To take advantage of the integrated bridge monitoring infrastructure, influence line analysis, which compares bridge responses collected by the sensors with analytically computed response using the FE model, is conducted following the procedure described by Hou *et al*. [115]. In this analysis, we utilize sensor data collected from a field test for identification of vehicle-bridge interaction [115]. In the dynamic loading test, a single test truck instrumented with GPS sensor crosses the TRB without other traffics. The test truck passes the middle lane of the bridge at approximately 60 mph. The specification of the test

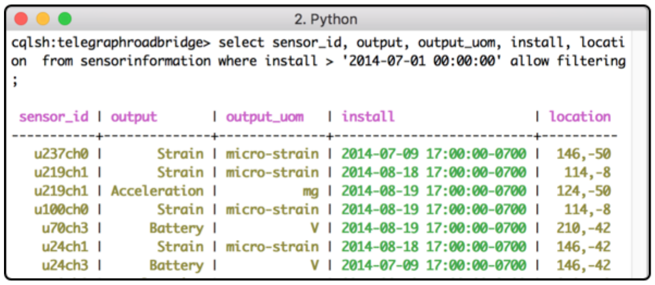


(a) Finite element model



(b) Bridge information model (geometry)

Figure 3.20 Bridge information model retrieved from Cassandra database in main server



```
cqlsh:telegraphroadbridge> select sensor_id, output, output_uom, install, locati
on  from sensorinformation where install > '2014-07-01 00:00:00' allow filtering
;

 sensor_id | output       | output_uom   | install                  | location
-----------+--------------+--------------+--------------------------+----------
   u237ch0 |       Strain | micro-strain | 2014-07-09 17:00:00-0700 |   146,-50
   u219ch1 |       Strain | micro-strain | 2014-08-18 17:00:00-0700 |    114,-8
   u219ch1 | Acceleration |           mg | 2014-08-19 17:00:00-0700 |   124,-50
   u100ch0 |       Strain | micro-strain | 2014-08-19 17:00:00-0700 |    114,-8
    u70ch3 |      Battery |            V | 2014-08-19 17:00:00-0700 |   210,-42
    u24ch1 |       Strain | micro-strain | 2014-08-18 17:00:00-0700 |   146,-42
    u24ch3 |      Battery |            V | 2014-07-09 17:00:00-0700 |   146,-42
```

Figure 3.21 Sensor information retrieved from Cassandra database in main server

truck can be found in Figure 3.22 and Table 3.1 [115]. When the test truck crosses the bridge, strain gauges (installed as described in Figure 3.23) measure the dynamic strain response of the bridge GPS sensor measures the location of the truck [115]. The collected data sets are stored in the Cassandra database in the main server. In addition to the sensor data, corresponding vehicle load and vehicle lane are defined in the FE model of the TRB (as shown in Figure 3.24) for the simulation. The FE model is mapped and transmitted to the Cassandra database in the main server.

Table 3.1 Test truck load description (unit: pound) [112]

Steer Axle	Drive Axle	Trailer Lead Axle	Trailer Rear Axle	Total
9,460	17,620	17,820	17,600	62,500

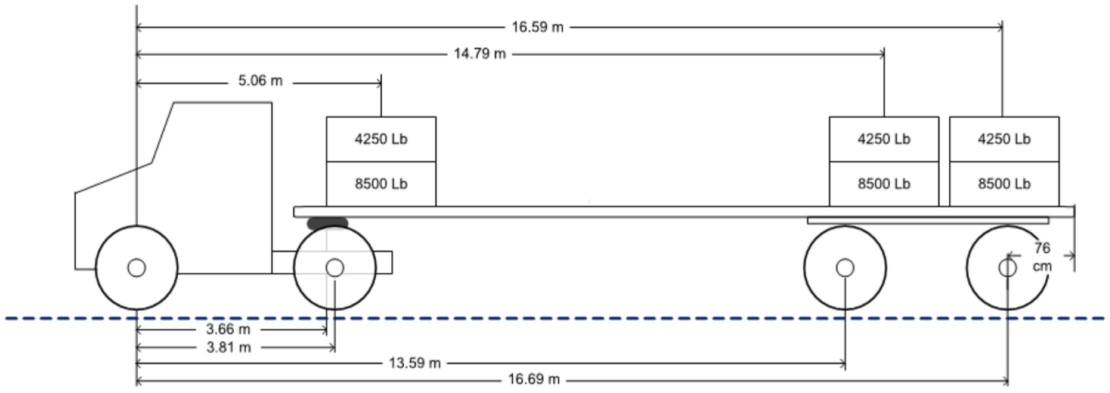
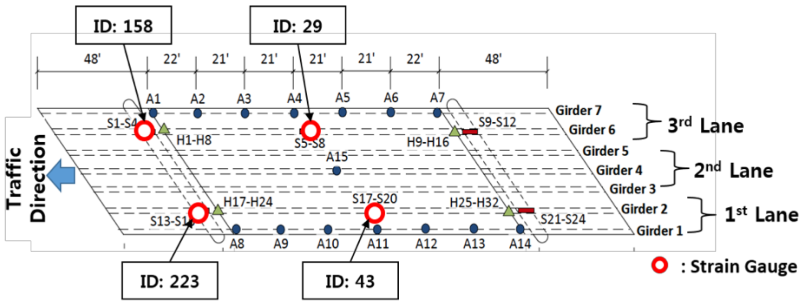
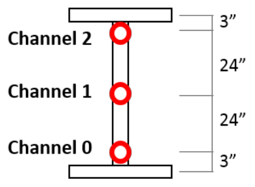


Figure 3.22 Test truck dimension

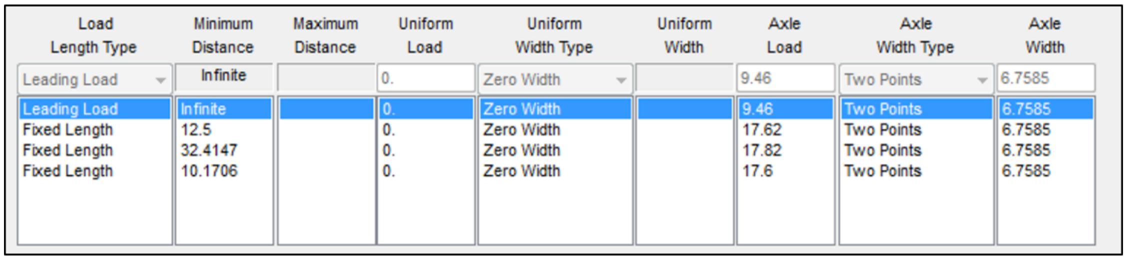


(a) Plan view



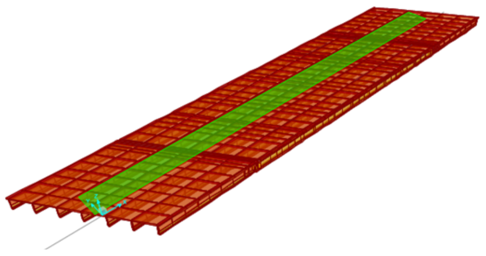
(b) Section view

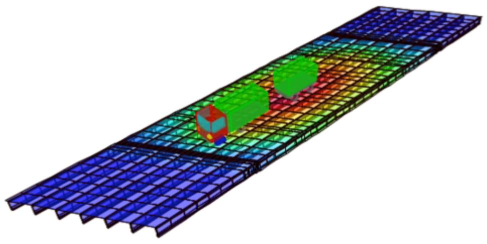
Figure 3.23 Location of strain gauges



Load Length Type	Minimum Distance	Maximum Distance	Uniform Load	Uniform Width Type	Uniform Width	Axle Load	Axle Width Type	Axle Width
Leading Load	Infinite		0.	Zero Width		9.46	Two Points	6.7585
Leading Load	Infinite		0.	Zero Width		9.46	Two Points	6.7585
Fixed Length	12.5		0.	Zero Width		17.62	Two Points	6.7585
Fixed Length	32.4147		0.	Zero Width		17.82	Two Points	6.7585
Fixed Length	10.1706		0.	Zero Width		17.6	Two Points	6.7585

(a) Vehicle load configuration

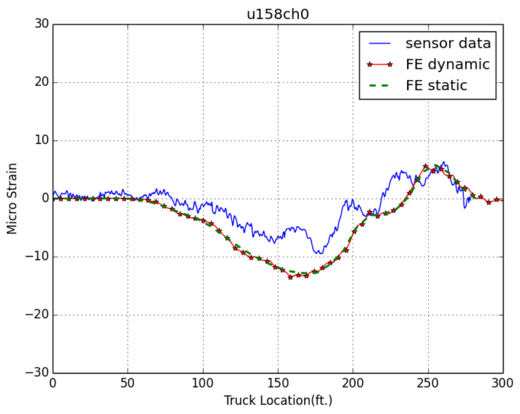




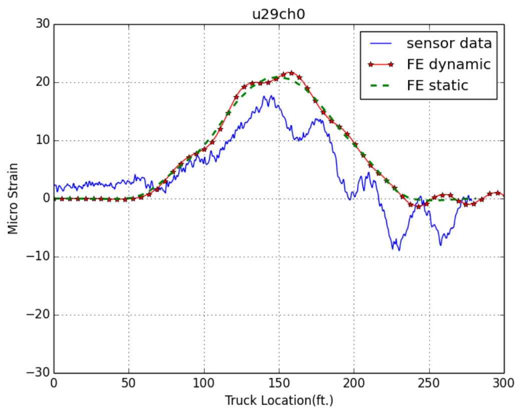
(b) Defined vehicle lane (c) Visualized test truck model

Figure 3.24 Test truck defined in FE model

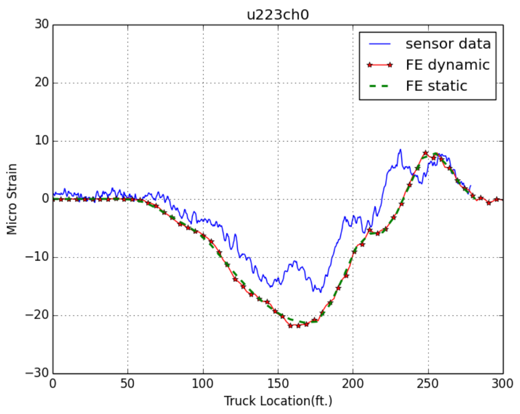
Once all the data is stored in the main server, we plot the influence lines for sensor data by retrieving the collected sensor data, including strain data and truck location data, and then plotting the strain response along the truck location. Next, we download the FE model from the main server and conduct static and dynamic FE analysis to compute the influence line at the locations of the strain gauges. For the FE analysis, direct integration method (Hilber-Hughes-Taylor method) without damping is employed, and 0.03 second is selected for the time step for the integration. Furthermore, since strain response cannot be directly obtained from the analysis results of CSiBridge, we calculate the strain indirectly using the stress response obtained from the analysis. Finally, we compare the measured response and analytical response of the bridge by overlaying the obtained influence lines. Figure 3.25(a), (b), (c), and (d) show the overlays of the influence lines at four different sensor locations, respectively. The results show that the measured response is very similar with the analytical response, although the analytical response shows slightly higher maximum response than the measured response.



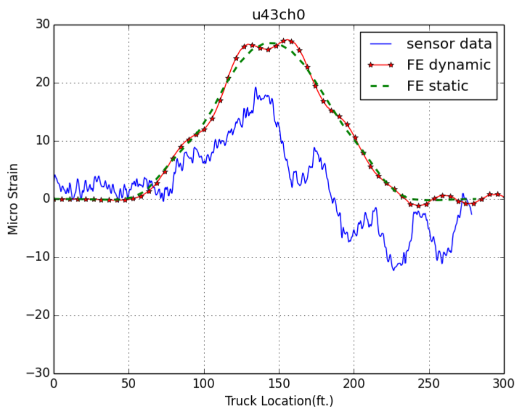
(a) Strain gauge at end-span of girder 6



(b) Strain gauge at mid-span of girder



(c) Strain gauge at mid-span of girder 2



(d) Strain gauge at mid-span of girder

Figure 3.25 Influence line analysis result

3.4.3 Comparison of sensor data and analytically computed bridge response

In this 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 3.26 shows the basic steps implemented for this example scenario.

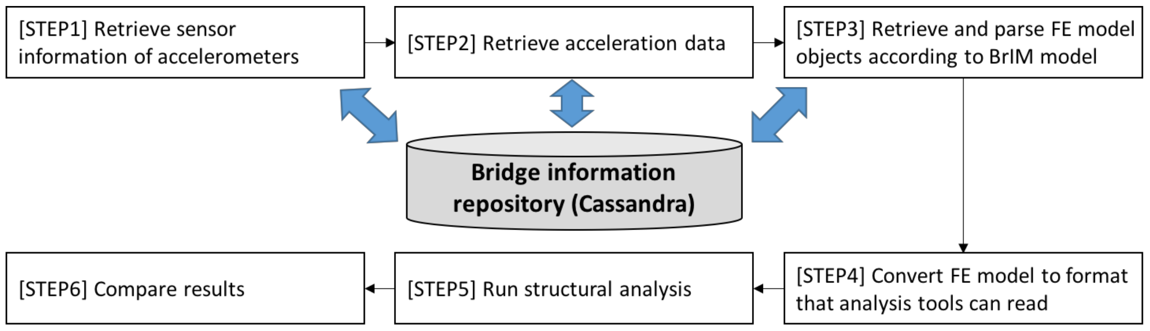
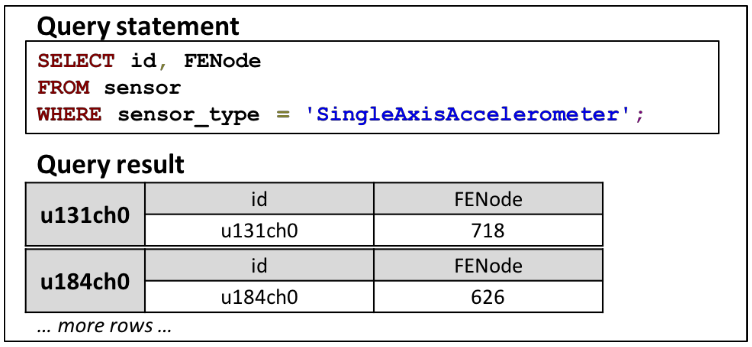

[STEP1] Retrieve sensor information of accelerometers
[STEP2] Retrieve acceleration data
[STEP3] Retrieve and parse FE model objects according to BrIM model
Bridge information repository (Cassandra)
[STEP4] Convert FE model to format that analysis tools can read
[STEP5] Run structural analysis
[STEP6] Compare results

Figure 3.26 Workflow of example scenario for comparing sensor data and analytically computed bridge reponse

As shown in Figure 3.26, the comparison of measurement and computed data is implemented in six steps. In step 1, we retrieve sensor information from the Cassandra database using CQL and Cassandra API. As shown in Figure 3.27, 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 3.27. 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 3.28 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 `2014-08-01T00:00:00` to `2014-08-01T02:00:00`. As shown in Figure 3.28, 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.

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 3.29 shows the pseudo code for retrieving and rebuilding BrIM model using the recursive function



Query statement

```
SELECT id, FENode
FROM sensor
WHERE sensor_type = 'SingleAxisAccelerometer';
```

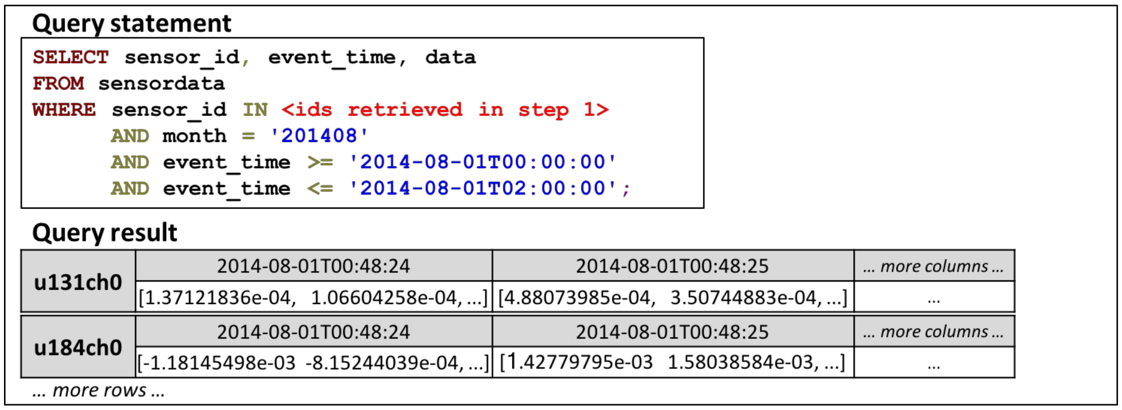
Query result

	id	FENode
u131ch0	u131ch0	718

	id	FENode
u184ch0	u184ch0	626

... more rows ...

Figure 3.27 Step 1: Retrieving sensor information



Query statement

```
SELECT sensor_id, event_time, data
FROM sensordata
WHERE sensor_id IN <ids retrieved in step 1>
      AND month = '201408'
      AND event_time >= '2014-08-01T00:00:00'
      AND event_time <= '2014-08-01T02:00:00';
```

Query result

	2014-08-01T00:48:24	2014-08-01T00:48:25	*... more columns ...*
u131ch0	[1.37121836e-04, 1.06604258e-04, ...]	[4.88073985e-04, 3.50744883e-04, ...]	...

	2014-08-01T00:48:24	2014-08-01T00:48:25	*... more columns ...*
u184ch0	[-1.18145498e-03 -8.15244039e-04, ...]	[1.42779795e-03 1.58038584e-03, ...]	...

... more rows ...

Figure 3.28 Step 2: Retrieving sensor data

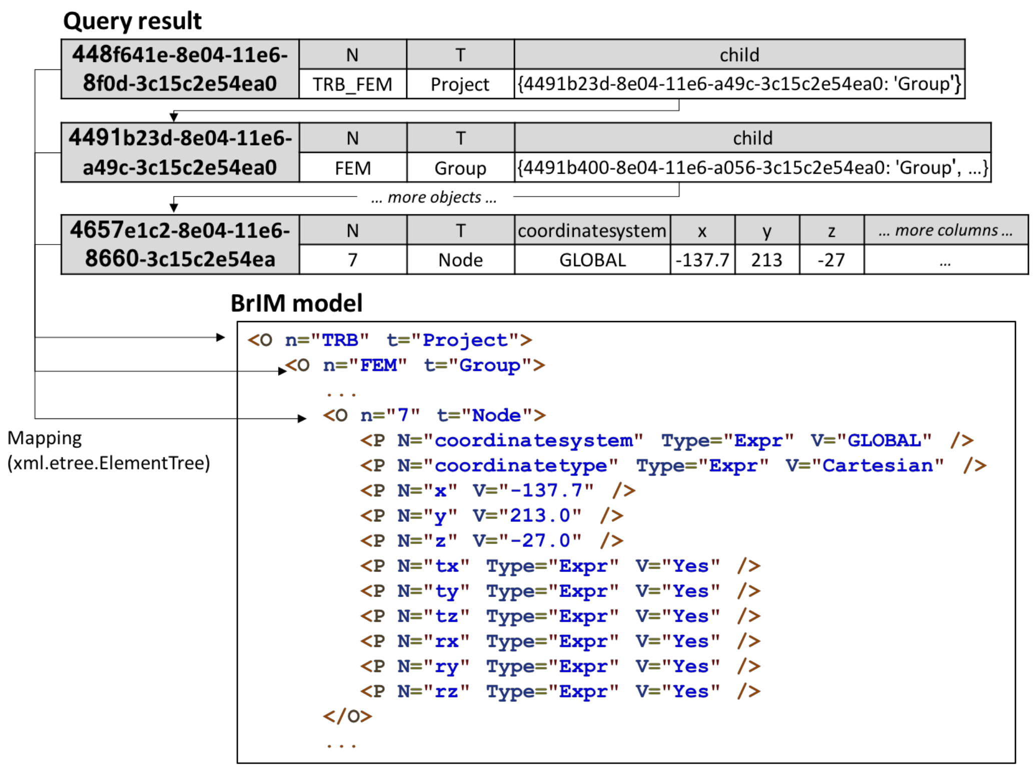
`RetrieveDataByKey` (lines 1 to 10), which includes the query (see lines 2 to 4) to (recursively) retrieve the FE model information. If retrieved object contains child column, the function `RetrieveDataByKey` calls itself with input argument specifying the `uid` and column family of child objects (line 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 [108], as illustrated in Figure 3.30.

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 CSiBridge software. Figure 3.31 shows the

Pseudo code

```
def RetrieveDataByKey(key, columnfamily):
  Query = "SELECT *
           FROM columnfamily
           WHERE uid = key"
  Object = RunQuery(Query)
  if RetrievedObject["child"] is not None:
      children = Object["child"]
      for child in children:
          Object.attach(RetrieveDataByKey(child, children[child])) # Recursive Function
  return Object

def main():
   # Start recursive function from the root object
   RetrieveDataByKey("urn:uuid:448f641e-8e04-11e6-8f0d-3c15c2e54ea0", "Project")
```

Figure 3.29 Step 3: Pseudo code of a recursive function for FE model retrieval



448f641e-8e04-11e6-8f0d-3c15c2e54ea0	N	T	child
	TRB_FEM	Project	{4491b23d-8e04-11e6-a49c-3c15c2e54ea0: 'Group'}

4491b23d-8e04-11e6-a49c-3c15c2e54ea0	N	T	child
	FEM	Group	{4491b400-8e04-11e6-a056-3c15c2e54ea0: 'Group', ...}

4657e1c2-8e04-11e6-8660-3c15c2e54ea	N	T	coordinatesystem	x	y	z	... more columns ...
	7	Node	GLOBAL	-137.7	213	-27	...

```
<O n="TRB" t="Project">
  <O n="FEM" t="Group">
    ...
    <O n="7" t="Node">
      <P N="coordinatesystem" Type="Expr" V="GLOBAL" />
      <P N="coordinatetype" Type="Expr" V="Cartesian" />
      <P N="x" V="-137.7" />
      <P N="y" V="213.0" />
      <P N="z" V="-27.0" />
      <P N="tx" Type="Expr" V="Yes" />
      <P N="ty" Type="Expr" V="Yes" />
      <P N="tz" Type="Expr" V="Yes" />
      <P N="rx" Type="Expr" V="Yes" />
      <P N="ry" Type="Expr" V="Yes" />
      <P N="rz" Type="Expr" V="Yes" />
    </O>
    ...
```

Figure 3.30 Step 3: Data mapping from query result to hierarchical BrIM model

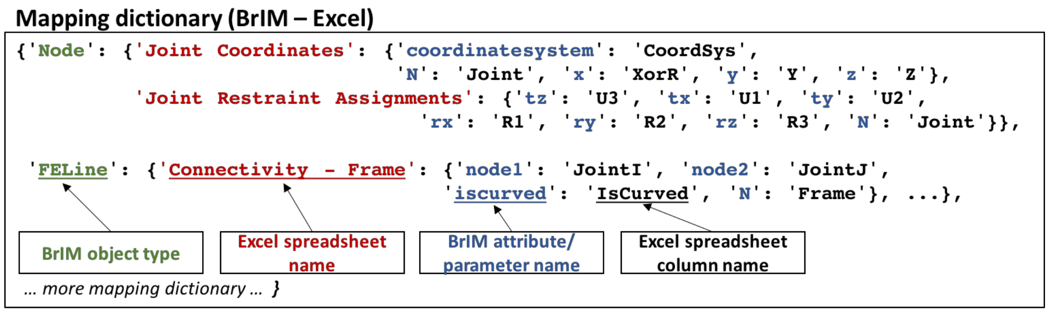
pseudo code for the data mapping from BrIM to the Excel spreadsheet. The pseudo code has a recursive function to explore every object in hierarchical object-tree structure. Specifically, the recursive function parses attributes, parameters and child objects of a single object (lines 2 to 4), and then 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 3.32). The xml.etree.ElementTree package [108] and openpyxl package [116] are employed for parsing XML and Excel spreadsheet files.

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 CSiBridge's APIs. As shown in Figure 3.33, Python script calls the VBA scripts using Python extensions for Windows (pywin32) package [117]. 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`.

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    PareObject(getroot(inputXMLFile))
```

Figure 3.31 Step 4: Pseudo code for FE model mapping from BrIM to Excel spreadsheet

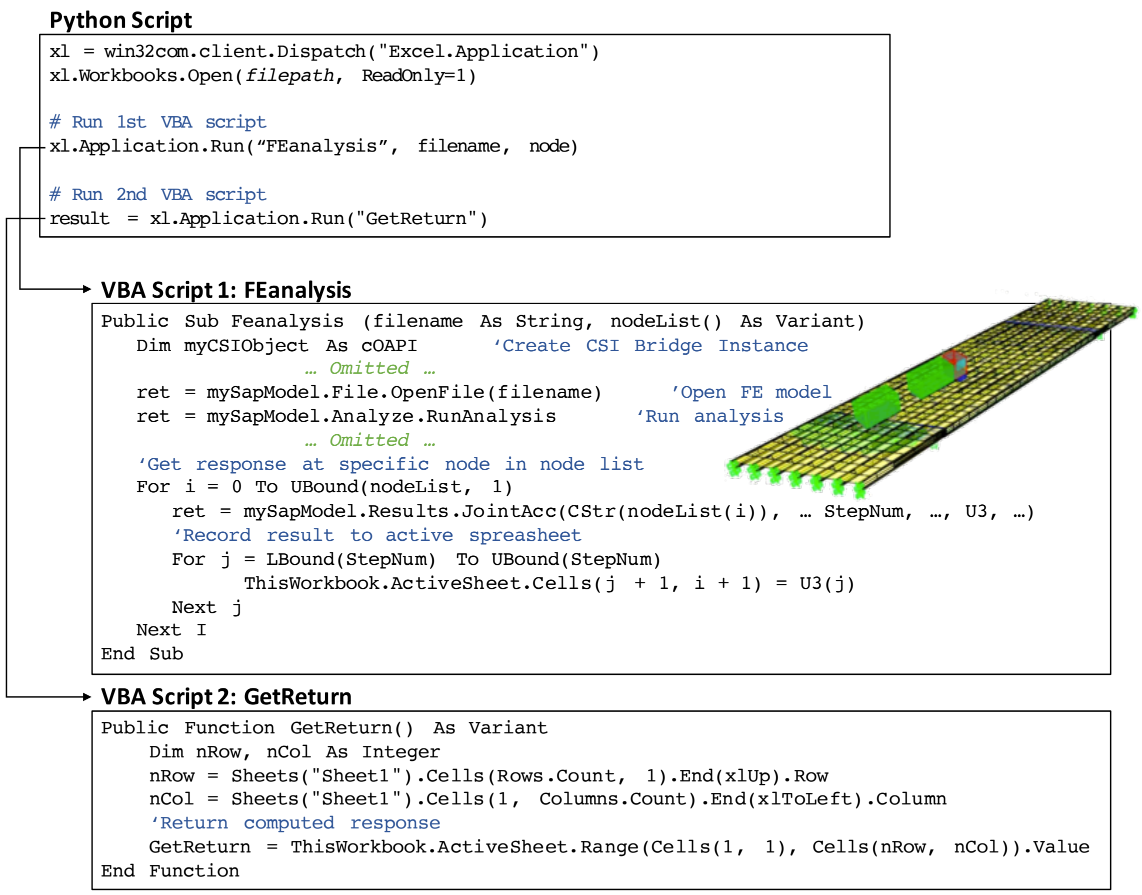


Mapping dictionary (BrIM – Excel)

```
{'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'}, ...},
```

… more mapping dictionary … }

Figure 3.32 Step 4: Mapping dictionary from BrIM to Excel spreadsheet



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 spreasheet
      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 3.33 Step 5: Running FE analysis using CSiBridge and its APIs

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 3.34. 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.

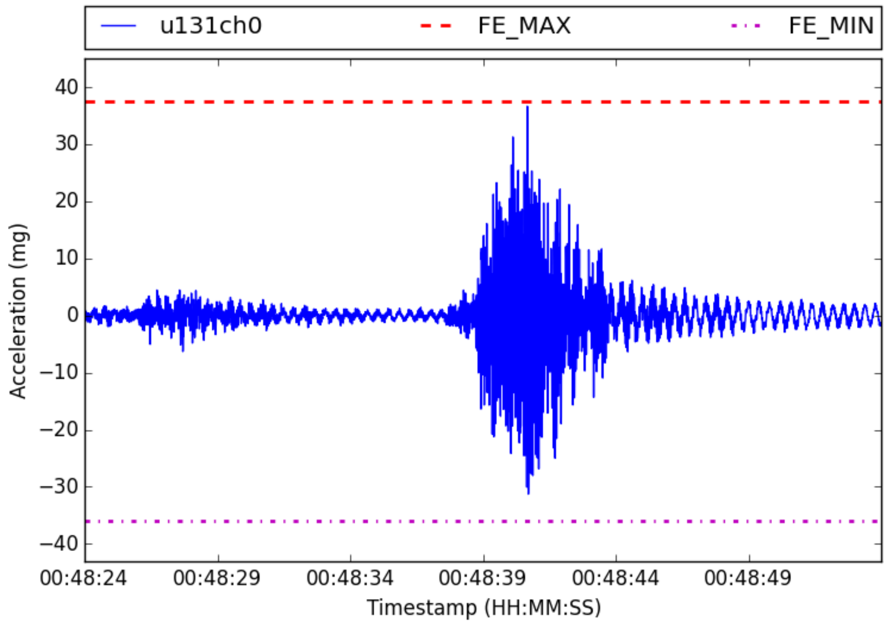


Figure 3.34 Step 6: Plotting retrieved sensor measurement (u131ch0) along with the maximum and minimum values of the response obtained from FE simulation

3.4.4 Retrieval of sensor data along with traffic image data

In this 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 3.35 shows the three basic steps for implementing the data retrieval process.

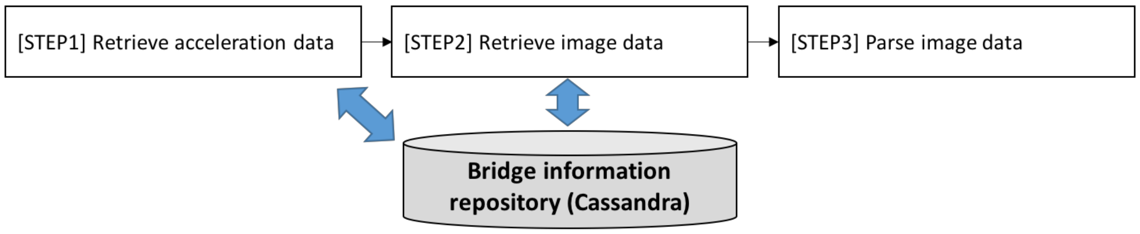

[STEP1] Retrieve acceleration data
[STEP2] Retrieve image data
[STEP3] Parse image data
Bridge information repository (Cassandra)

Figure 3.35 Workflow of example scenario for retrieving sensor data along with traffic-image data

In step 1, the sensor measurements are retrieved using CQL. As shown in Figure 3.36, 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 `2016-08-23T10:02:09` and `2016-08-23T10:03:08`.

In step 2, traffic-monitoring images are retrieved to observe the vehicles that affect the bridge response retrieved in step 1. Figure 3.37 shows the query statement for retrieving the image data from the camera ID `telegraph2`, date `20160823` and the period between `2016-08-23T10:01:54` and `2016-08-23T10:03:08`. 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.

In step 3, the binary data is converted to image, for example, using Python's StringIO library [118]. Figure 3.38 shows the retrieved images with the corresponding sensor data.

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 3.36 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';
```

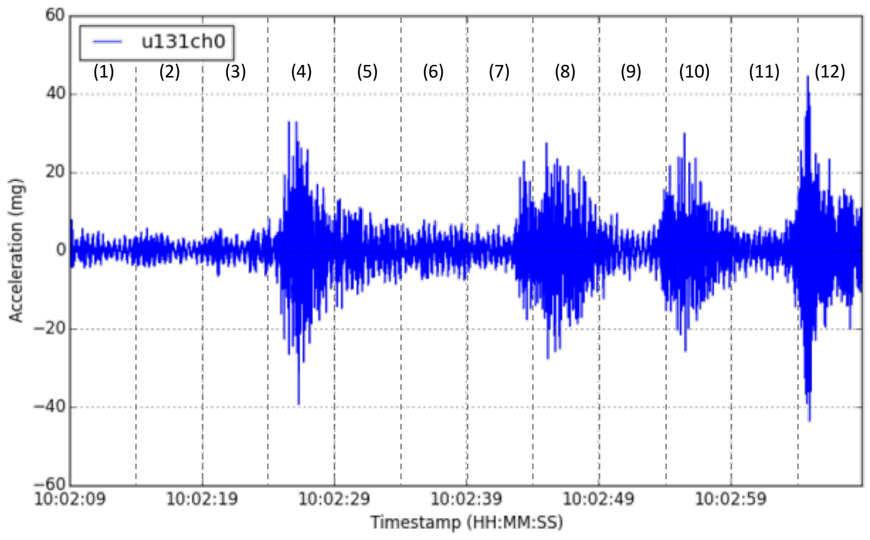
Figure 3.37 Step 2: Query statement for retrieving image data

The images shown in Figure 3.38(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 3.38) 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.

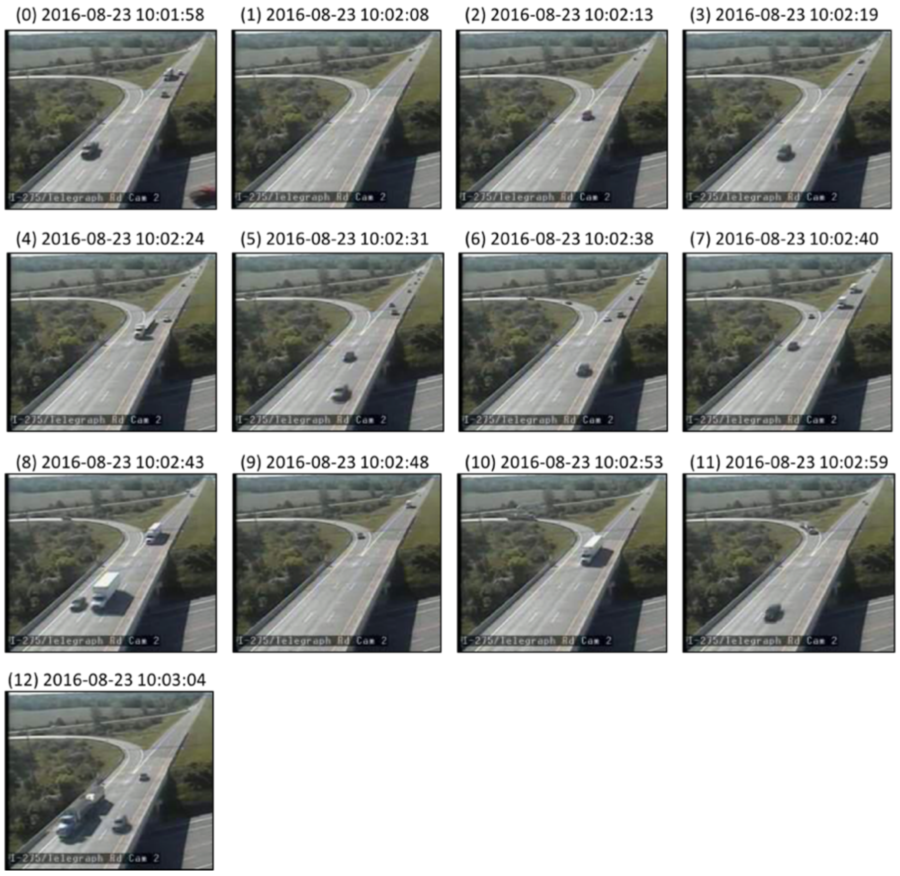
3.5 Summary

This chapter presents a scalable data management framework for handling massive data collected from civil infrastructure monitoring. NoSQL database systems are leveraged for the implementation of the framework. Unlike the traditional civil infrastructure monitoring systems, the proposed framework offers high scalability on a distributed computing environment, which makes the framework as a desirable alternative to handle ever-increasing monitoring data. Furthermore, the proposed framework provides data schema flexibility, which is useful to manage object-oriented civil infrastructure information. This framework also enables data interoperability and integration based on the information modeling standards.

The data management framework consists of four major components. The onsite computer receives data from the sensors, stores it temporarily, (optionally) performs simple analysis and transmits the data to the main server. The main server stores the sensor data permanently along with the relevant information, such as sensor metadata, geometric model and engineering model. The local computer retrieves data from the main server, performs analyses and returns analysis results back to the main server. The end user device



(a) Retrieved acceleration data



(b) Retrieved traffic-monitoring images

Figure 3.38 Acceleration response of the TRB from 2016-08-23T10:02:09 to 2016-08-23T10:03:08 collected by "u131ch0" and corresponding traffic-monitoring images.

retrieves data and visualizes the data on user interfaces. Based on the data management requirements of each component, a column family database (Cassandra database in current implementation) that is suitable for large-scale distributed database is employed for the main server, while a document-oriented database (MongoDB in current implementation) that has advantages on the schema-less data structure and fast, convenient query is employed for the onsite computer and local computer. Data schemas for sensor data, sensor information and bridge information model are designed to facilitate system automation and to improve data management performance. More specifically, data schemas for bridge information and sensor information are defined based on the BrIM schema to enable data mapping between BrIM schema and database schema, which can allow the data stored in the database to be utilized and integrated by different applications. Furthermore, data schema for time-series sensor data is defined using time-series data modelling scheme of Cassandra database for the fast query performance.

The proposed framework is demonstrated using the sensor data collected from the Telegraph Road Bridge and bridge models of the bridge. In addition to basic data store and retrieval examples, this chapter presents three case scenarios that involve different types of bridge information, which are typically managed by isolated systems and hard to integrate. The first scenario is the influence line analysis that compares the influence line measured by sensors with the influence line obtained from finite element analysis. The second scenario compares the vibration response measured by sensors with the vibration response obtained from finite element analysis. The third scenario compares the vibration response measured by sensors with the traffic images collected by traffic monitoring system. The results show that the proposed data management framework not only offers scalable data management environment, but also allows client systems to easily query and integrate heterogeneous data in bridge monitoring applications.

Chapter 4

A Cloud-based Cyberinfrastructure Platform for Civil Infrastructure Monitoring

4.1 Introduction

With the advances in information and communication technology (ICT), as well as reduced cost of monitoring system, recent trends of civil infrastructure monitoring include the use of sensor network with higher density. The advanced ICT and increasing use of sensors will realize the concept of cyber-physical system (CPS) wherein physical systems (e.g., civil infrastructure) and computational systems (e.g., data repository and analysis modules) are tightly integrated [119]. Physical systems can be monitored, assessed and controlled with or without human intervention. Furthermore, with the rapid development of data-driven analysis methods, massive and diverse data collected from monitoring systems offer promising opportunities to find new insights about the physical systems. To facilitate data utilizations, data needs to be accessed and retrieved easily by different applications, such as data analysis modules and user interfaces. However, current monitoring systems are not designed to support interoperable data access; rather engineers often need to download data and feed the data into applications manually. This limits the rapid prototyping of applications and system automation. In order to facilitate the use of valuable monitoring data, a comprehensive data management platform that offers interoperable interfaces and easy-to-use data management service will be necessary.

In the IoT domain, many IoT software platforms have been developed to support sensor data management [25, 26, 27, 28, 29]. Sensor can transmit data to such IoT platforms via interfaces adhering to standard communication protocols. Sensor data transmitted to IoT platforms can be retrieve by different applications also via standardized interfaces. Based on the machine-understandable interfaces, IoT platforms can ease the application development and enables automated data utilization. However, such IoT platforms are typically designed to handle mainly sensor measurement data. In civil infrastructure monitoring domain, a data management platform needs to handle not only the sensor data, but also the domain information, such as engineering model, geometric model, inspection information, etc.

This chapter presents a cyberinfrastructure platform which offers data management as services similar to IoT software platforms but is tailored to civil infrastructure monitoring [73, 74, 75, 76]. The proposed platform brings together the information modeling (described in Chapter 2), NoSQL-based data management system (described in Chapter 3), web service technologies and cloud computing to offer interoperable data management services. Based on standard communication protocols, web services, as a database wrapper, offer interfaces to different systems and applications for accessing the database. The cyberinfrastructure platform is deployed on cloud computing environment to offer reliable data management service for handling continuously incoming sensor data, as well as scalability for handling ever-increasing amount of sensor data. This chapter also discusses a hybrid cloud-based deployment of the platform for outsourcing the management of non-sensitive, voluminous data to the public cloud, while managing sensitive data and applications within the private cloud separately.

This chapter is organized as follows. Section 4.2 provides an overview of the proposed cyberinfrastructure platform for civil infrastructure monitoring. Section 4.3 and Section 4.4 describe data store process and data retrieval process, respectively, with the details of web services. Section 4.5 presents cloud-based implementations of the cyberinfrastructure platform. Section 4.6 demonstrates a prototype implementation of the cyberinfrastructure platform using the monitoring data of the Telegraph Road Bridge in Monroe, Michigan. This chapter is concluded with a summary in Section 4.7.

4.2 Overview of Cyberinfrastructure Platform

Figure 4.1 shows the overall architecture of the proposed cyberinfrastructure platform for civil infrastructure monitoring [74]. The platform receives data from various data sources, including sensor networks and information models. Sensor networks (e.g., structural health monitoring system) collects heterogeneous sensor data, ranging from high-frequency time-series data to video and camera images. Collected sensor data is transmitted to the cyberinfrastructure platform through the communication network (e.g., the Internet) and

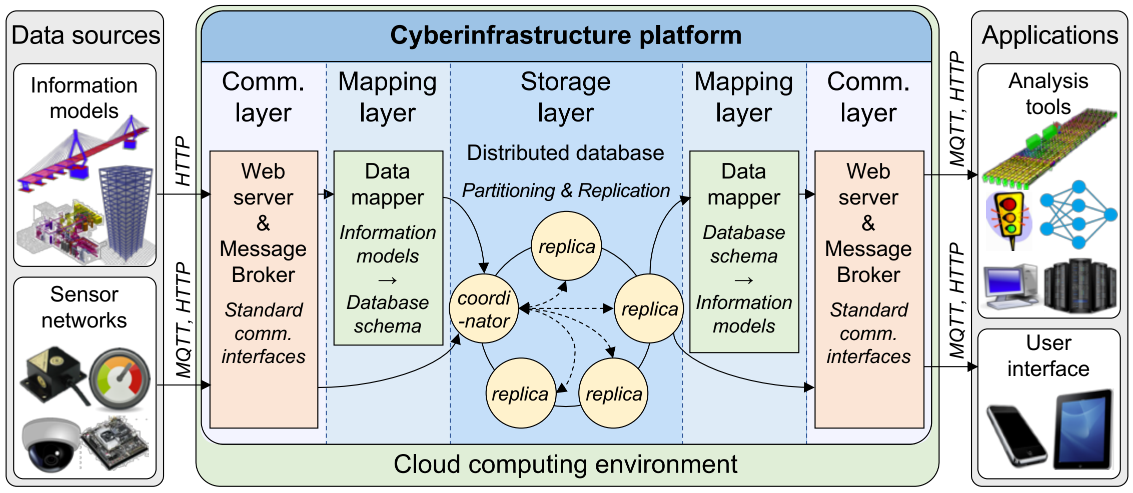


Figure 4.1 Overall architecture of the cyberinfrastructure platform for engineering

stored. Information contains comprehensive information (e.g., geometry, physical properties, functional characteristics and sensor information) of target systems. Information models are created on engineers' personal computer and transmitted to the cyberinfrastructure platform through the communication networks.

The cyberinfrastructure platform serves as a data hub that receives, processes, analyzes, stores, distributes and shares monitoring data. The cyberinfrastructure platform is composed of three basic layers, namely, communication layer, mapping layer and storage layer, to support data store and retrieval. For data store processes, the communication layer handles communication with the data sources. Specifically, the web server and message broker in the communication layer provide standardized interfaces to receive data from different data sources via the Internet. The mapping layer includes data mappers (which is described in Section 3.3.2.2.2) that map the received semi-structured information models onto the database schema. The mapped data is passed to the storage layer which includes a NoSQL-based distributed database system (which is described in Chapter 3) that partitions, replicates and stores data.

The data stored in the cyberinfrastructure platform needs to be accessed and retrieved by different applications, such as data analysis tools, engineering analysis software, 3-D modeling tools and mobile user devices. To support data retrieval services that can be invoked by different applications, the cyberinfrastructure platform offers platform-neutral interfaces which are hosted on the web server in the communication layer. Once invoked, a data retrieval service retrieves data from the storage layers and maps the data through the mapping layer. The retrieved data is then delivered to the application that invokes the data retrieval service.

The proposed cyberinfrastructure platform is deployed on a cloud computing environment for scalability, accessibility and reliability. Specifically, the cyberinfrastructure platform can be deployed on the Infrastructure as a Service (IaaS) layer of cloud (i.e., virtual machines offered by cloud) [56], which assures platform portability across different cloud vendors or among public cloud, private cloud and hybrid cloud.

4.3 Data Store Process

This section describes the process of acquiring and storing the data with the cyberinfrastructure platform [73, 74]. Data store requests are processed through three layers: communication, mapping and storage. The communication layer employs web server and message broker built upon standard protocols, so that various data sources can access the platform. The mapping layer defines the mapper to help store the semi-structured information models in the database. Finally, the storage layer employs a distributed NoSQL database to enable scalable data management.

4.3.1 Communication layer

The communication layer is exposed to clients (e.g., data sources) via the Internet to enable remote access. The layer serves as intermediary and accepts messages with the data from the clients, parses the message to extract the data, and passes the data to the appropriate layer. To support different communication protocols often used in IoT applications, this layer includes two systems: (1) a web server based on the Hypertext Transfer Protocol (HTTP) for supporting client-server communications and (2) a message broker based on the Message Queuing Telemetry Transport (MQTT) for supporting publish-subscribe communications. In the prototype implementation, both systems are deployed using Node.js [120], a server-side JavaScript runtime environment.

4.3.1.1 Web server

The web server is implemented to provide web services, which is, as defined by W3C, is a “software system designed to support interoperable machine-to-machine interaction over a network [121].” Web services enable sharing of data and integration of applications over the network. Since the cyberinfrastructure platform needs to support utilization of data from various devices (e.g., cloud, local computer, micro-computer and mobile devices) and platforms (e.g., different operating systems), it is important to employ a widely-adopted web service protocol. Furthermore, the cyberinfrastructure platform needs to support conditional queries involved in SHM applications (e.g., range query for time-series data). To meet these requirements, the cyberinfrastructure platform employs RESTful web services [66] which have fast performance and high scalability [68]. RESTful web services are described using five constraints [122]:

- *Resource identification.* Resources are identified by uniform resource identifiers (URI).
- *Uniform interface*. Resources can be accessed via the HTTP.
- *Self-descriptive messages.* Resources are represented using standardized formats, such as Hypertext Markup Language (HTML), XML and JavaScript Object Notation (JSON).
- *Hypermedia as the engine of application state.* Resources contain links by which clients can interact with web services.
- *Stateless interactions*. Requests contain all the required information for web services to process the requests.

The proposed web server hosts a set of web services to handle different types of data. Table 4.1 summarizes the data store web services currently implemented in the cyberinfrastructure platform. Here, the HTTP method `POST` is used to submit data to the specified URIs [66]. For example, Figure 4.2 shows a schematics of a web server hosting web services for storing engineering models and sensor data. In this example, a client system, such as a local desktop computer, sends an HTTP request specifying the host name

Table 4.1 RESTful data store web services currently implemented on the cyberinfrastructure platform

Service	**HTTP method**	**URI**
Sensor data store	`POST`	`/sensordata`
Traffic image store	`POST`	`/imagedata`
Sensor information store	`POST`	`/sensor`
Geometric model store	`POST`	`/geometricmodel`
Engineering model store	`POST`	`/femodel`

(`<wsAddress>`) and the Uniform Resource Identifier (URI) (`/femodel`) of the web service for engineering model store. In addition, the HTTP request includes an engineering model file written in standard syntax such as eXtensible Markup Language (XML) that the client wants to store in the cyberinfrastructure platform. Upon receiving the message, the web server calls the web service corresponding to the request specified. The web service then parses the HTTP request, extracts the engineering model file, and delivers the file to the next layer (i.e., mapping layer). Once the process is completed, the web service returns an HTTP response notifying that the request has been processed successfully. If any error occurs during the process, the web service returns an HTTP response with an error message.

The web server processes a sensor data store request in a similar manner. In Figure 4.2, for example, a gateway system connected to a sensor network sends an HTTP request that contains sensor data written in JSON format, as well as the host address (`<wsAddress>`) and the URI (`/sensordata/001`) for the sensor data store service. Once the web server receives the request, a web service is invoked, parses the request and extracts sensor data. It should be noted that the sensor data is delivered directly to the storage layer without

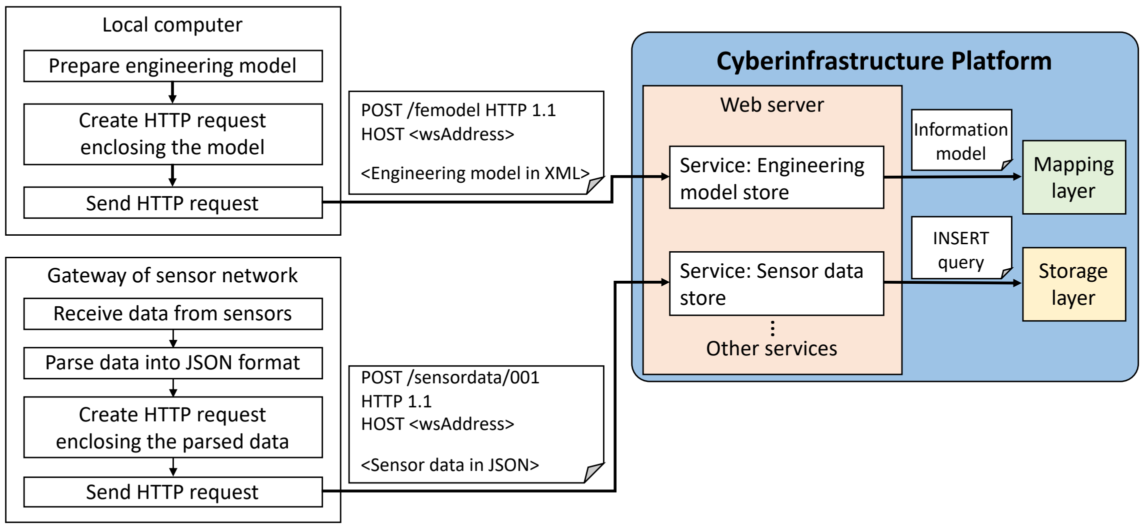

Local computer
Prepare engineering model
Create HTTP request enclosing the model
Send HTTP request
POST /femodel HTTP 1.1
HOST <wsAddress>
<Engineering model in XML>
Gateway of sensor network
Receive data from sensors
Parse data into JSON format
Create HTTP request enclosing the parsed data
Send HTTP request
POST /sensordata/001 HTTP 1.1
HOST <wsAddress>
<Sensor data in JSON>
Cyberinfrastructure Platform
Web server
Service: Engineering model store
Service: Sensor data store
Other services
Information model
INSERT query
Mapping layer
Storage layer

Figure 4.2 Web server hosting web services for engineering model and sensor data store

going through the mapping layer in the current implementation, since sensor data typically does not require complex data mapping.

4.3.1.2 Message broker

The message broker is implemented to support M2M communication for sensor data exchange based on the publish-subscribe paradigm. Publisher clients (e.g., data sources) and subscriber clients (e.g., application programs) can exchange messages in one-to-one, one-to-many and many-to-many communication in real time. This real time messaging can be used not only to store data, but also to enable tasks, such as real time analytics and event triggering, executed in real time. The message broker manages several "topics", each of which is defined for a specific type of sensor data. Data sources can use the topic to specify the type of data. Figure 4.3, for example, describes a message broker for sensor data store. In this example, a gateway device publishes a message including a topic (`sensor/data`) and sensor data written in JSON to the message broker through the MQTT protocol. The message broker classifies published message based on the topic, parses message to extract data and passes the data to the storage layer by sending an `INSERT` query statement. In addition, the published message is broadcasted to every application subscribing the topic (`sensor/data`) in real-time. Therefore, the message broker can support real-time tasks, such as real-time analysis and event handling.

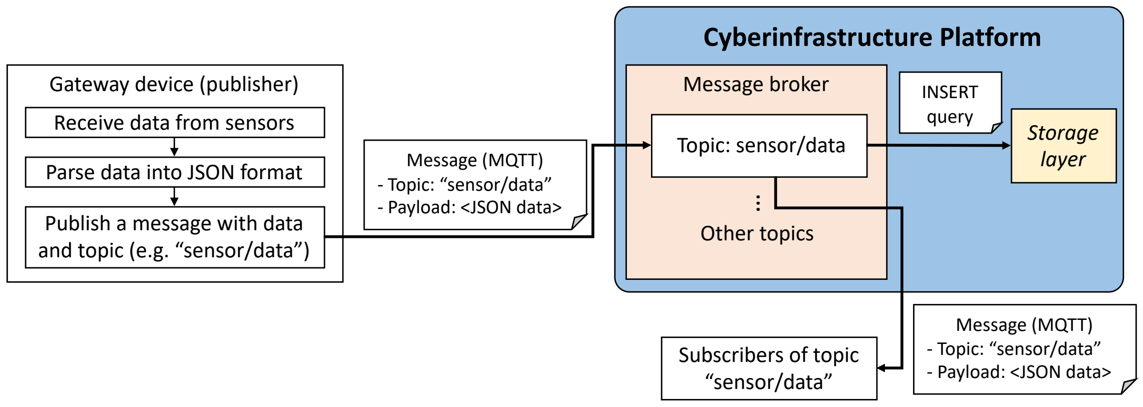

Cyberinfrastructure Platform
Gateway device (publisher)
Receive data from sensors
Parse data into JSON format
Publish a message with data and topic (e.g. "sensor/data")
Message (MQTT)
- Topic: "sensor/data"
- Payload: <JSON data>
Message broker
Topic: sensor/data
Other topics
INSERT query
Storage layer
Subscribers of topic "sensor/data"
Message (MQTT)
- Topic: "sensor/data"
- Payload: <JSON data>

Figure 4.3 Message broker including a topic for sensor data exchange

4.3.2 Mapping layer

The mapping layer receives the information models from the communication layer, maps the model to a database schema, and loads the mapped data to the database. Information models are typically written in object-oriented manner to represent a system as a set of hierarchical objects. Each object includes information about its characteristics, such as physical properties, functional role and relationship with other objects. For interoperability, information models are often written with XML or XML-based syntax adopted by information modeling standards. For the mapping of such information models, the mapping layer includes a data mapper that has information about the relation between the information model schema and the database schema. In the prototype implementation, a Python script is written as a data mapper which can be called by a web service in the communication layer. The data mapper uses an XML parser (e.g., Python xml.etree.ElementTree package [108]) to parse an XML-based information model and a database driver (e.g., Cassandra driver API [105]) to transmit the mapped data to the storage layer.

The data mapper works as follows. Upon receiving an information model, the data mapper is invoked. The data mapper first parses the model written in XML into a hierarchical object structure using the XML parser. Each object in the structure includes information about its properties as well as hierarchical relationship (e.g., parent and child objects information). The mapping script accesses the root object and maps the root object into the database schema. The data mapper then creates an `INSERT` query request for the mapped object and delivers the query request to the next layer (i.e., storage layer) using the database driver. This process is conducted recursively for the child objects in the hierarchical object structure until all the leaf node objects are processed.

4.3.3 Storage layer

The storage layer provides a data management service implemented using a distributed database. This layer receives information model data (from the mapping layer) and sensor data (from the communication layer), and stores them in the database. For the effective data management, it is critical to choose an appropriate database management system (DBMS). Since the cyberinfrastructure platform aims to manage a large volume of sensor data and engineering information model, scalability is an important factor for choosing a DBMS. As discussed in Chapter 3, Apache Cassandra database [99] is suitable for large-scale distributed data management. The prototype implementation of the platform employs Cassandra database for data management in the storage layer.

Using cloud computing environments (in particular, IaaS), a Cassandra database cluster is composed of multiple nodes, each of which can be employed on a physical or virtual machine. Cassandra glues the nodes distributed over multiple machines. Furthermore, Cassandra handles data partitioning and replications over multiple nodes according to the defined network topology and replication factor. For instance, a Cassandra database cluster, which has replication factor of two, partitions incoming data into multiple pieces and stores them twice over the distributed database nodes. With partitioning and replication, a Cassandra database cluster can be available even when some of the nodes are down.

4.4 Data Retrieval Process

This section describes data retrieval process with the proposed cyberinfrastructure platform [73, 74]. Web services for data retrieval are developed to provided standardized interfaces that applications on various systems and devices can invoke. In a data retrieval process, a request is delivered from an application to the communication layer, to the mapping layer and to the storage layer, whereas data is delivered in reverse order. The provided web services can be composed together to create new services.

4.4.1 Data retrieval using web services

The communication layer handles the communication required by the applications. The communication layer includes a web server that provides RESTful web services for applications to retrieve the data, including (partial and entire) information models and heterogeneous sensor data. Table 4.2 summarizes the data store web services currently implemented in the cyberinfrastructure platform. Here, the HTTP method `GET` is used to retrieve data from the specified URIs [66].

For example, as shown in Figure 4.4, an engineering analysis application can download an analysis model, which is a part of an information model, through the communication, mapping and storage layer, as follows:

(1) The client sends a `GET` request to the web server in the communication layer with URI (`/femodel/TRB`), protocol (`HTTP/1.1`) and host (`<ws_address>`).

(2) The corresponding web service runs a BrIM mapper (`cass_to_brimfem.py`) in the mapping layer.

Table 4.2 RESTful data retrieval web services currently implemented on the cyberinfrastructure platform

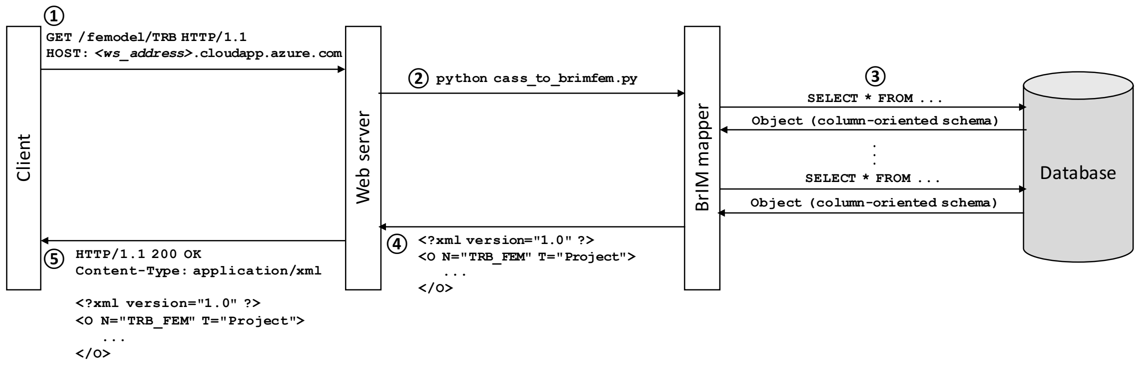
Service	HTTP Method	URI	Parameter
Sensor data retrieval	`GET`	`/sensordata/{sensorID}`	`event_time_begin`, `event_time_end`
Traffic image retrieval	`GET`	`/imagedata/{cameraID}`	`event_time_begin`, `event_time_end`
Sensor list retrieval	`GET`	`/sensor`	`sensor_type`, `install`, `remove`
Sensor information retrieval	`GET`	`/sensor/{sensorID}`	`install`, `remove`
Geometric model retrieval	`GET`	`/geometricmodel/{BridgeID}`	
Engineering model retrieval	`GET`	`/femodel/{BridgeID}`	`file_format` (xml or xlsx)

(3) The BrIM mapper retrieves and maps relevant data entities recursively from the distributed database in the storage layer using the child list stored in each row.

(4) The BrIM mapper returns XML-encoded engineering model to the web server. The web server returns a response enclosing the XML-encoded engineering model and the status code `200` to the client.

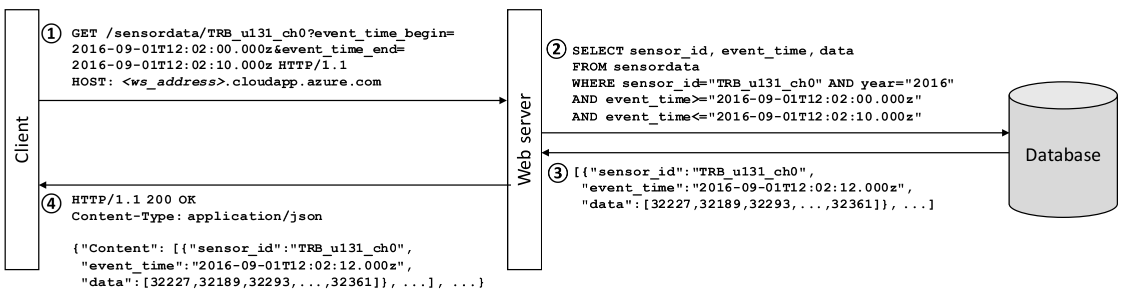
Similarly, as shown in Figure 4.5, an engineering analysis application can retrieve sensor data from the platform by sending an HTTP `GET` request, as follows:

(1) The client sends a `GET` request to the web server with URI (`/sensordata/TRB_u131_ch0`), query parameters (e.g., `event_time_begin` and `event_time_end`), protocol (`HTTP/1.1`) and host (`<ws_address>`).



```
① GET /femodel/TRB HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com

② python cass_to_brimfem.py

③ SELECT * FROM ...
Object (column-oriented schema)
...
SELECT * FROM ...
Object (column-oriented schema)

④ <?xml version="1.0" ?>
<O N="TRB_FEM" T="Project">
  ...
</O>

⑤ HTTP/1.1 200 OK
Content-Type: application/xml

<?xml version="1.0" ?>
<O N="TRB_FEM" T="Project">
  ...
</O>
```

Figure 4.4 Engineering model retrieval from the cyberinfrastructure platform



```
① GET /sensordata/TRB_u131_ch0?event_time_begin=
2016-09-01T12:02:00.000z&event_time_end=
2016-09-01T12:02:10.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com

② SELECT sensor_id, event_time, data
FROM sensordata
WHERE sensor_id="TRB_u131_ch0" AND year="2016"
AND event_time>="2016-09-01T12:02:00.000z"
AND event_time<="2016-09-01T12:02:10.000z"

③ [{"sensor_id":"TRB_u131_ch0",
  "event_time":"2016-09-01T12:02:12.000z",
  "data":[32227,32189,32293,...,32361]}, ...]

④ HTTP/1.1 200 OK
Content-Type: application/json

{"Content": [{"sensor_id":"TRB_u131_ch0",
 "event_time":"2016-09-01T12:02:12.000z",
 "data":[32227,32189,32293,...,32361]}, ...], ...}
```

Figure 4.5 Sensor data retrieval from the cyberinfrastructure platform

(2) The web server sends a `SELECT` query corresponding to the query parameters to the database.
(3) The database returns query result to the web server.
(4) The web server returns a response enclosing JSON-encoded query result and the status code `200` to the client.

4.4.2 Service composition

This section describes how web services offered by the cyberinfrastructure platform can be used for developing and integrating SHM applications. Web service composition refers to the process of combining different web services to provide a new service that carries out composite functions [123]. Standardized web services can be efficiently composed. Different methods have been suggested for composition of RESTful web services [67, 124, 125, 126]. For visual demonstration purpose, this study adopts the approach in [126] that uses JOpera [127], a visual composition language. JOpera describes control flow and data flow between programs using a graphical model [128]. Each node of the graph represents either a program, an input of a program, or an output of a program, while each edge of the graph represents a control flow or data flow between nodes. Each program performs a function, such as web service invocation, script execution and HTML document creation. The following briefly describes two examples to illustrate the composition of RESTful web services implemented on the cyberinfrastructure platform.

Figure 4.6 shows the JOpera data flow of the first demonstrative application named `DataRetrievalByLocation`. This application composes two web services Sensor list retrieval and Sensor data retrieval in order to retrieve sensor data measured at a specified location by a specified type of sensor. Here, the hollow arrows describe the input and the output flow of each program, while the solid arrows describe the data flow and control flow between programs. As described in Figure 4.6, the application consists of three programs, namely `SensorListRetrieval`, `SearchByLocation` and `SensorDataRetrieval`, and processes a request in five steps:

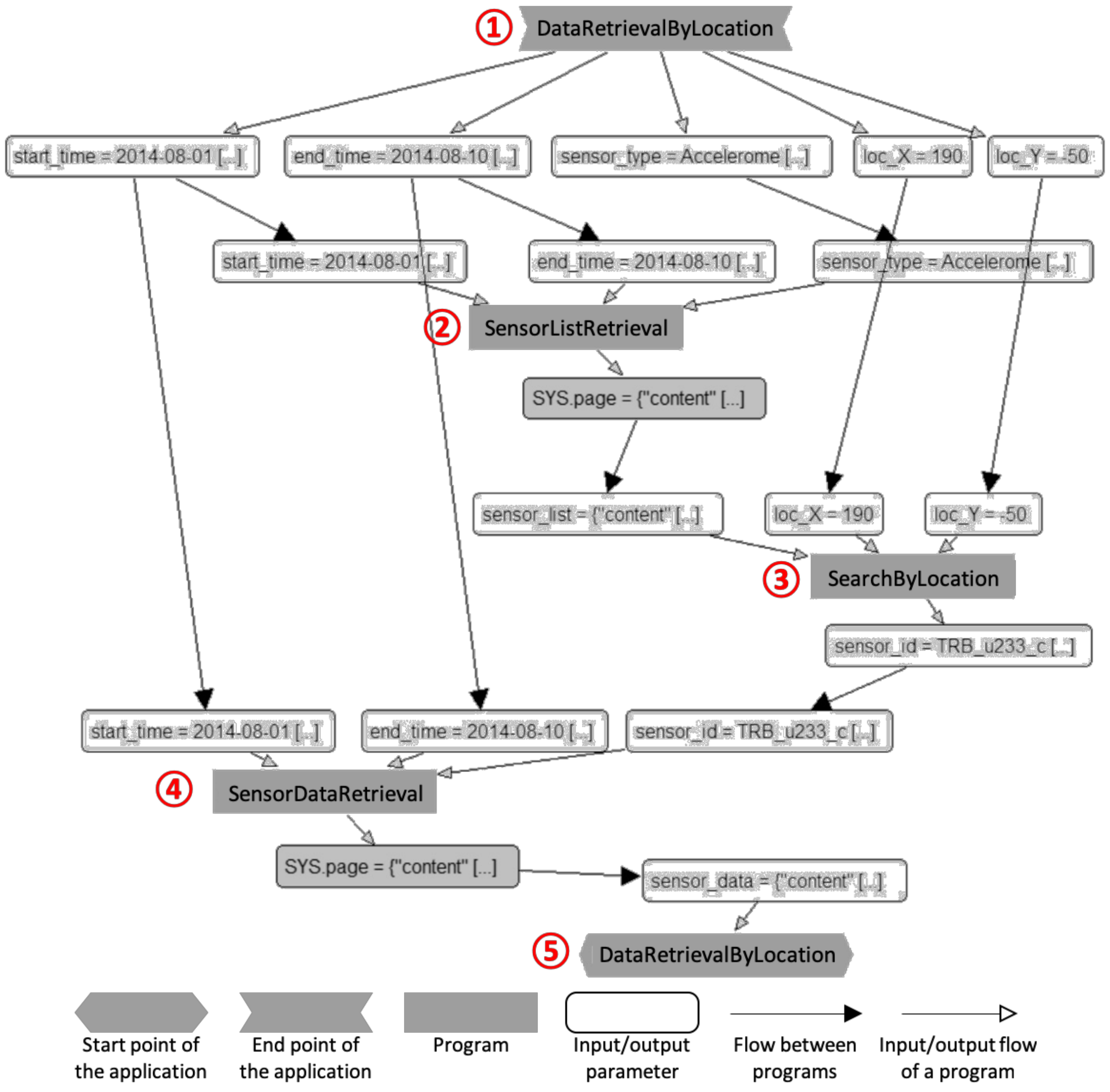

① DataRetrievalByLocation
start_time = 2014-08-01 [...]
end_time = 2014-08-10 [...]
sensor_type = Accelerome [...]
loc_X = 190
loc_Y = -50
start_time = 2014-08-01 [...]
end_time = 2014-08-10 [...]
sensor_type = Accelerome [...]
② SensorListRetrieval
SYS.page = {"content" [...]
sensor_list = {"content" [...]
loc_X = 190
loc_Y = -50
③ SearchByLocation
sensor_id = TRB_u233_c [...]
start_time = 2014-08-01 [...]
end_time = 2014-08-10 [...]
sensor_id = TRB_u233_c [...]
④ SensorDataRetrieval
SYS.page = {"content" [...]
sensor_data = {"content" [...]
⑤ DataRetrievalByLocation
Start point of the application
End point of the application
Program
Input/output parameter
Flow between programs
Input/output flow of a program

Figure 4.6 A composite application DataRetrievalByLocation: data flow

(1) The application accepts input arguments including target time period (`start_time`, `end_time`), sensor type (`sensor_type`) and local coordinate (`loc_X` and `loc_Y`) from a client.

(2) The `start_time`, `end_time` and `sensor_type` are passed to the program `SensorListRetrieval` as input parameters. The program invokes the Sensor list retrieval service with the input parameters, and then returns an output parameter `SYS.page` enclosing the retrieved sensor list.

(3) The `loc_X` and `loc_Y` and the `SYS.page` from the previous step are passed to the program `SearchByLocation` as input parameters. The program searches a sensor corresponding to the `loc_X` and `loc_Y` from the sensor list, and then returns `sensor_id` of the searched sensor.

(4) The `start_time`, `end_time` and the `sensor_id` from the previous step are passed to the program `SensorDataRetrieval` as input parameters. The program invokes the Sensor data retrieval service with the input parameters, and then returns an output parameter `SYS.page` enclosing the retrieved sensor data.

(5) Finally, the `SYS.page` from the previous step is passed to the application's output `sensor_data` which is returned to the client.

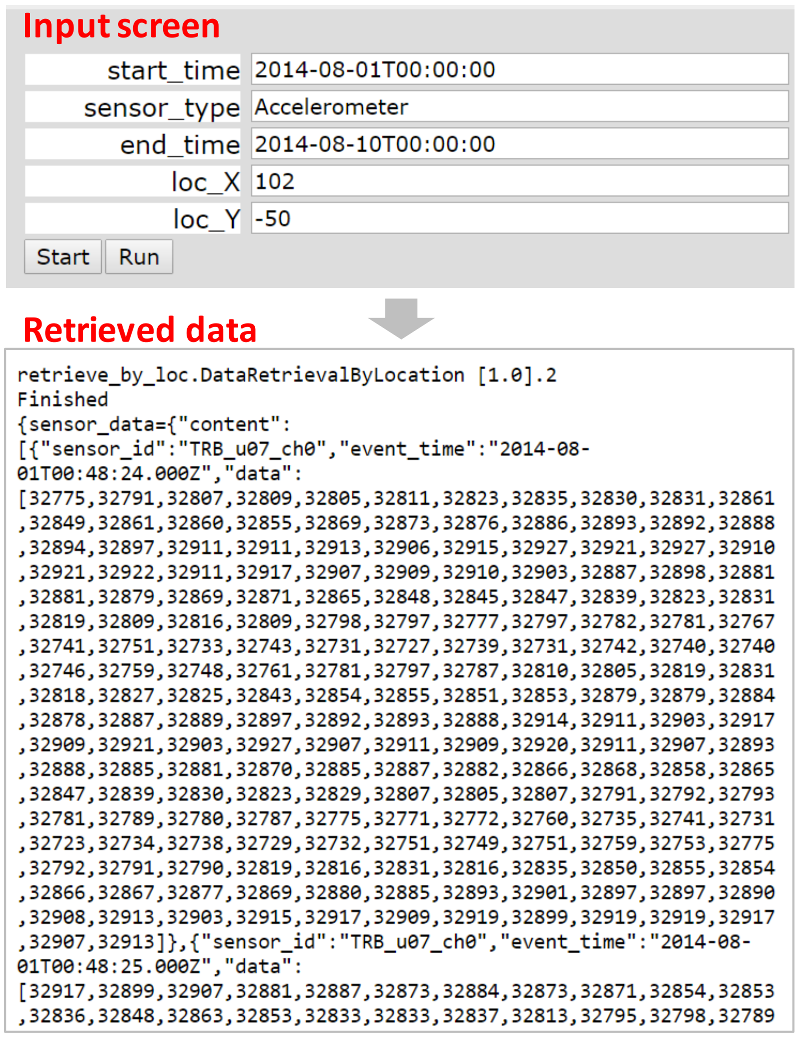
Figure 4.7 shows the retrieved sensor data when the application is executed with the input arguments `2014-08-01T00:00:00`, `2014-08-10T00:00:00`, `Accelerometer`, `102` and `-50`, which correspond to `start_time`, `end_time`, `sensor_type`, `loc_X` and `loc_Y`, respectively.

Figure 4.8 shows the data flow of the second application named `SensorInfoOnMap` that composes an internal web service Sensor information retrieval and an external service Google Map API. Given a sensor's ID, the application shows sensor information at the location of the sensor on the map. As shown in Figure 4.8, the application consists of three programs, including `SensorInfoRetrieval`, `SensorInfoParser` and `MapHandler`, and processes a request in five steps:

(1) The application accepts an input argument `sensor_id` from a client.

(2) The `sensor_id` is passed to the `SensorInfoRetrieval` as an input parameter. The program invokes the Sensor information retrieval service with the input parameter, and then returns an output parameter `SYS.page` enclosing the sensor information.

(3) The output parameter of the previous step (i.e., `SYS.page`) is passed to the `SensorInfoParser` as an input parameter. The program parses the sensor



start_time	2014-08-01T00:00:00
sensor_type	Accelerometer
end_time	2014-08-10T00:00:00
loc_X	102
loc_Y	-50

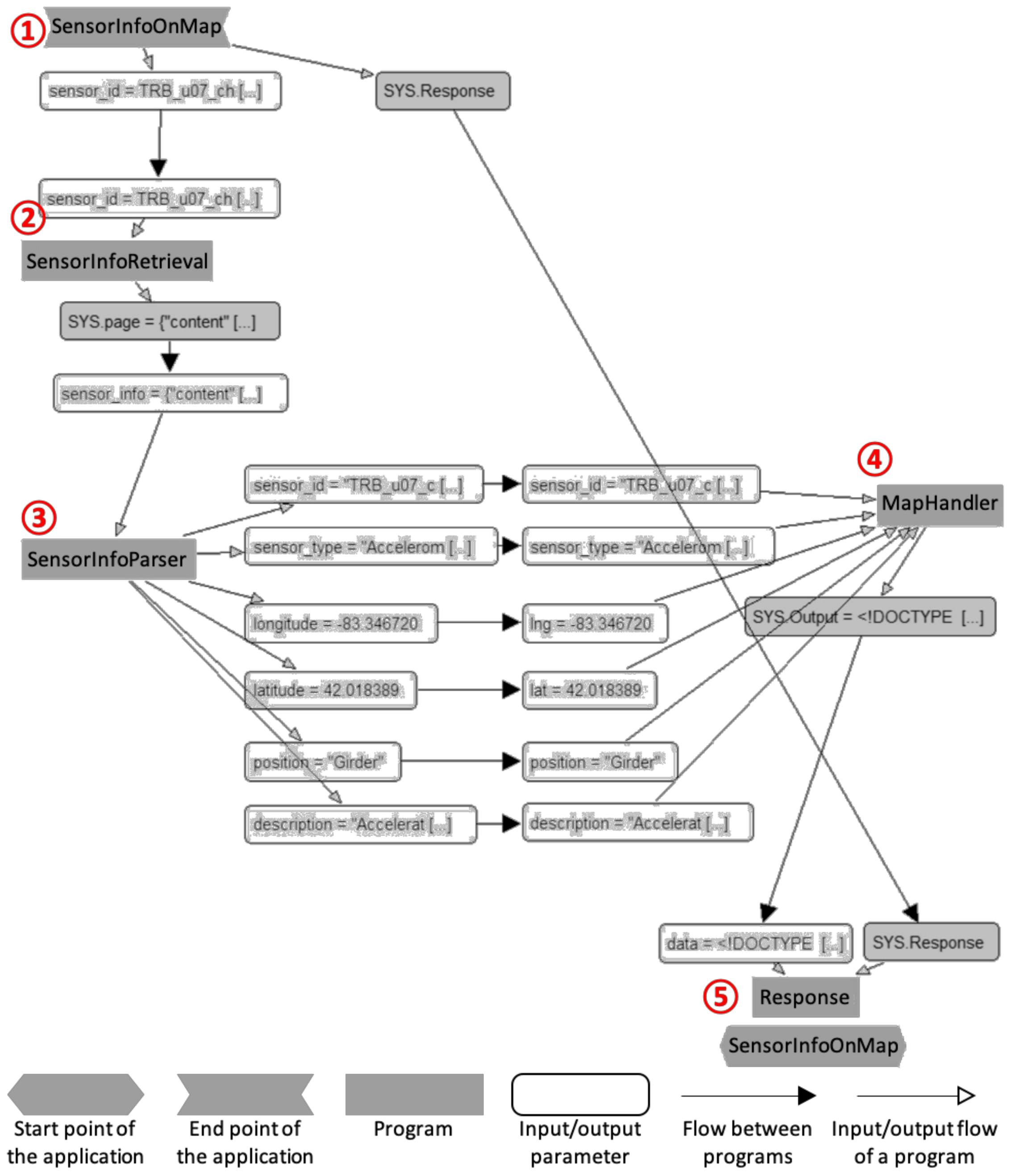
```
retrieve_by_loc.DataRetrievalByLocation [1.0].2
Finished
{sensor_data={"content":
[{"sensor_id":"TRB_u07_ch0","event_time":"2014-08-
01T00:48:24.000Z","data":
[32775,32791,32807,32809,32805,32811,32823,32835,32830,32831,32861
,32849,32861,32860,32855,32869,32873,32876,32886,32893,32892,32888
,32894,32897,32911,32911,32913,32906,32915,32927,32921,32927,32910
,32921,32922,32911,32917,32907,32909,32910,32903,32887,32898,32881
,32881,32879,32869,32871,32865,32848,32845,32847,32839,32823,32831
,32819,32809,32816,32809,32798,32797,32777,32797,32782,32781,32767
,32741,32751,32733,32743,32731,32727,32739,32731,32742,32740,32740
,32746,32759,32748,32761,32781,32797,32787,32810,32805,32819,32831
,32818,32827,32825,32843,32854,32855,32851,32853,32879,32879,32884
,32878,32887,32889,32897,32892,32893,32888,32914,32911,32903,32917
,32909,32921,32903,32927,32907,32911,32909,32920,32911,32907,32893
,32888,32885,32881,32870,32885,32887,32882,32866,32868,32858,32865
,32847,32839,32830,32823,32829,32807,32805,32807,32791,32792,32793
,32781,32789,32780,32787,32775,32771,32772,32760,32735,32741,32731
,32723,32734,32738,32729,32732,32751,32749,32751,32759,32753,32775
,32792,32791,32790,32819,32816,32831,32816,32835,32850,32855,32854
,32866,32867,32877,32869,32880,32885,32893,32901,32897,32897,32890
,32908,32913,32903,32915,32917,32909,32919,32899,32919,32919,32917
,32907,32913]},{"sensor_id":"TRB_u07_ch0","event_time":"2014-08-
01T00:48:25.000Z","data":
[32917,32899,32907,32881,32887,32873,32884,32873,32871,32854,32853
,32836,32848,32863,32853,32833,32833,32837,32813,32795,32798,32789
```

Figure 4.7 A composite application DataRetrievalByLocation: Execution example

information and returns extracted data entities including sensor's ID, coordinate, type, position and description.

(4) The data entities from the previous step are passed to the `MapHandler`. The program returns an HTML document that displays extracted data entities on the Google Map by using the Google Map JavaScript API [129].

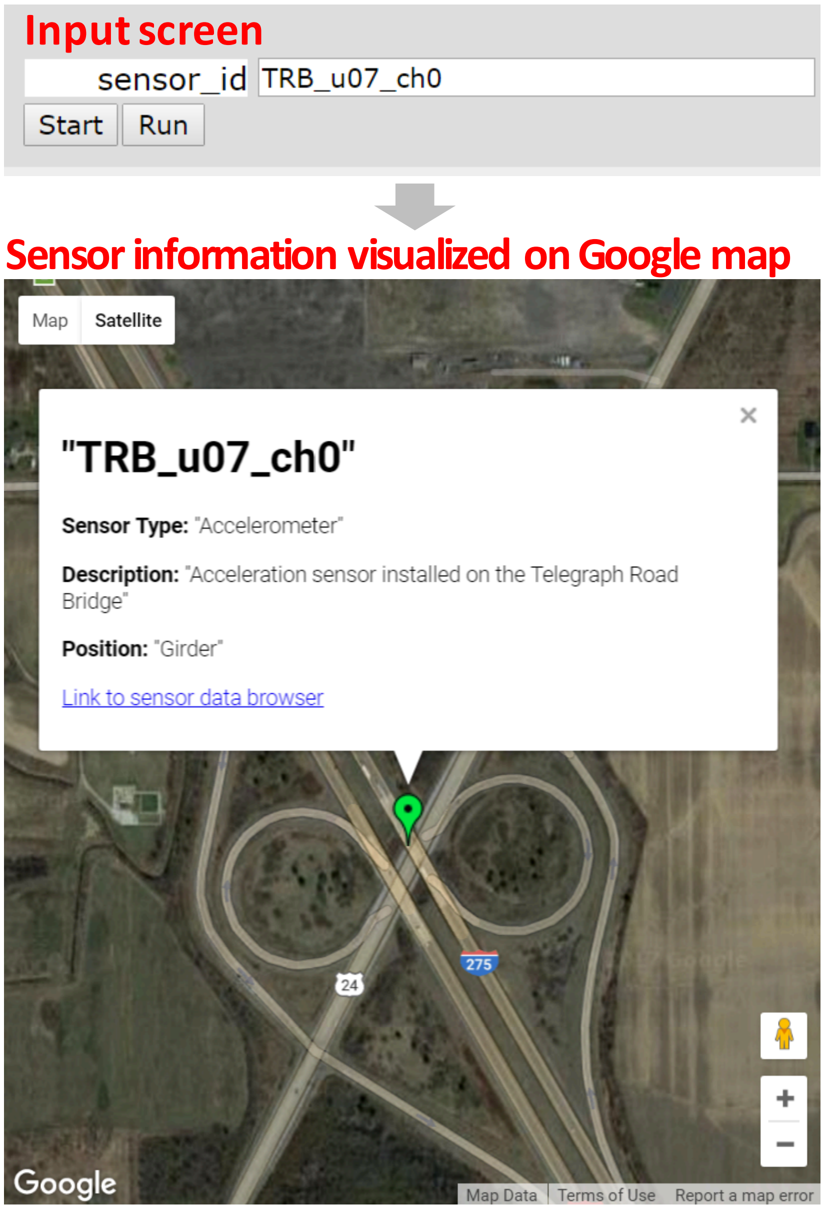
(5) Finally, the HTML document from the previous step is passed to the application's output that can be visualized by a web browser.



①
SensorInfoOnMap
sensor_id = TRB_u07_ch [...]
SYS.Response
sensor_id = TRB_u07_ch [...]
②
SensorInfoRetrieval
SYS.page = {"content" [...]
sensor_info = {"content" [...]
③
SensorInfoParser
sensor_id = "TRB_u07_c [...]
sensor_type = "Accelerom [...]
longitude = -83.346720
latitude = 42.018389
position = "Girder"
description = "Accelerat [...]
sensor_id = "TRB_u07_c [...]
sensor_type = "Accelerom [...]
lng = -83.346720
lat = 42.018389
position = "Girder"
description = "Accelerat [...]
④
MapHandler
SYS.Output = <!DOCTYPE [...]
data = <!DOCTYPE [...]
SYS.Response
⑤
Response
SensorInfoOnMap
Start point of the application
End point of the application
Program
Input/output parameter
Flow between programs
Input/output flow of a program

Figure 4.8 A composite application SensorInfoOnMap: Data flow

Figure 4.9 shows the result of the `SensorInfoOnMap` application with an input argument `TRB_u07_ch0`, where the sensor information and the location marker are displayed on the Google map.



sensor_id TRB_u07_ch0

"TRB_u07_ch0"

Sensor Type: "Accelerometer"

Description: "Acceleration sensor installed on the Telegraph Road Bridge"

Position: "Girder"

Link to sensor data browser

Figure 4.9 A composite application SensorInfoOnMap: Execution example

4.5 Cloud-based Implementation

This section describes the cloud-based implementation of the proposed cyberinfrastructure platform [73, 74]. The cyberinfrastructure platform needs to be scalable to handle large and increasing amount of sensor data. The use of cloud computing enables scalability of the cyberinfrastructure platform. The prototype implementation leverages Infrastructure as a Service (IaaS) of cloud computing to enable portability. In addition, a hybrid cloud-based decentralized data management is discussed to facilitate information sharing.

4.5.1 Cloud computing environment

The past decade has seen wide adoption of cloud computing in many large-scale industrial applications, particularly in the Internet of Things (IoT) and big data arena. Advances in cloud computing provide highly scalable and accessible computing environment, lessen the burdens on the deployment, operation, maintenance and management of computational resources, and reduce the cost [7, 57, 58, 59]. Many state-of-the-art data management platforms take advantage of cloud computing to allow communication and data sharing among physical systems, sensors, software applications and users. Using cloud computing, an application, such as an engineering cyberinfrastructure platform, can be easily scaled according to demand and optimized for usages of computing and storage resources.

Cloud computing services are typically categorized into three service models [56]: (1) Software as a Service (SaaS) that provides applications and web services to end users, (2) Platform as a Service (PaaS) that provides runtime and database supports, and (3) Infrastructure as a Service (IaaS) that provides the basic computing utilities including network, processor and storage. As depicted in Figure 4.10, the cloud-based engineering cyberinfrastructure platform acts as PaaS and SaaS that utilize the computing infrastructures and platforms (i.e., IaaS and PaaS) for hosting the data management and application services.

IaaS utilities are typically offered in the form of virtual machines (VMs). A VM is a virtualized computing system that emulates the underlying architecture of a physical computer and offers the same functionalities of the physical computer [130]. A VM can be provisioned and configured in minutes and be managed through cloud interfaces offered by a cloud vendor. For example, Figure 4.11(a) shows the web portal interface of the Microsoft Azure cloud platform [131] that shows the information about a VM such as its

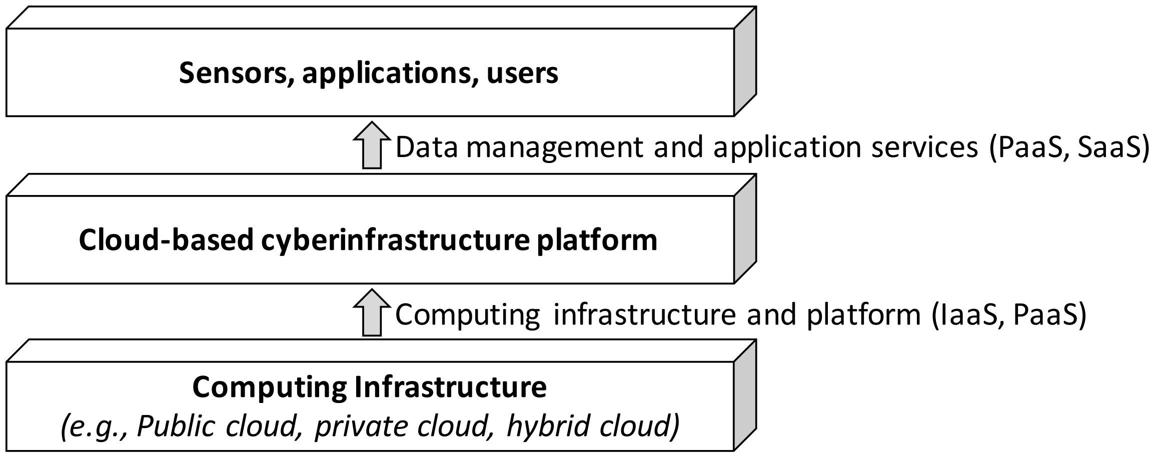
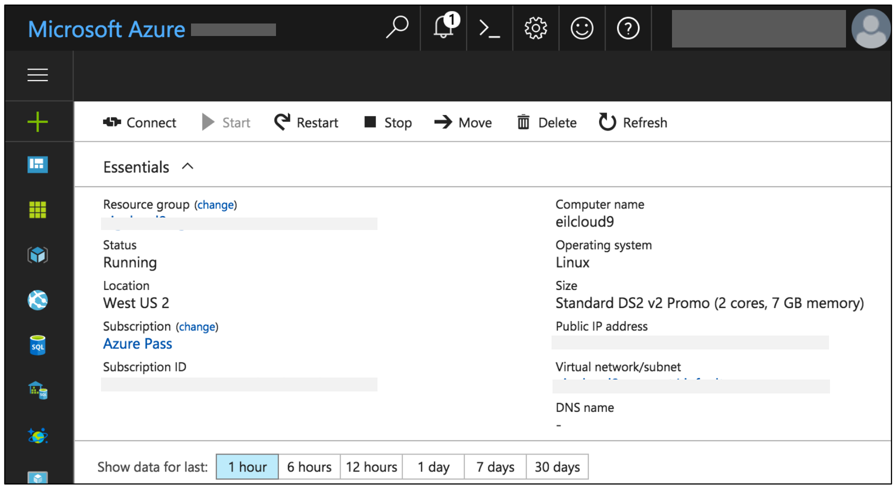

Sensors, applications, users
Data management and application services (PaaS, SaaS)
Cloud-based cyberinfrastructure platform
Computing infrastructure and platform (IaaS, PaaS)
Computing Infrastructure
(e.g., Public cloud, private cloud, hybrid cloud)

Figure 4.10 A model of cloud computing for SHM

name, status, operating system (OS) and size. Once provisioned, a VM can be accessed via standard network protocols, such as Secure Shell (SSH) and Secure Copy Protocol (SCP). Figure 4.11(b) shows the shell interface of a VM on the Azure cloud platform accessed via the SSH protocol. Similar to using a remote server, a VM can be used to gain access to the computing resources and software tools. The proposed cyberinfrastructure platform utilizes VMs to access the computational services, such as distributed database, web servers and engineering software applications.

The IaaS utilities can be scaled both vertically (by increasing capability of a VM) and horizontally (by adding new VMs) on demand. While vertical scalability is limited to the maximum capability of a single VM, the horizontal scalability is nearly unlimited since cloud vendors allow adding as many VMs as needed. Figure 4.11(c), for example, shows that multiple VMs are deployed as needed on the Azure cloud platform. To take advantage of the scalability of an IaaS utility, thereby enabling scalable SHM data management, the cloud-based cyberinfrastructure platform is designed to run on a distributed computing environment such that new VMs can be dynamically added on demand. For example, the cyberinfrastructure platform adopts a NoSQL database system which can be effectively executed on multiple VMs to support distributed data management.



Resource group (change)
[redacted]

Status
Running

Location
West US 2

Subscription (change)
Azure Pass

Subscription ID
[redacted]

Computer name
eilcloud9

Operating system
Linux

Size
Standard DS2 v2 Promo (2 cores, 7 GB memory)

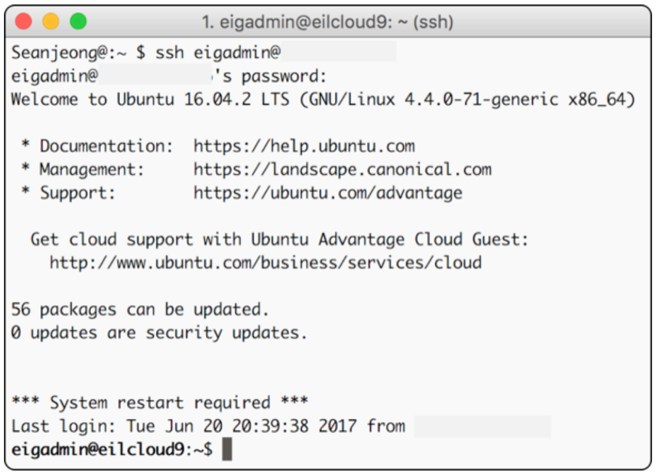
Public IP address
[redacted]

Virtual network/subnet
[redacted]

DNS name
-

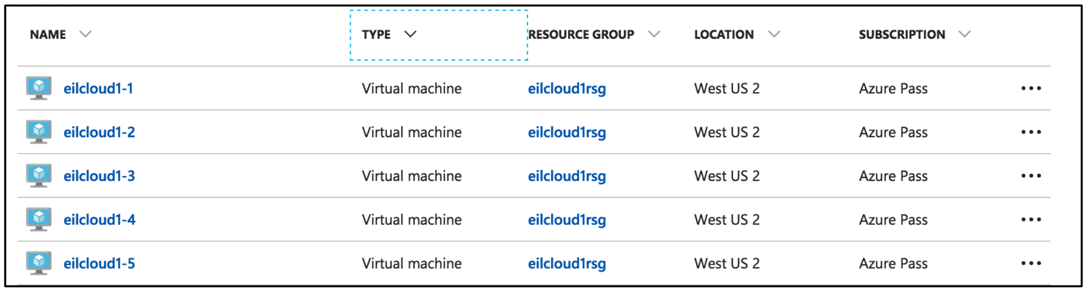
Show data for last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

(a) Web-based cloud interface



```
Seanjeong@:~ $ ssh eigadmin@[redacted]
eigadmin@[redacted]'s password:
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-71-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

56 packages can be updated.
0 updates are security updates.


*** System restart required ***
Last login: Tue Jun 20 20:39:38 2017 from [redacted]
eigadmin@eilcloud9:~$
```

(b) Shell interface



NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION	
eilcloud1-1	Virtual machine	eilcloud1rsg	West US 2	Azure Pass	•••
eilcloud1-2	Virtual machine	eilcloud1rsg	West US 2	Azure Pass	•••
eilcloud1-3	Virtual machine	eilcloud1rsg	West US 2	Azure Pass	•••
eilcloud1-4	Virtual machine	eilcloud1rsg	West US 2	Azure Pass	•••
eilcloud1-5	Virtual machine	eilcloud1rsg	West US 2	Azure Pass	•••

(c) List of virtual machines deployed on Azure cloud platform

Figure 4.11 Virtual machine created on Microsoft Azure cloud platform

4.5.2 Hybrid cloud-based decentralized data management

Public cloud is the most common and well-known deployment model of cloud computing. Public cloud vendor (e.g., Amazon, Microsoft and Google) owns, manages and operates huge data centers and lend computing resources to customers over the Internet on a pay-per-use basis. Based on resource pooling and virtualization, public cloud customers can easily create, configure and scale computing resources on public cloud without the hassle of managing and operating server hardware. Given the scalability, flexibility and reduced maintenance effort, public cloud can be a viable alternative to on-premise server. In civil infrastructure monitoring practice, however, there is also still a desire to maintain an on-premise server to manage sensitive data, which civil infrastructure managers do not want to upload to a public cloud operated by third-party vendors due to security concern. This demand remains to be an issue that impedes the adoption of cloud computing for civil infrastructure monitoring systems.

Hybrid cloud is another cloud deployment model where heterogeneous cloud infrastructures (e.g. public cloud operated by public cloud vendor and on-premise private cloud operated by private company) are bound together [56]. Hybrid cloud is useful to optimize computing resources and to protect privacy of data. More specifically, a company can adopt hybrid cloud for outsourcing the management of non-sensitive, voluminous data to the public cloud, while managing sensitive data and applications within the private cloud [132, 133]. Therefore, hybrid cloud can be a suitable computing infrastructure for cyberinfrastructure platform for civil infrastructure monitoring.

Figure 4.12 depicts the conceptual framework of the hybrid-cloud based implementation of cyberinfrastructure platform. Unlike the on-premise or public cloud-based platforms, hybrid cloud-based platform can distribute data and applications over multiple locations (e.g., public cloud and private cloud) according to their characteristic, such as volume, sensitivity, privacy and ownership. In the proposed platform, public cloud system is employed for the management of voluminous sensor data which requires high scalability

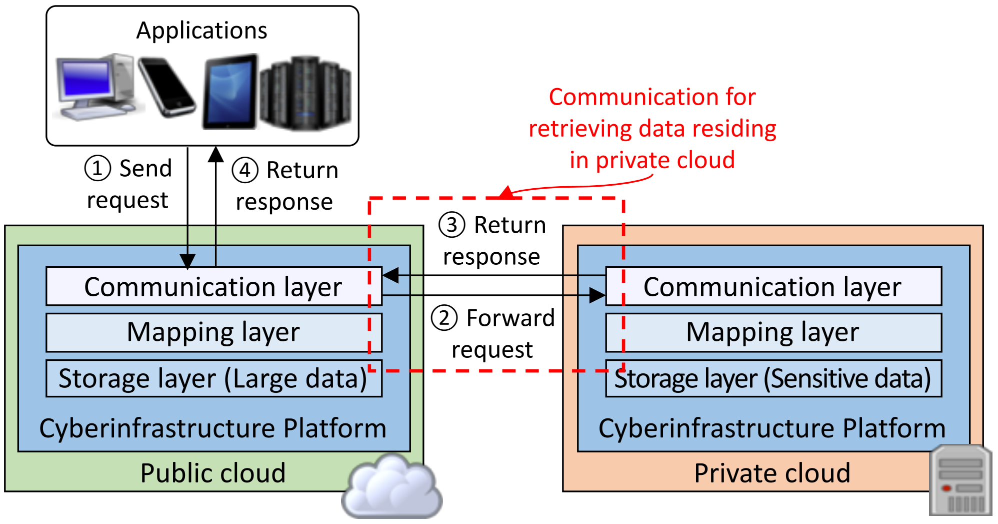

Applications
① Send request
④ Return response
Communication for retrieving data residing in private cloud
③ Return response
② Forward request
Communication layer
Mapping layer
Storage layer (Large data)
Cyberinfrastructure Platform
Public cloud
Communication layer
Mapping layer
Storage layer (Sensitive data)
Cyberinfrastructure Platform
Private cloud

Figure 4.12 A framework of hybrid cloud-based implementation of cyberinfrastructure platform for decentralized data management

and availability, while the private cloud plays a role to handle sensitive data, such as engineering models.

These two separate systems can communicate with each other through web services. For example, when an application program requests sensor data retrieval from the public cloud, the data is directly retrieved from the database on public cloud. On the other hand, when an application program requests for an engineering model from the public cloud, the public cloud forwards the request to the private cloud. The private cloud then retrieves corresponding information from its database and deliver the information to the client via the public cloud. In this way, the hybrid cloud system can abstract the underlying complex structure and provides unified web services to clients.

4.6 Case Scenario: Civil Infrastructure Monitoring Application

This section describes a case scenario using a bridge monitoring system as an example [73]. For demonstration, data collected from the Telegraph Road Bridge (TRB) is used. As discussed in the previous chapter, the data include sensor data, traffic video image data, geometric model, engineering model and sensor information. A prototype cyberinfrastructure platform is implemented using the VMs provisioned on the Microsoft Azure cloud computing service, as well as a private server. Table 4.3 summarizes the list of computers composing the prototype cyberinfrastructure platform and their role.

4.6.1 Automated data store and retrieval

Two automated data store applications are developed using the cyberinfrastructure's web services to archive sensor data and video image data, respectively. Figure 4.13 shows the workflow of the first application that runs on onsite computers and transmits sensor data from an onsite computer to the cloud-based cyberinfrastructure platform. When new data is transmitted from the sensor network to the onsite computer, the application records the list of the new data files and labels them as "un-transmitted". For an "un-transmitted" file, the application parses the raw data file encoded in DAT file format (see Figure 4.14(a)) into a JSON format (see Figure 4.14(b)) that the "Sensor data store" service of the cyberinfrastructure can read. The application then invokes the "Sensor data store" service with the parsed sensor data. If the service returns a status code `200`, the application changes the label of the data file to "transmitted". Otherwise, the application retries invoking the "Sensor data store" service up to `N` times (i.e., a predefined maximum number of retries) to ensure the transmission is done properly. This parsing and storing process is repeated

Table 4.3 Specification and role of cloud virtual machines and private server composing cyberinfrastructure platform

Type	**Spec**	**Quantity**	**Role**
Public cloud VM	Azure Standard_A2m_v2 (2 cores, 16 GB memory)	5	Distributed database for large data
	Azure Standard DS2 v2 (2 cores, 7 GB memory)	3	Web server
Private server	Dell PowerEdge T620	1	Local data storage for sensitive data

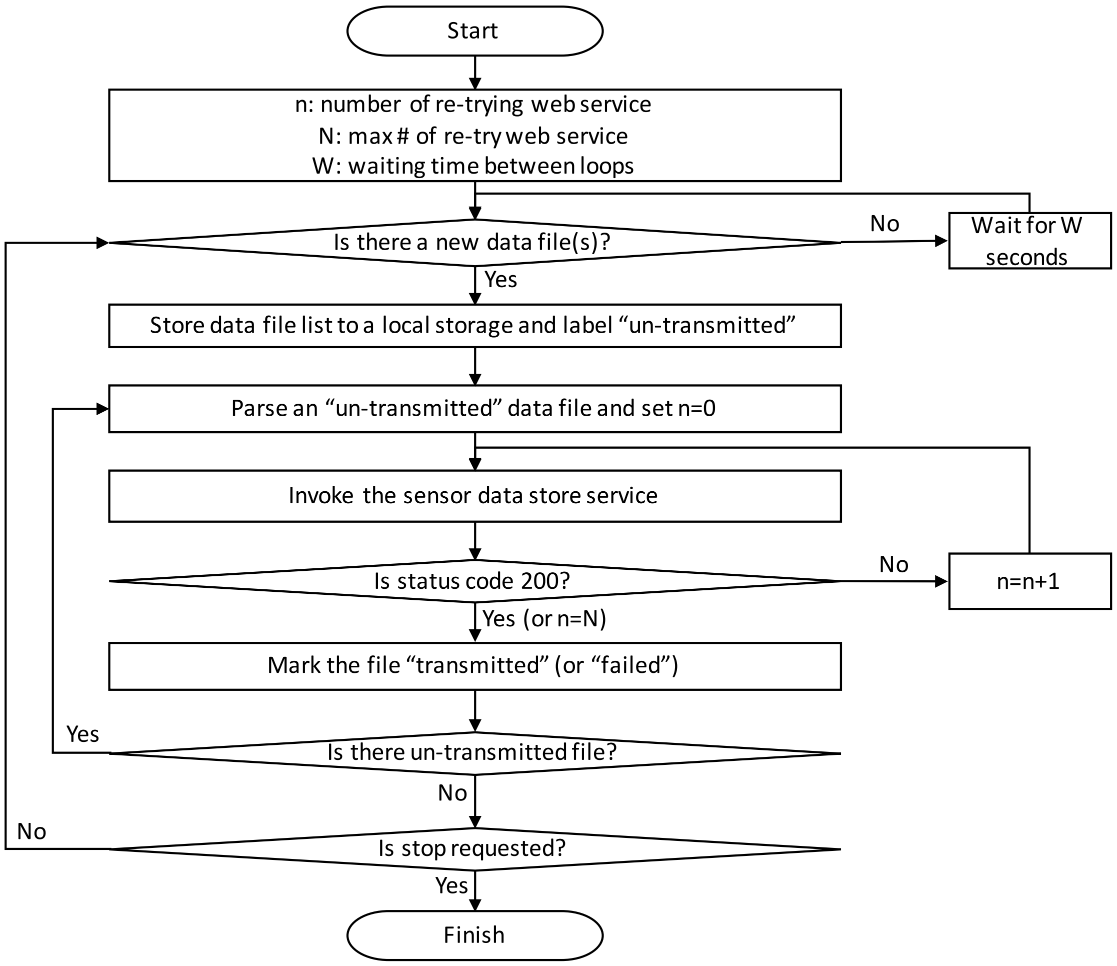
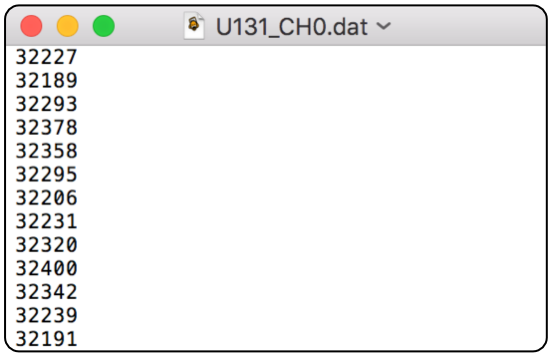
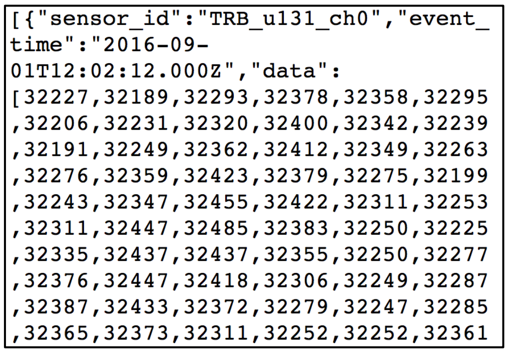

Start
n: number of re-trying web service
N: max # of re-try web service
W: waiting time between loops
Is there a new data file(s)?
No
Wait for W seconds
Yes
Store data file list to a local storage and label "un-transmitted"
Parse an "un-transmitted" data file and set n=0
Invoke the sensor data store service
Is status code 200?
No
n=n+1
Yes (or n=N)
Mark the file "transmitted" (or "failed")
Yes
Is there un-transmitted file?
No
No
Is stop requested?
Yes
Finish

Figure 4.13 Workflow: application for data store automation



```
32227
32189
32293
32378
32358
32295
32206
32231
32320
32400
32342
32239
32191
```

(a) Raw data file



```
[{"sensor_id":"TRB_u131_ch0","event_
time":"2016-09-
01T12:02:12.000Z","data":
[32227,32189,32293,32378,32358,32295
,32206,32231,32320,32400,32342,32239
,32191,32249,32362,32412,32349,32263
,32276,32359,32423,32379,32275,32199
,32243,32347,32455,32422,32311,32253
,32311,32447,32485,32383,32250,32225
,32335,32437,32437,32355,32250,32277
,32376,32447,32418,32306,32249,32287
,32387,32433,32372,32279,32247,32285
,32365,32373,32311,32252,32252,32361
```

(b) Parsed sensor data

Figure 4.14 Sensor data file

until there are no “un-transmitted” data files in the list. Unless the application is forced to be stopped, the application repeats the whole process every `W` seconds (i.e., a predefined waiting duration between the repeats). Since the application keeps track of data transmission status of data files, data losses due to an unstable network connection can be minimized.

Figure 4.15 shows the workflow of the second application for collecting traffic video images from an external data source (i.e., MDOT’s traffic monitoring system) and the manner by which images are archived along with the camera ID and timestamp. The application first accesses the data source to locate the dynamic Uniform Resource Locators (URLs) of video image files. Once the URLs are found, the application fetches the video image files and converts them to the BLOB format. The application then transmits the

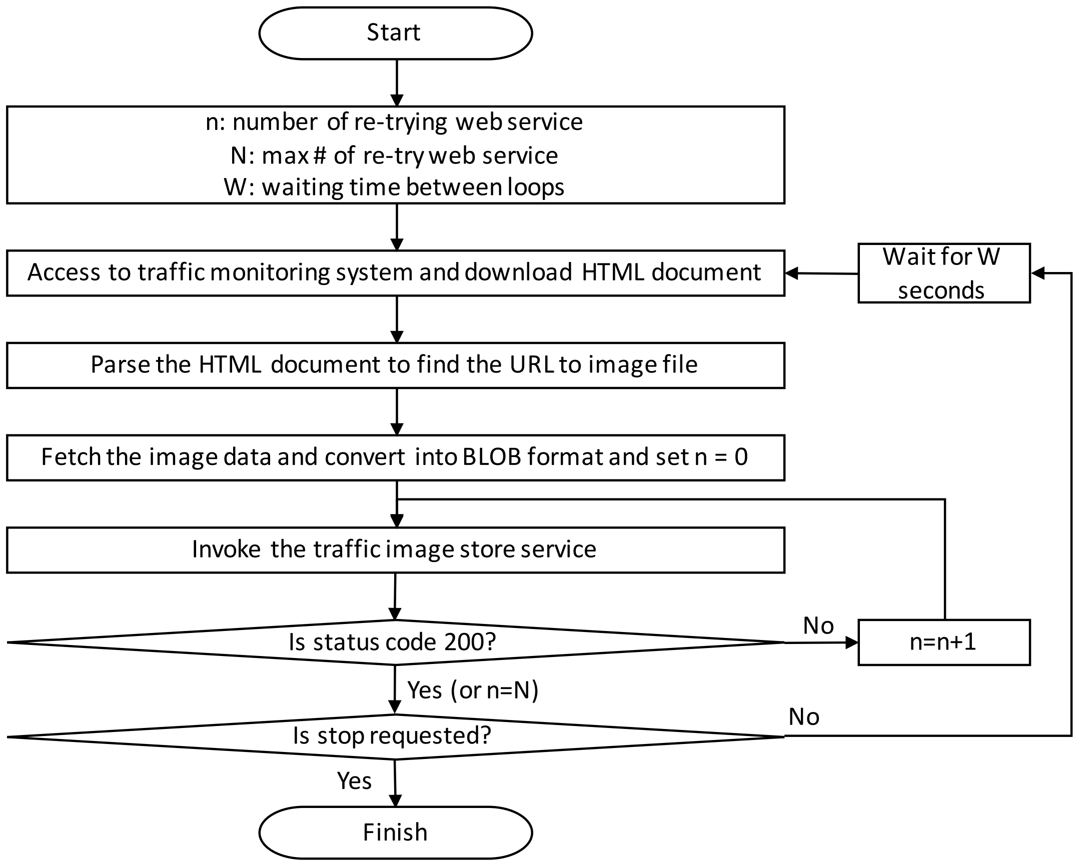

Start
n: number of re-trying web service
N: max # of re-try web service
W: waiting time between loops
Access to traffic monitoring system and download HTML document
Wait for W seconds
Parse the HTML document to find the URL to image file
Fetch the image data and convert into BLOB format and set n = 0
Invoke the traffic image store service
Is status code 200?
No
n=n+1
Yes (or n=N)
Is stop requested?
No
Yes
Finish

Figure 4.15 Workflow: traffic video image collecting application

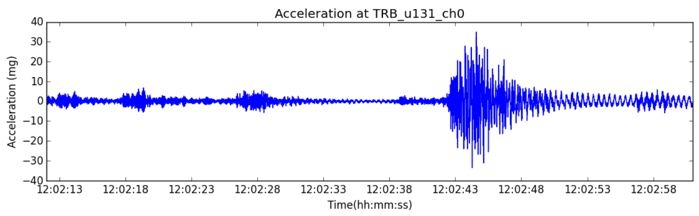
BLOB data to the cyberinfrastructure by invoking the "Traffic image store" service. The application service repeats this process every two seconds corresponding to the time interval between new images in the MDOT's traffic monitoring system.

Data stored in the cloud-based cyberinfrastructure platform can be easily retrieved using data retrieval services. For example, Figure 4.16(a) shows a request for the "Sensor data retrieval" service with query conditions including sensor ID (`TRB_u131_ch0`), `event_time_begin` (`2016-09-01T12:02:00.000z`) and `event_time_end` (`2016-09-12T12:02:10.000z`). Similarly, Figure 4.16(b) shows a request for the "Traffic image retrieval" service with query conditions including camera ID (`Telegraph2`), `event_time_begin` (`2016-08-18T18:01:00.000z`) and event_time_end (`2016-08-18T18:01:20.000z`).

4.6.2 Data integration and utilization

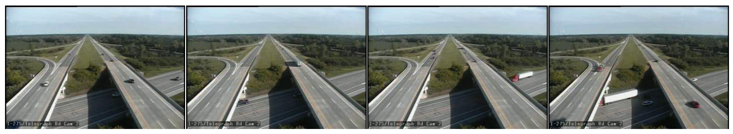
The advantages of the cloud-based cyberinfrastructure platform are its ability to support easy access, integration and utilization of SHM data. This section presents a demonstrative example application developed to extract patterns from heterogeneous data (e.g., structural

```
GET /sensordata/TRB_u131_ch0?
event_time_begin=2016-09-
01T12:02:00.000z
&event_time_end=2016-09-
01T12:02:10.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
```



(a) Sensor data retrieved by invoking the sensor data retrieval service

```
GET /imagedata/Telegraph2?
event_time_begin=2016-08-
18T18:01:00.000z
&event_time_end=2016-08-
18T18:01:20.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
```



(b) Traffic images retrieved by invoking the traffic image retrieval service

Figure 4.16 Data retrieval using web services

sensing data and environmental data) to find the relationships between the modal frequencies derived from sensor data with temperature measurements. The cyberinfrastructure platform enables machine-to-machine communication so that the workflow can be fully automated with applications written using programming scripts, for example, Python. As shown in the conceptual workflow described in Figure 4.17, the application accesses the cyberinfrastructure platform via web services, retrieve acceleration data and temperature data, and performs analyses. The application service comprises of the following steps:

(1) The application reads the input arguments `StartTime` and `EndTime` specifying the target time period which typically includes multiple data acquisition events.

(2) The application service retrieves the accelerometer list for a data acquisition event by submitting request as a web service to the "Sensor list retrieval service" as shown in Figure 4.18(a).

(3) The application service retrieves the acceleration data collected from the data acquisition event for each accelerometer in the list by submitting a request (as

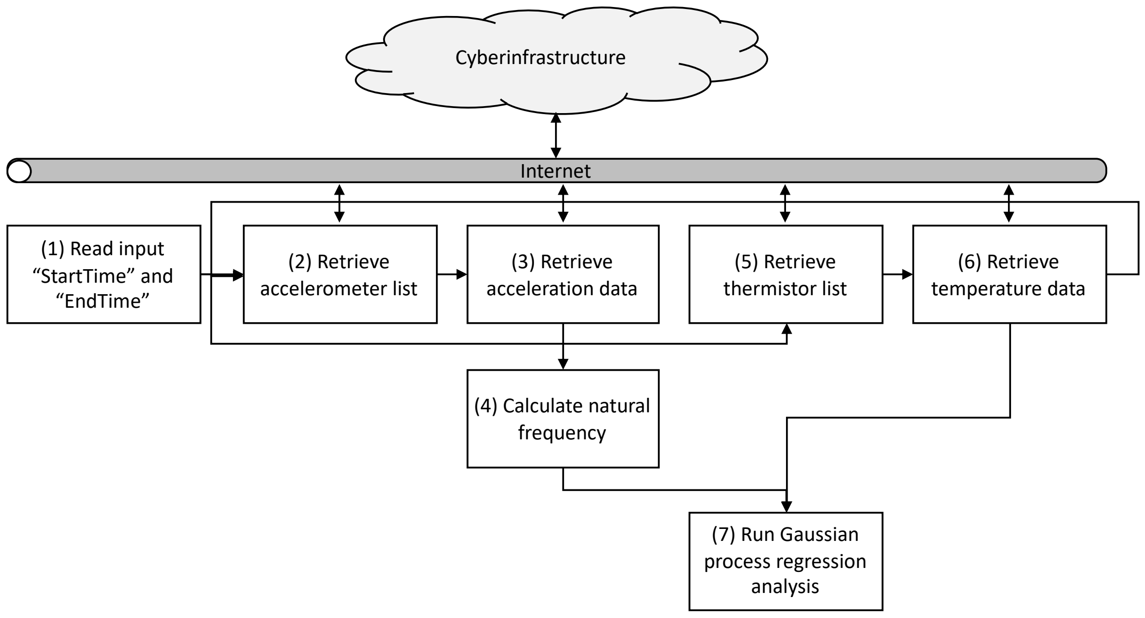

Cyberinfrastructure
Internet
(1) Read input "StartTime" and "EndTime"
(2) Retrieve accelerometer list
(3) Retrieve acceleration data
(5) Retrieve thermistor list
(6) Retrieve temperature data
(4) Calculate natural frequency
(7) Run Gaussian process regression analysis

Figure 4.17 A workflow for relating structural behavior with temperature data

another web service) to the "Sensor data retrieval" service, as shown in Figure 4.18(b).

(4) The application executes the subspace identification module service [134] to compute the modal frequency from the retrieved acceleration data.

(5) The application service retrieves the thermistor list for the data acquisition event by submitting a request (as a web service) to the "Sensor list retrieval service" as shown in Figure 4.18(c).

(6) The application service retrieves temperature data collected from the data acquisition event by submitting a request to the "Sensor data retrieval" service, as shown in Figure 4.18(d).

```
GET /sensor?sensorType=Accelerometer&
install=2014-08-01T00:00:00.000z&
remove=2014-08-01T02:02:00.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
              <-- Rest is omitted -->
```

(a) HTTP request for accelerometer list retrieval

```
GET /sensordata/TRB_u07_ch0?
event_time_begin=2014-08-01T00:00:00.000z&
event_time_end=2014-08-01T02:02:00.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
              <-- Rest is omitted -->
```

(b) HTTP request for acceleration data retrieval

```
GET /sensor?sensorType=Thermistor&
install=2014-08-01T00:00:00.000z&
remove=2014-08-01T02:02:00.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
              <-- Rest is omitted -->
```

(c) HTTP request for thermistor list retrieval

```
GET /sensordata/TRB_u45_ch0?
event_time_begin=2014-08-01T00:00:00.000z&
event_time_end=2014-08-01T02:02:00.000z HTTP/1.1
HOST: <ws_address>.cloudapp.azure.com
              <-- Rest is omitted -->
```

(d) HTTP request for temperature data retrieval

Figure 4.18 HTTP requests and corresponding CQL queries for sensor list and data

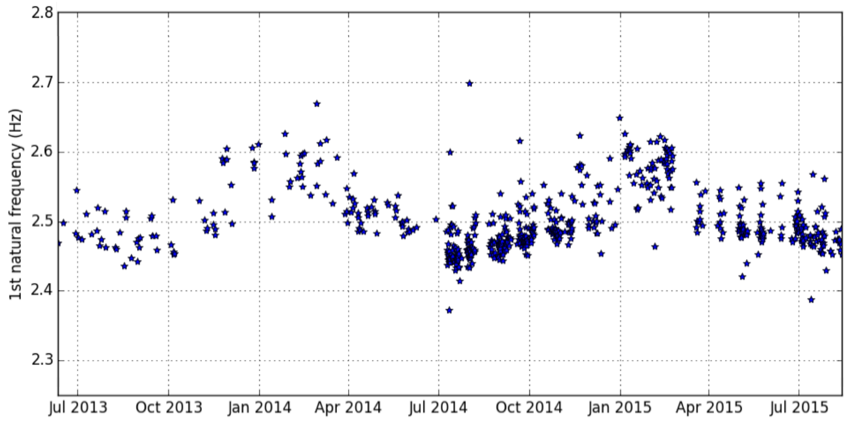
(7) Executing Gaussian Process Regression (GPR) service to find the general pattern on the variation between the fundamental modal frequency and temperature measurements.

The application service repeats step 2 to step 6 by moving on to the next data acquisition event until reaching the `EndTime` of the targeted period. Once modal frequencies and temperature measurements that share the same timestamps are collected, the application service proceeds to step 7 to perform regression analysis.

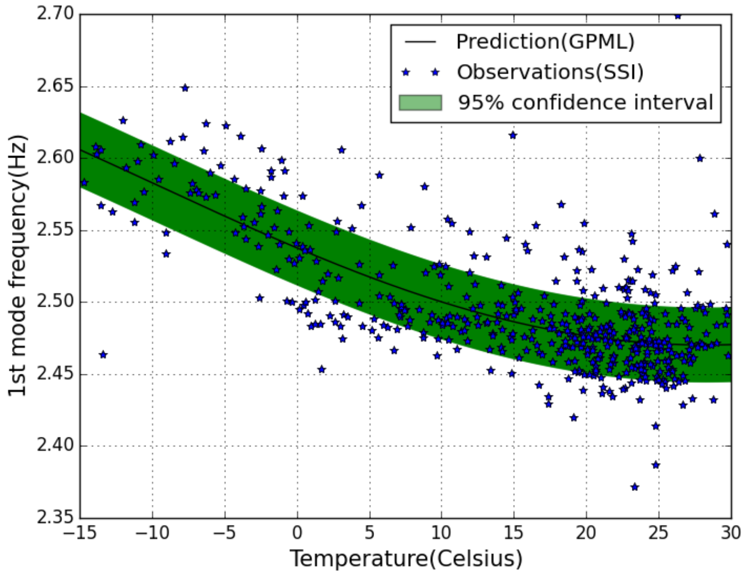
Figure 4.19(a) shows the history of the first modal frequency over the duration from June 2013 to August 2015. The modal frequencies vary with a seasonal trend and tend to increase during the winter and decrease during the summer. As shown in Figure 4.19(b), the GPR analysis result shows a nearly bilinear relation between the modal frequency and temperature. This example application shows that the cyberinfrastructure platform can be used to automate repetitive tasks involving multiple SHM data sets.

4.6.3 Web and mobile user interfaces

To facilitate ubiquitous access to the bridge monitoring information, preliminary web and mobile user interfaces are developed based on the cloud-based cyberinfrastructure platform. A web interface is an interactive program that reads user inputs via a web browser (e.g., Google Chrome), invokes the requested web services and returns a web page displayed on a web browser. The preliminary web interface supports the retrieval of sensor list, sensor data, traffic video images and bridge models. The sensor information retrieval interface (Figure 4.20(a)) allows users to retrieve the sensor list with query parameters (e.g., "Sensor ID", "Sensor type", "Install before" or "Removed after") by invoking the "Sensor list retrieval" service. The sensor data retrieval interface (Figure 4.20(b)) accepts query parameters (e.g., "Sensor ID, "Begin timestamp" and "End timestamp") and returns corresponding sensor data by invoking the "Sensor data retrieval" service. Similarly, the traffic video image retrieval interface (Figure 4.20(c)) accepts query parameters (e.g., "Camera ID", "Begin timestamp" and "End timestamp") and returns corresponding traffic



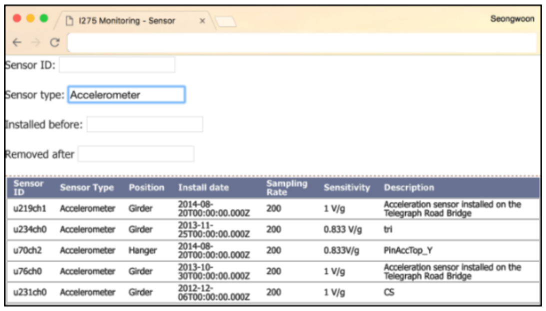
(a) History of first modal frequency (from August 2013 to August 2015)



(b) Gaussian process regression showing the confidence interval of modal frequency according to temperature changes

Figure 4.19 Patterns of modal frequency of the Telegraph Road Bridge

video images by invoking the “Traffic image retrieval” service. The bridge model retrieval interface (Figure 4.20(d)) allows users to download bridge models by invoking either the “Geometric model retrieval” or “Engineering model retrieval” service. Through this interface, users can specify the “Bridge name” and model type (e.g., “GeometricModel”, "FEModel (xml)" and "FEModel (xlsx)"). The bridge models downloaded can be regenerated by proper software tools, such as the OpenBrIM Viewer [81] for geometric models (Figure 4.21(a)) and CSiBridge for engineering models (Figure 4.21(b)).



Sensor ID:

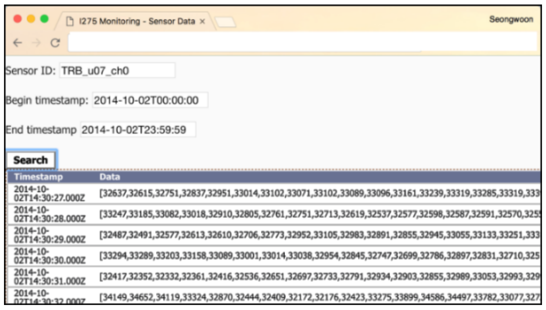
Sensor type: Accelerometer

Installed before:

Removed after

Sensor ID	Sensor Type	Position	Install date	Sampling Rate	Sensitivity	Description
u219ch1	Accelerometer	Girder	2014-08-20T00:00:00.000Z	200	1 V/g	Acceleration sensor installed on the Telegraph Road Bridge
u234ch0	Accelerometer	Girder	2013-11-25T00:00:00.000Z	200	0.833 V/g	tri
u70ch2	Accelerometer	Hanger	2014-08-20T00:00:00.000Z	200	0.833V/g	PinAccTop_Y
u76ch0	Accelerometer	Girder	2013-10-30T00:00:00.000Z	200	1 V/g	Acceleration sensor installed on the Telegraph Road Bridge
u231ch0	Accelerometer	Girder	2012-12-06T00:00:00.000Z	200	1 V/g	CS

(a) Sensor information retrieval



Sensor ID: TRB_u07_ch0

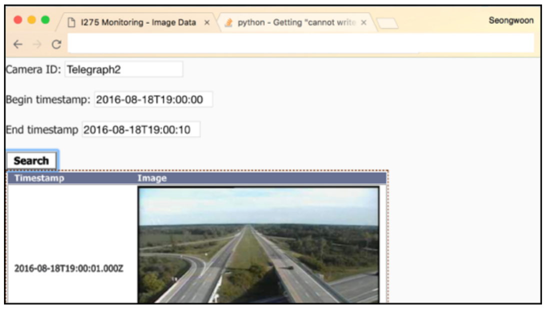
Begin timestamp: 2014-10-02T00:00:00

End timestamp 2014-10-02T23:59:59

Search

Timestamp	Data
2014-10-02T14:30:27.000Z	[32637,32615,32751,32837,32951,33014,33102,33071,33102,33089,33096,33161,33239,33319,33285,33319,333
2014-10-02T14:30:28.000Z	[33247,33185,33082,33018,32910,32805,32761,32751,32713,32619,32537,32577,32598,32587,32591,32570,325
2014-10-02T14:30:29.000Z	[32487,32491,32577,32613,32610,32706,32773,32952,33105,32983,32891,32855,32945,33055,33133,33251,333
2014-10-02T14:30:30.000Z	[33294,33289,33203,33158,33089,33001,33014,33038,32954,32845,32747,32699,32786,32897,32831,32710,325
2014-10-02T14:30:31.000Z	[32417,32352,32332,32361,32416,32536,32651,32697,32733,32791,32934,32903,32855,32989,33053,32993,329
2014-10-02T14:30:32.000Z	[34149,34652,34119,33324,32870,32444,32409,32172,32176,32423,33275,33899,34586,34497,33782,33077,327

(b) Sensor data retrieval



Camera ID: Telegraph2

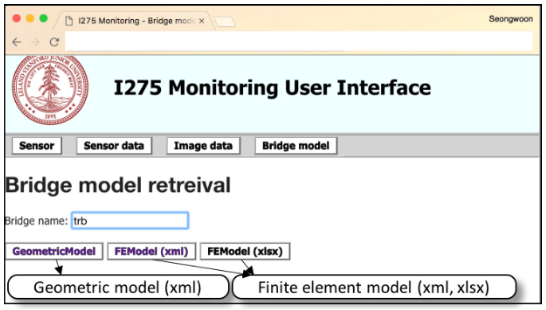
Begin timestamp: 2016-08-18T19:00:00

End timestamp 2016-08-18T19:00:10

Search

Timestamp	Image
2016-08-18T19:00:01.000Z	

(c) Traffic image retrieval

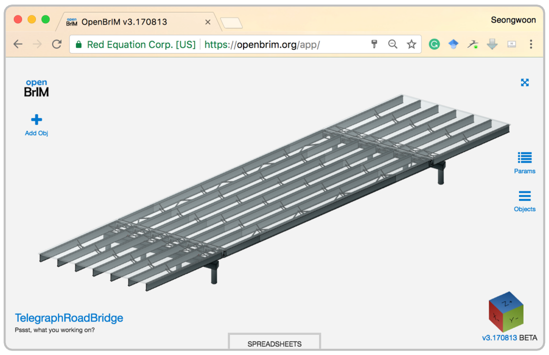


Bridge model retreival

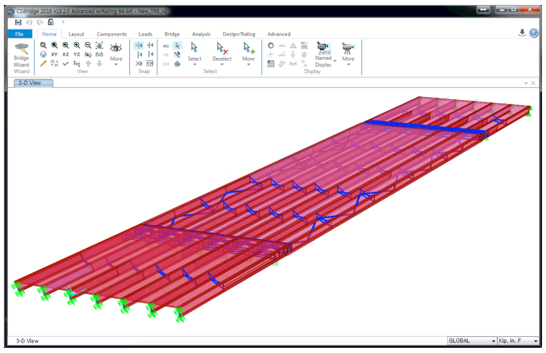
Bridge name: trb

(d) Bridge model retrieval

Figure 4.20 Prototype web-based user interface



(a) Geometric model visualized by OpenBrIM Viewer



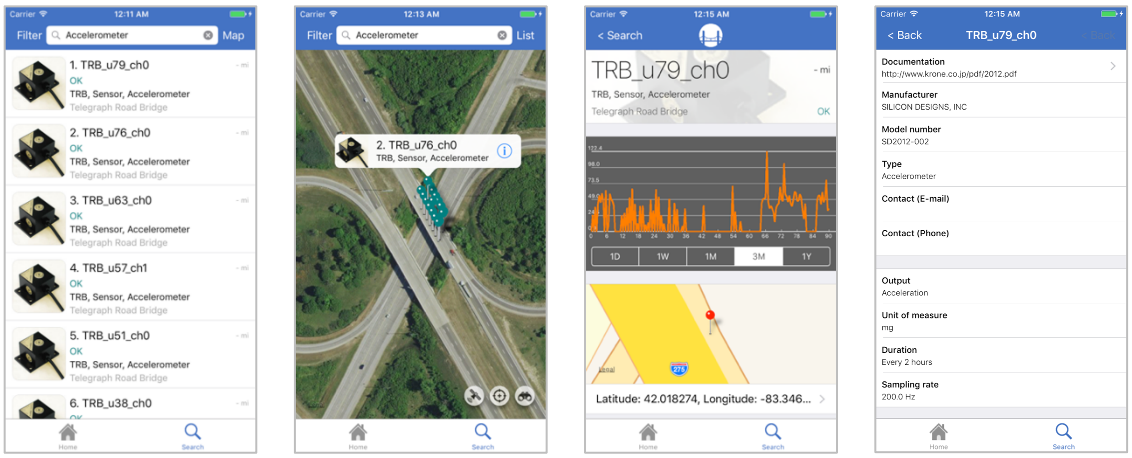
(b) Engineering model visualized by CSiBridge

Figure 4.21 Telegraph Road Bridge model downloaded from the web-based user interface

A mobile user interface is also developed based on the cyberinfrastructure platform. The mobile user interface reads user inputs via the mobile devices, invokes the web services, and displays the retrieved information on the mobile devices. The preliminary mobile user interface is built upon iOS operating system and supports the retrieval of sensor list, sensor information and sensor data. For example, Figure 4.22(a) shows the sensor list view that reads user's search keyword (e.g., "Accelerometer") and retrieves sensor list by invoking the "Sensor list retrieval" service. Figure 4.22(b) shows the sensor map view that displays the retrieved sensor list on a map view. Figure 4.22(c) shows the sensor detail view that displays brief information about a sensor along with its sensor data by invoking the "Sensor data retrieval" service. Figure 4.22(d) shows the sensor information view that displays a sensor's detailed information by invoking the "Sensor information retrieval" service.

4.7 Summary

This chapter describes a cyberinfrastructure platform tailored to civil infrastructure monitoring. The platform offers easy-to-use data management services with interoperable



1. TRB_u79_ch0
OK
TRB, Sensor, Accelerometer
Telegraph Road Bridge

2. TRB_u76_ch0
OK
TRB, Sensor, Accelerometer
Telegraph Road Bridge

3. TRB_u63_ch0
OK
TRB, Sensor, Accelerometer
Telegraph Road Bridge

4. TRB_u57_ch1
OK
TRB, Sensor, Accelerometer
Telegraph Road Bridge

5. TRB_u51_ch0
OK
TRB, Sensor, Accelerometer
Telegraph Road Bridge

6. TRB_u38_ch0

2. TRB_u76_ch0
TRB, Sensor, Accelerometer

TRB_u79_ch0
TRB, Sensor, Accelerometer
Telegraph Road Bridge
OK

Latitude: 42.018274, Longitude: -83.346...

Documentation
http://www.krone.co.jp/pdf/2012.pdf

Manufacturer
SILICON DESIGNS, INC

Model number
SD2012-002

Type
Accelerometer

Contact (E-mail)

Contact (Phone)

Output
Acceleration

Unit of measure
mg

Duration
Every 2 hours

Sampling rate
200.0 Hz

(a) Sensor list view (b) Sensor map view (c) Sensor detail view (d) Sensor information view

Figure 4.22 Prototype mobile interface

interfaces, through which client systems can easily store and retrieve data, in order facilitate the utilization of monitoring data. The platform consists of three layers: communication layer, mapping layer and storage layer. The communication layer leverages web services based on standard communication protocols to offer interoperable interfaces for data management services. The mapping layer enables data mapping between the standardized information modeling schema and database schema. The storage layer adopts the NoSQL-based data management system (which was described in Chapter 3) to enable scalable data management. The data management services offered by the cyberinfrastructure platform can not only be invoked by different client systems, but also be composed with internal and external web services to develop new services.

This chapter also presents the implementation of the cyberinfrastructure platform on cloud computing environment. For the implementation, virtual machines deployed on public cloud are employed. The use of cloud computing services has advantages on the scalability, reliability, availability and easy maintenance of the underlying computing infrastructure. In addition to public cloud-based implementation, this chapter also describes the hybrid cloud-based implementation of the platform for the secure data management. Specifically, the hybrid cloud-based platform is designed to handle voluminous sensor data on the scalable public cloud, while the sensitive bridge information is stored on the private cloud.

The cyberinfrastructure platform is demonstrated with a case scenario of civil infrastructure monitoring. Specifically, the prototype cyberinfrastructure platform has been implemented for the bridge monitoring systems on the Telegraph Road Bridge. Several demonstrative applications have been shown to illustrate the ease of use of the cyberinfrastructure platform for civil infrastructure monitoring. For example, automated data store applications are developed to transmit sensor data and traffic video image to the cyberinfrastructure platform in real time. Furthermore, a demonstrative application that calculates the modal frequency history is developed to show that applications, which involves multiple analysis tasks, multiple queries and iterative jobs, can be developed by leveraging the cyberinfrastructure platform. Finally, preliminary web and mobile user interfaces are developed based on the cyberinfrastructure platform to support ubiquitous access to the bridge monitoring. The demonstration results show that the cyberinfrastructure platform enables easy prototyping and development of applications, and thus, facilitates the utilization of monitoring data.

Chapter 5

Implementation of Data-driven Sensor Data Reconstruction Procedure utilizing the Cyberinfrastructure Platform

5.1 Introduction

The cyberinfrastructure platform described in the previous chapter is designed to facilitate sharing and utilization of data involved in civil infrastructure monitoring. Various applications deployed on different computing systems (e.g., desktop computers, mobile devices and cloud computing environments) can easily retrieve data in machine-readable formats from the cyberinfrastructure platform via standardized communication protocols. Through the cyberinfrastructure platform, applications can also share analysis results among each other. This distributed computing environment can be very useful, particularly, in performing computationally demanding data-driven analyses. For example, training of artificial neural network (ANN) models is often computationally demanding and not efficient to be performed on ordinary desktop computers. The cyberinfrastructure platform can enable an automated data analysis pipeline that trains ANN models remotely on a high-performance computing platform, stores the trained ANN models in a model repository and shares the trained ANN models for use on ordinary desktop computers. The desktop computers can perform less computational prediction procedure using the shared trained model. To showcase the use of the cyberinfrastructure platform, this chapter presents a data-driven sensor data reconstruction method implemented using the cyberinfrastructure platform.

Sensor data reconstruction is an important task for the recovery of missing or faulty sensor data, as well as for the detection of anomalies [135, 136]. Accurate reconstruction of sensor data is critical for ensuring and maintaining a healthy sensor network of a monitored system. Many sensor data reconstruction methods use the spatial correlation among the data in a sensor network in order to estimate the data of the target sensors based on the data collected by other sensors. For example, Kerschen *et al*. [137] present a principal component analysis (PCA)-based method that extracts the PCA modes from the training data to perform detection, identification and reconstruction of a faulty sensor. Kullaa [138] presents a minimum mean square error (MMSE)-based method to reconstruct sensor data using the covariance among the data from different sensors. Many data-driven machine learning techniques have also been applied for the fault detection and diagnosis (FDD) problem [139]. Artificial neural networks (ANNs), for instance, is one widely used technique for sensor validation and reconstruction. For example, feedforward neural networks (FNN) have been employed for sensor data reconstruction by structuring a neural network to have the data of a target sensor as output and the data of other sensors as input [140, 141, 142, 143]. These studies show that FNN can effectively learn the nonlinear spatial relations among the data collected from multiple spatially distributed sensors. Support vector regression (SVR) has also been employed for sensor data reconstruction [144]. Methods that consider only the correlations among the spatially distributed sensors, however, are not effective when spatial correlation among sensors is weak (e.g., sensors separated by a long distance). The spatial correlation-based methods can be improved for obtaining better accuracy by taking into consideration other relevant information, such as the temporal correlations of the sensor data.

Engineering systems, which are typically dynamic systems, often involve temporal correlation (e.g., natural frequencies of the system) in addition to spatial correlation (e.g., mode shapes of the system) within the sensor data [145]. Effective use of spatiotemporal correlation can potentially improve the accuracy of data reconstruction. However, relatively little attention has been paid on spatiotemporal correlation-based sensor data reconstruction, probably due to the increase of complexity. For example, by extending the MMSE-based method to utilize the linear spatiotemporal correlation among the sensors, Kullaa [138] has shown that more accurate data reconstruction can be achieved when compared to using only the spatial correlation. ANN-based methods can be employed to effectively learn the nonlinear spatiotemporal correlation among the sensor data. Moustapha and Selmic [146], for example, employ recurrent neural network (RNN) to learn about the spatial and the temporal correlation among the sensor data. One shortcoming of RNN is that it considers only the information from the past in its input. Future context, if available, can further improve the accuracy of sensor data reconstruction.

This chapter introduces an ANN-based sensor data reconstruction method that considers both spatial and bidirectional temporal correlation among sensor channels from the same system [77]. Specifically, bidirectional recurrent neural network (BRNN) [147], which is an ANN architecture designed to learn about the temporal behavior in both the positive time direction (i.e., past to present) and the negative time direction (i.e., future to present), is employed. While this method can potentially improve reconstruction accuracy compared to other existing methods, the training of a BRNN model is computationally demanding. For efficient BRNN model training and sensor data reconstruction, a data analysis pipeline is designed to enable seamless data flow between high-performance computing platform and ordinary user devices (e.g., desktop computers) by leveraging the previously described cyberinfrastructure platform.

This chapter is organized as follows. Section 5.2 presents a BRNN-based sensor data reconstruction method. Section 5.3 validates the sensor data reconstruction method with datasets obtained from numerical simulations. Section 5.4 presents a data analysis pipeline for deploying a sensor data reconstruction procedure based on the cyberinfrastructure platform. Section 5.5 demonstrates the use of the data analysis pipeline for the BRNN-based sensor data reconstruction method with sensor data collected from the Telegraph Road Bridge (Monroe, MI). Finally, this chapter conclude with a summary in Section 5.6.

5.2 BRNN-based Sensor Data Reconstruction Method

This section describes a data-driven BRNN-based method for sensor data reconstruction. The basic assumption is that the target system contains redundant information inherent in the sensor network and that there exist spatial and temporal correlations among the sensor data. The time series data of the (output) sensor of interest can be estimated using the time series data collected from the other (input) sensors. Figure 5.1 describes the overall framework of the sensor data reconstruction process which consists of two main phases:

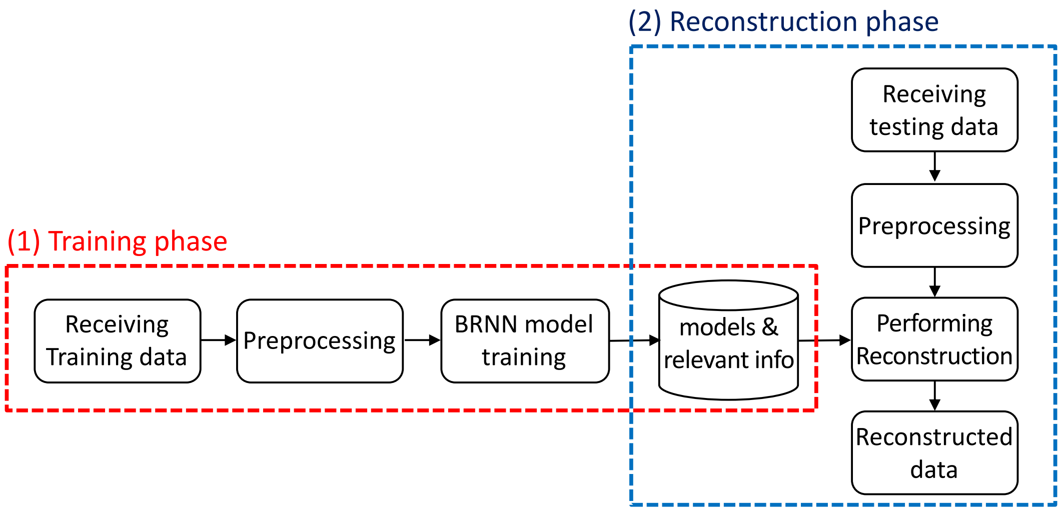


Figure 5.1 Overview of sensor data reconstruction

(1) the training phase, and (2) the reconstruction phase. In the training phase, the time series sensor data collected from the normal state of the target system is employed as the training dataset. The training data is preprocessed (i.e., normalized and scaled) and the normalization and scaling factors are stored for later use in the sensor data reconstruction phase. BRNN models are then constructed and trained using the pre-processed data. In the reconstruction phase, the trained BRNN models utilize the preprocessed testing data and reconstructs the time-series data for the sensor of interest.

5.2.1 Data preprocessing and normalization

Consider the time series input data **X** collected from *N* sensors denoted as:

$$\mathbf{X} = \begin{bmatrix} x_1^1 & \cdots & x_1^n & \cdots & x_1^N \\ \vdots & \ddots & \vdots & & \vdots \\ x_t^1 & \cdots & x_t^n & \cdots & x_t^N \\ \vdots & & \vdots & \ddots & \vdots \\ x_T^1 & \cdots & x_T^n & \cdots & x_T^N \end{bmatrix} \tag{5.1}$$

where x_t^n denote the measurements at time t for an input sensor n. The t^{th} row of **X** is denoted as $\boldsymbol{x}_t = (x_t^1, \dots, x_t^n, \dots, x_t^N)$ which contains the measurements from all N input sensors at time t. The n^{th} column of **X** is denoted as $\boldsymbol{x}^n = (x_1^n, \dots, x_t^n, \dots, x_T^n)^T$ which is the time series measurements of length T for the n^{th} input sensor. Similarly, the output data $\boldsymbol{y}$ from a single output sensor is denoted as:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_t \\ \vdots \\ y_T \end{bmatrix} \tag{5.2}$$

where y_t denote the measurements at time t for the output sensor. Given the paired time series data $\mathbf{D} = (\mathbf{X}; \boldsymbol{y})$, the sensor data reconstruction problem is considered as a supervised regression problem for finding the nonlinear relationships between the input sensors and output sensor utilizing the spatial and temporal correlations among the sensor data.

To ensure the covariance of the input data in approximately of the same order between different instances of the data (which ensures good convergence during the training process), the input time series data is normalized to have zero mean and scaled within the range of $\langle -1,1 \rangle$ as [148]:

$$x_t^n \leftarrow \frac{x_t^n - \mu^n}{s^n} \tag{5.3}$$

where μ^n and s^n are, respectively, the mean and the absolute maximum value of the input data vector $\boldsymbol{x^n}$ for sensor n. Similarly, to make the output values within the typical range of the neural network, the output time series data is normalized and scaled as [149]:

$$y_t \leftarrow \frac{y_t - \bar{\mu}}{\bar{s}} \tag{5.4}$$

where $\bar{\mu}$ and $\bar{s}$ are, respectively, the mean and the absolute maximum value for the output data vector $\boldsymbol{y}$. The mean and the absolute maximum values for both the input and output data vectors are recorded for later use during the sensor data reconstruction phase.

5.2.2 Architecture of the BRNN model

The bidirectional recurrent neural network (BRNN) model with a single hidden layer is depicted as shown in Figure 5.2, Figure 5.3 and Figure 5.4. Figure 5.2 shows the basic computational flow of the BRNN. Note that the computations involve values from both positive and negative time directions across all time steps $t = 1, \ldots, T$. As shown in Figure 5.3, for every time step t, the BRNN model consists of three basic layers:

- An input layer with N input units containing the input data vector $\boldsymbol{x}_t$
- An output layer with a single output unit containing the predicted output $\hat{y}_t$

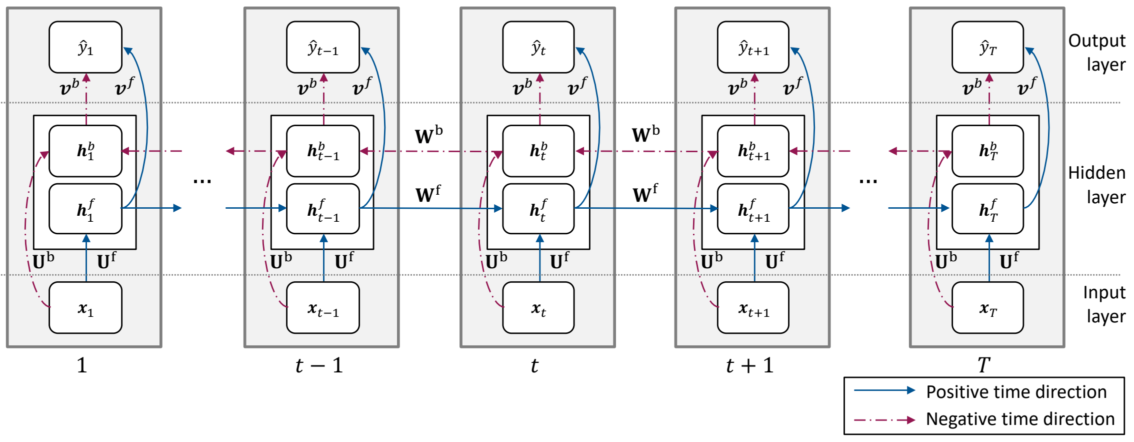


Figure 5.2 Structure of bidirectional recurrent neural network with a single hidden layer and the computation flow of a forward pass

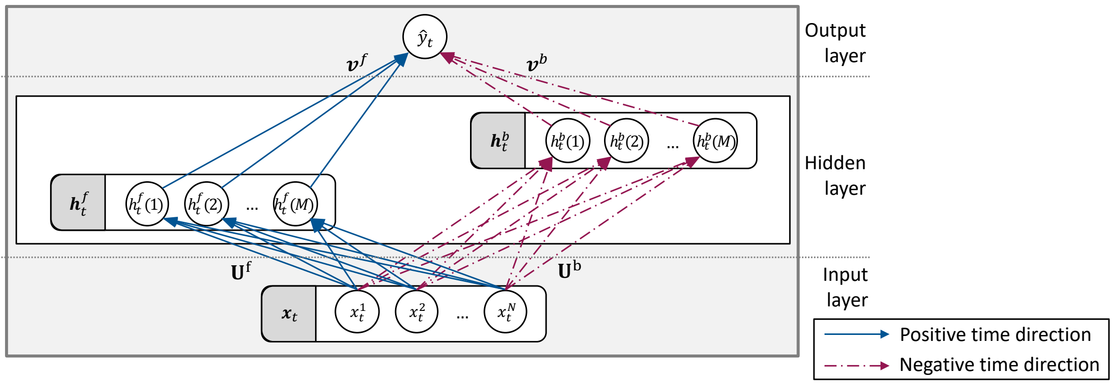


Figure 5.3 Organization of BRNN layers at a time step t

- A hidden layer that consists of M forward hidden units with values denoted by $\{h_t^f(1), \dots, h_t^f(M)\}$ and M backward hidden units with values denoted by $\boldsymbol{h}_t^b = \{h_t^b(1), \dots, h_t^b(M)\}$

One distinguishing feature of BRNN is the cyclic computational flows among the computational units (called neurons) that enable the outputs of neurons at a time step to become a part of the inputs of the neurons at different time steps. To consider both the past data and the future data when reconstructing the present data, the BRNN model has connections in the positive time direction for the forward hidden units, as well as

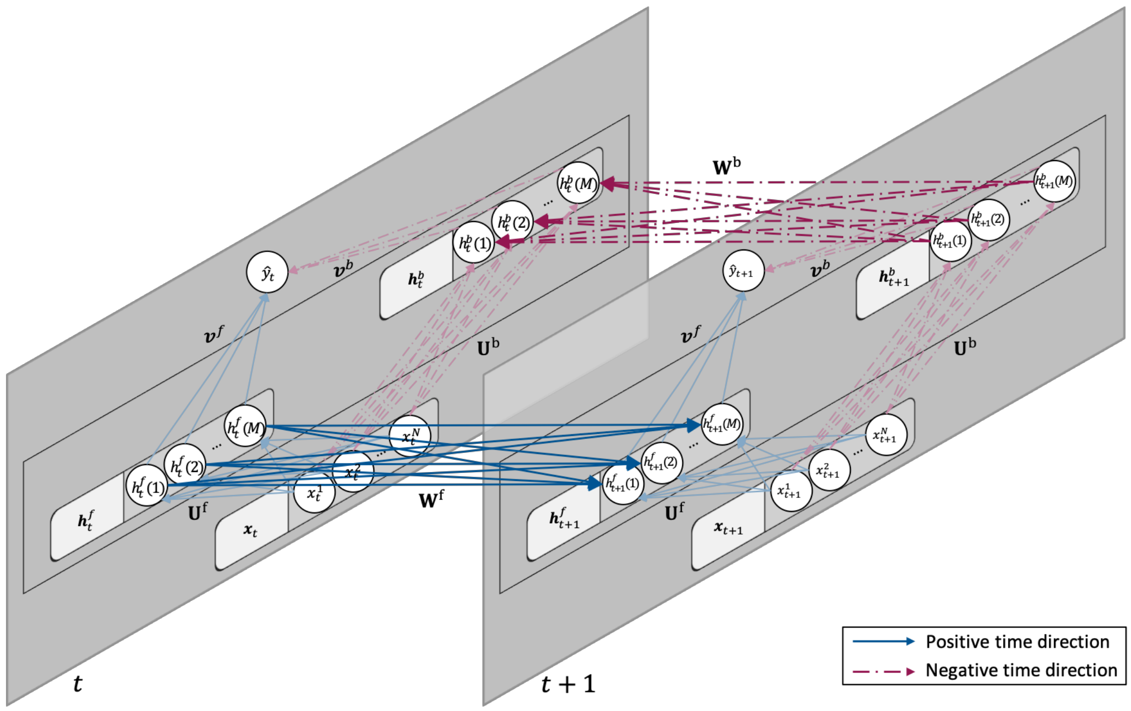


Figure 5.4 Expanded view of interconnection between time steps t and $t+1$ in the BRNN structure

connections in the negative time direction for the backward hidden units, as shown in Figure 5.2. For each time step t, the neurons are interconnected as follows:

- Each of the M forward hidden units containing $\boldsymbol{h}_t^f$ is connected to each of the input units of $\boldsymbol{x}_t$ as well as the M forward hidden units containing $\boldsymbol{h}_{t-1}^f$ of the previous time step $t-1$.
- Each of the M backward hidden units containing $\boldsymbol{h}_t^b$ is connected to each of the input units of $\boldsymbol{x}_t$ as well as the M backward hidden units containing $\boldsymbol{h}_{t+1}^b$ of the next time step $t+1$.
- An output unit for $\hat{y}_t$ is connected to all the M forward and the M backward hidden units containing $\boldsymbol{h}_t^f$ and $\boldsymbol{h}_t^b$, respectively.

In summary, as illustrated in Figure 5.4, the BRNN is designed as a fully connected network within each time step and between the hidden layers of the adjacent time steps.

In Figure 5.2, the variables, $\mathbf{U}^{\mathrm{f}}$, $\mathbf{U}^{\mathrm{b}}, \mathbf{W}^{\mathrm{f}}, \mathbf{W}^{\mathrm{b}}, \boldsymbol{v}^{f}$ and $\boldsymbol{v}^{b}$, denote the weights for the neural network. The values $\boldsymbol{h}_t^f$ of the forward hidden units are computed using the input data $\boldsymbol{x}_t$ and the values $\boldsymbol{h}_{t-1}^f$ from the previous time step via an activation function $f(\cdot)$ as:

$$\boldsymbol{h}_t^f = f\left(\mathbf{U}^{\mathrm{f}}\boldsymbol{x}_t + \mathbf{W}^{\mathrm{f}}\boldsymbol{h}_{t-1}^f + \boldsymbol{b}^f\right) \tag{5.5}$$

where $\boldsymbol{b}^f$ is termed the bias for the forward hidden units. Similarly, the values $\boldsymbol{h}_t^b$ of the backward hidden units are computed using the input $\boldsymbol{x}_t$ and the values $\boldsymbol{h}_{t+1}^b$ from the next time step using the same activation function $f(\cdot)$ as:

$$\boldsymbol{h}_t^b = f\left(\mathbf{U}^{\mathrm{b}}\boldsymbol{x}_t + \mathbf{W}^{\mathrm{b}}\boldsymbol{h}_{t+1}^b + \boldsymbol{b}^b\right) \tag{5.6}$$

where $\boldsymbol{b}^b$ is the bias for the backward hidden units. Because of its ability to model nonlinear relationships among the data, the hyperbolic tangent function is employed as the activation function:

$$f(z) = \tanh(z) = \frac{(e^z - e^{-z})}{(e^z + e^{-z})} \tag{5.7}$$

Finally, the predicted output is computed using a linear function as:

$$\hat{y}_t = \boldsymbol{v}^f \cdot \boldsymbol{h}_t^f + \boldsymbol{v}^b \cdot \boldsymbol{h}_t^b + c \tag{5.8}$$

where c is the bias for the output unit. The weights and the bias represent the parameters of the BRNN model to be “trained” using the input and output data sets.

5.2.3 Training of BRNN model

The BRNN model is trained by adjusting iteratively its parameters to minimize the loss function L that measures the difference between the output data $\boldsymbol{y} = (y_1, \dots, y_t, \dots, y_T)^T$ and the reconstructed output data $\hat{\boldsymbol{y}} = (\hat{y}_1, \dots, \hat{y}_t, \dots, \hat{y}_T)^T$. For initialization, the biases $\boldsymbol{b}^f, \boldsymbol{b}^b$ and c are set to zero and the elements for the weights $\mathbf{U}^{\mathrm{f}}, \mathbf{U}^{\mathrm{b}}, \mathbf{W}^{\mathrm{f}}, \mathbf{W}^{\mathrm{b}}, \boldsymbol{v}^f$ and $\boldsymbol{v}^b$ of the BRNN are assigned randomly using a uniform distribution between $-\frac{1}{\sqrt{l}}$ and $\frac{1}{\sqrt{l}}$, where l is the size of a prior layer (where, in this case, $l = N$ for $\mathbf{U}^{\mathrm{f}}$ and $\mathbf{U}^{\mathrm{b}}$, and $l = M$ for $\mathbf{W}^{\mathrm{f}}, \mathbf{W}^{\mathrm{b}}, \boldsymbol{v}^{f}$ and $\boldsymbol{v}^{b}$) [150].

Once the parameters are initialized, the backpropagation through time (BPTT) algorithm is employed for training the BRNN model [147, 151]. For each iteration, BPTT consists of three steps: forward pass, backward pass and parameter updating. In each iteration, the values $\boldsymbol{h}_0^f$ of the forward hidden units and the values $\boldsymbol{h}_{T+1}^b$ for the backward hidden units are initialized to be zero. In the forward pass, the predicted output $\hat{y}_t$, $t = 1, \dots, T$, is calculated using the input data $\mathbf{X}$ through Eqs. (5.5), (5.6) and (5.8). In the backward pass, the gradients of the loss function L (i.e., difference between the measured output $\boldsymbol{y}$ and the predicted output $\hat{\boldsymbol{y}}$) with respect to the parameters are computed. In this study, the loss function, L, is computed using the mean-squared-error (MSE) function as:

$$L = \frac{1}{T}\sum_{t=1}^{T}(y_t - \hat{y}_t)^2 \tag{5.9}$$

For each time step t, the gradient of the loss function L with respect to the predicted output $\hat{y}_t$ is computed and is then propagated forward and backward through time via the network to calculate the gradients of the loss function L with respect to the parameters. The gradients for each parameter are summed over all T time steps for parameter updating. The gradients of L computed with respect to the parameters are used to adjust the corresponding parameters using an optimization procedure, such as Adaptive Moment Estimation (Adam) [152]. Additionally, learning rate which limits the amount of adjustment per each iteration is defined. The learning rate can affect the number of iterations in the training process and the accuracy of the BRNN model. A higher learning rate can speed up the training, while a lower learning rate likely improves the accuracy of the BRNN model. The three steps of forward pass, backward pass and parameter updating iterate until either L converges or the number of epochs (i.e., the number of times that the entire datasets are being processed) reaches a prescribed limit. It should be noted that in the BRNN model, the weights

$\mathbf{U}^{\mathrm{f}}, \mathbf{U}^{\mathrm{b}}, \boldsymbol{v}^{f}$ and $\boldsymbol{v}^{b}$ and the biases $\boldsymbol{b}^{f}, \boldsymbol{b}^{b}$ and c are the same for every time step, and the weights $\mathbf{W}^{\mathrm{f}}$ and $\mathbf{W}^{\mathrm{b}}$ are the same for each forward and backward time step. Details on the forward pass, backward pass and parameter update for BRNN can be found in [151, 153].

One issue of training BRNN is their significant computational and memory requirements due to the chains of connections linking the neurons. For unidirectional RNN, truncated BPTT (TBPTT) methods are often used to reduce the computational cost and memory usage by limiting the number of time steps for which the loss is propagated [154, 155]. However, implementing the TBPTT strategy for BRNN is not practical because the hidden units along either the positive or negative time directions need to be processed over all time steps in order to compute the estimated outputs. Instead, this work adopts a batch learning approach [155]. As summarized in Algorithm 5.1, the approach defines a batch size B which is the maximum number of consecutive data points to be used in an iteration (corresponding to one cycle of forward pass, backward pass and parameter updating for that batch). That is, the training dataset $\mathbf{D} = (\mathbf{X}; \boldsymbol{y})$ with T consecutive data points is divided into $\lfloor T/B \rfloor$ batches so that the batches $\mathbf{D}_1, \dots, \mathbf{D}_{\lfloor T/B \rfloor - 1}$ each contains B consecutive data points whereas $\mathbf{D}_{\lfloor T/B \rfloor}$ contains the remaining $T - (\lfloor T/B \rfloor - 1) \times B$ consecutive data points (see step 3 of Algorithm 5.1). As shown from steps 6 to 11 in Algorithm 5.1, each batch of the dataset is trained independently. Step 4 of the Algorithm 5.1 shows the stopping criteria for the training when either the maximum number of epochs R is reached or the loss change, ΔL, is within a defined threshold, $\Delta L_{\mathrm{thres}}$. With this approach, BRNN can be trained with a large amount of training data without causing excessive memory usage by limiting the number of times the loss is propagated.

5.2.4 Sensor data reconstruction

Consider a test dataset $\mathbf{D}' = (\mathbf{X}'; \boldsymbol{y}')$ collected from the same N input sensors and an output sensor with T' consecutive data points per sensor. The length of the data points (i.e., the sensor data) should span over a certain time period sufficient to establish the temporal correlation among the sensor data. The sufficient period may preclude true real-time data

Algorithm 5.1. Training of BRNN model

```
Input:
        $\mathbf{D} = (\mathbf{X};\ \boldsymbol{y})$: preprocessed training dataset
        $T$: number of training data points
        $B$: batch size
        $R$: number of maximum epochs
        $\Delta L_{\text{thres}}$: minimum value of loss change
        $\eta$: learning rate
1:  Initialize the BRNN model's weights and biases
2:  Initialize: number of epochs $r \leftarrow 0$, loss in the previous step $L_{\text{prev}} \leftarrow \infty$, and
    change of loss $\Delta L \leftarrow \infty$
3:  Divide $\mathbf{D}$ into $\lfloor T/B \rfloor$ batches: $\mathbf{D} = \{\mathbf{D}, \dots, \mathbf{D}_{\lfloor T/B \rfloor}\}$
4:  while $(r < R)$ or $(\Delta L > \Delta L_{\text{thres}})$ do
5:      Set $L_{\text{curr}} \leftarrow 0$
6:      for each $\mathbf{D}_i = (\mathbf{X}_i; \boldsymbol{y}_i), i = 1, \dots, \lfloor T/B \rfloor$ do
7:          Forward pass: compute $\widehat{\boldsymbol{y}}_i$ using input $\mathbf{X}_i$ according to Eqs. (5.5), (5.6) and
              (5.8)
8:          Backward pass: (a) compute the loss function $L$ between output $\boldsymbol{y}_i$ and $\widehat{\boldsymbol{y}}_i$
              using the MSE function in Eq. (5.9), and (b) compute the gradients of
              the loss with respect to the BRNN parameters
9:          Parameter updating: update the weights and biases based on the gradients
              of the loss with respect to the parameters and the learning rate $\eta$ using
              Adam [146]
10:         $L_{\text{curr}} \leftarrow L_{\text{curr}} + L$
11:     end for
12:     Update $r \leftarrow (r+1)$; $L_{\text{curr}} \leftarrow \frac{L_{\text{curr}}}{\lfloor T/B \rfloor}$; $\Delta L \leftarrow |L_{\text{prev}} - L_{\text{curr}}|$; $L_{\text{prev}} \leftarrow L_{\text{curr}}$
13: end while
```

reconstruction, but the actual time duration is relatively short. As shown in the examples described later in Section 5.3, collecting 1,000 consecutive data points takes only five seconds at a sampling rate of 200 Hz.

The sensor data reconstruction is performed as follows. Using the normalization and scaling factors recorded in the training phase, the data $\mathbf{D}'$ is first normalized and scaled using Eqs. (5.3) and (5.4). The trained BRNN model then takes the input data $\mathbf{X}'$ to compute the prediction $\widehat{\boldsymbol{y}}'$ for the output sensor's data according to Eqs. (5.5), (5.6) and (5.8). Here, the initial forward hidden units $\boldsymbol{h}_0^f$ and the backward hidden units $\boldsymbol{h}_{T'+1}^b$ are assumed to be vectors of zeros. It should be noted that the reconstructed data for the first and the last few (e.g., 10) time steps can be erroneous due to the effect of the initial hidden units. The accuracy of the reconstructed data can be evaluated by measuring the discrepancy between the measured output $\boldsymbol{y}'$ and the predicted output $\hat{\boldsymbol{y}}'$, for example, using the root-mean-square-error (RMSE) as:

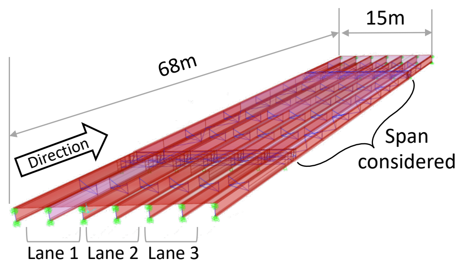
$$\epsilon = \sqrt{\frac{\sum_{t=1}^{T'} (\hat{y}_t' - y_t')^2}{T'}} \tag{5.10}$$

The trained BRNN model can have at least two different usages. First, if the output sensor is known to be faulty, the faulty measurement data can be replaced by the reconstructed data (as will be discussed in Section 5.5.4). The reconstructed data can be further utilized to estimate the health condition (e.g., operational frequency and mode shapes) of the monitored system even when some of the sensors are faulty. Second, if the output data possesses the possibility of anomalies, the testing error ϵ computed by Eq. (5.10) can be used as to validate the output sensor (as will be illustrated in Section 5.3.1.3). Since the BRNN model's testing errors computed using the intact datasets tend to lie within a narrow range, some statistical information (such as, maximum value or 95% confidence interval) about the testing errors can be used as the thresholds for distinguishing the potential anomaly. The utilization of the BRNN-based sensor data reconstruction method will be further discussed in Section 5.3 and Section 5.5.

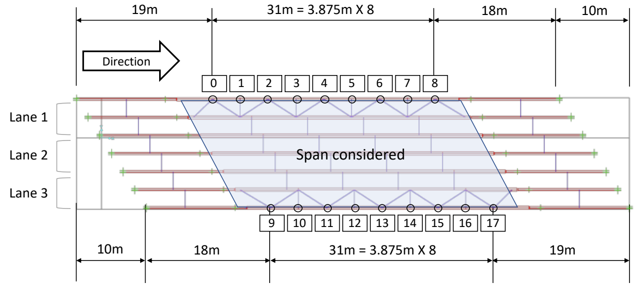
5.3 Demonstration of Sensor Data Reconstruction Method with Numerical Simulation

In this section, the BRNN-based sensor data reconstruction method is demonstrated with vibration data obtained from numerical simulations. The FE model, as shown in Figure 5.5(a), is constructed for the Telegraph Road Bridge [156, 157]. Vertical vibration responses due to randomly traveling vehicles are simulated and recorded at 18 (sensor) locations on the two exterior girders, as shown in Figure 5.5(b).

To emulate the load conditions on a bridge, the moving vehicles are randomly defined using the variables summarized in Table 5.1. The vehicle types (labelled as auto, H-20, HS-20 and HS-25) and their loads are defined based on the AASHTO (American Association of State Highway and Transportation Officials) standard [158], as shown in Figure 5.6. Dynamic time history analyses using CSiBridge are performed with randomized traffic on the bridge to generate the vertical vibration response (sensor) data. It should be noted that the vehicle-bridge interaction is ignored in the analyses. Each analysis assumes a 10-second duration with incremental time step size of 0.005 (i.e., a



(a) 3-D FE model



(b) Sensor layout

Figure 5.5 (a) Finite element model and (b) sensor layout of the Telegraph Road Bridge

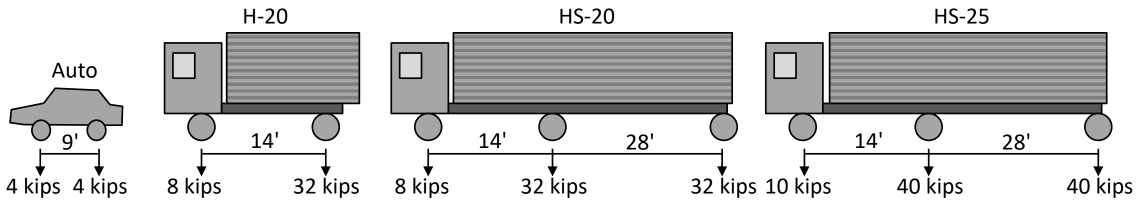


Figure 5.6 Vehicle load definition

Table 5.1 Random variables composing randomized traffic

Factor	Values
Number of vehicles	1, 2, …, 9
Vehicle type	Auto, H-20, HS-20, HS-25
Vehicle speed	70 – 120 km/h (45 – 75 mph)
Vehicle lane	Three lanes (lane 1, lane 2 and lane 3)
Vehicle interval in a lane	1.5, 3.0, 4.5, 6.0 seconds

sampling rate of 200 Hz). To generate sufficient amount of data for the data-driven method, a total of 300 analyses are conducted for each FE bridge model with randomized moving vehicle loads. Through 300 analyses, 600,000 data points per sensor location are collected, normalized, and scaled as shown in Figure 5.7.

BRNN models are constructed using PyTorch [159]. The hyperparameters for the BRNN model training are selected heuristically to ensure sufficient data reconstruction accuracy, as well as reasonable computing time. The selected hyperparameters are as follows:

- Number of forward hidden units: 50
- Number of backward hidden units: 50
- Maximum number of epochs: 200
- Learning rate 10^{-3}

The appropriate training dataset size and the batch size will be examined and discussed first based on the reconstruction accuracy of the BRNN.

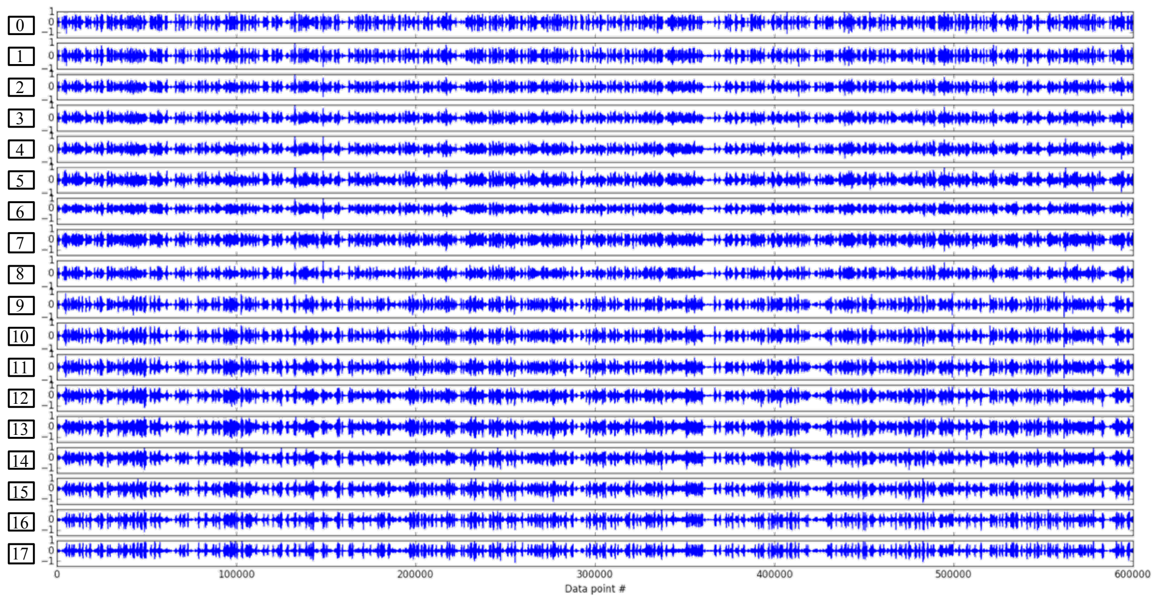
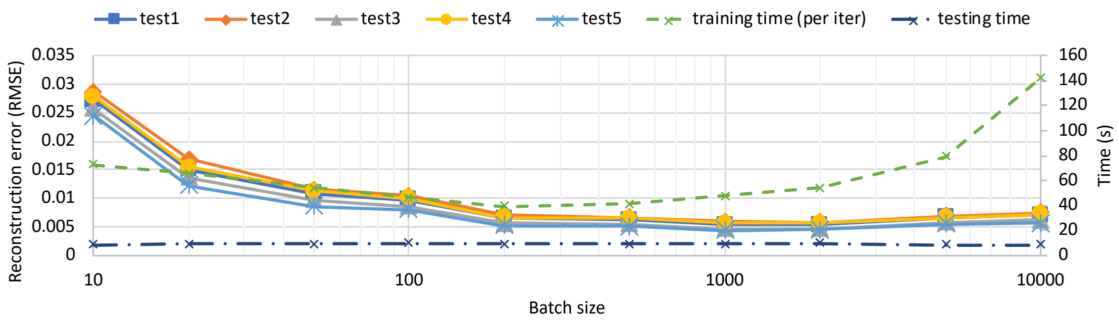


Figure 5.7 Simulated vertical acceleration measurements of the Telegraph Road Bridge (Monroe, MI)

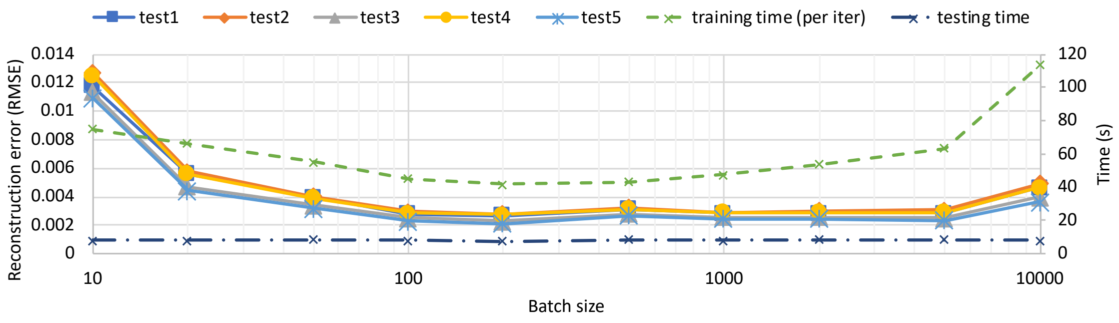
5.3.1 Effects of training parameters

This section describes the training of BRNN models for sensor data reconstruction and discusses the effect of batch size, training dataset size and testing dataset size on the data reconstruction accuracy and computing speed. Two BRNN models for two different output sensors are considered for testing purpose. The first model has Sensor 4 (located at the middle of the topmost girder of Figure 5.5(b)) as the output sensor, while all other sensors are the input sensors. The second model has Sensor 9 (located near the support of the bottommost girder) as the output sensor, while all the other sensors are the input sensors.

For the implementation of sensor data reconstruction procedure as described in Algorithm 5.1, it is important to choose the appropriate batch size. On the one hand, if the batch size is too small, sensor data reconstruction would be inaccurate because of the initial hidden units of zero values, as well as insufficient temporal correlation. On the other hand, if the batch size is too large, model training would require very large memory usage (which does not necessarily lead to better accuracy). To investigate the effect of batch size, the two BRNN models are trained with different batch sizes using the training dataset of fixed size. Here, 100,000 data points (from the data points 1 to 100,000) are selected as a training dataset. Five testing datasets, each of which has 40,000 consecutive data points per sensor, are selected from the data points 200,001 to 400,000. Figure 5.8 shows the RMSE error between the actual and reconstructed values, the training time per epoch and the reconstruction time with respect to the different batch sizes for Sensors 4 and 9. For both sensors, the errors are relatively low when the batch size is higher than 100. In addition, the training times are relatively low when the batch size ranges from 100 to 2,000. It can also be seen that the time for testing is almost the same for all batch sizes. Based on the results, the batch size ranging from 200 to 2,000 data points is able to achieve a good balance of training time and reconstruction error from the testing datasets.



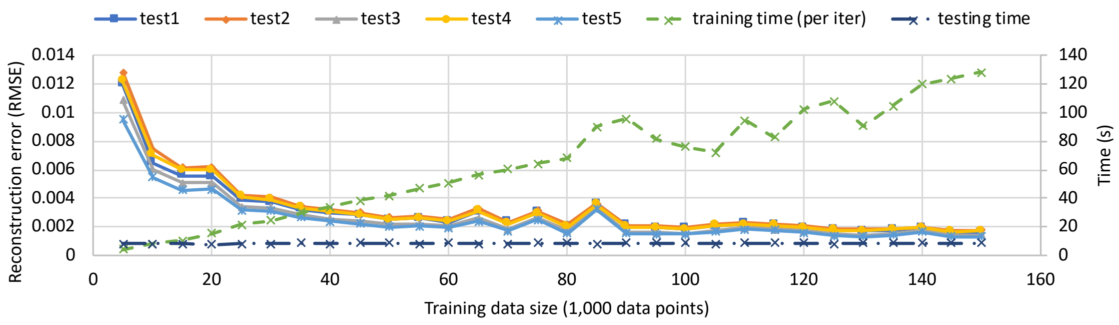
(a) Sensor 4



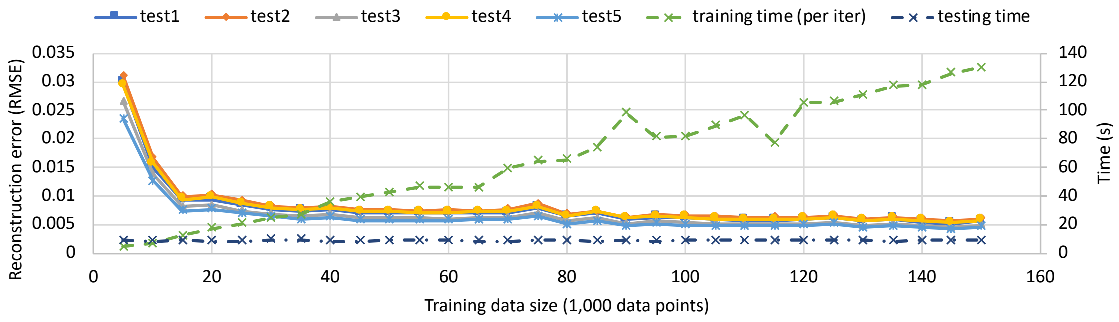
(b) Sensor 9

Figure 5.8 Reconstruction error, training time and reconstruction time with respect to different batch sizes

The next example examines the effect of the training dataset size on data reconstruction. The two BRNN models are trained using training datasets of different size ranging from 5,000 to 150,000, while the batch size is kept at 200 data points. For every training dataset, the normalization and scaling factors are obtained from the largest dataset with 150,000 data points so that the testing errors can be compared. Similar to the previous example, five testing cases each of which has 40,000 consecutive data points per sensors (selected from the data points 200,001 to 400,000) are used. Figure 5.9 shows the testing error, the training time per epoch and the testing time with respect to the different training data size for Sensors 4 and 9. As can be seen in the figure, the training time increases nearly linear as the training data size increases, while the testing times are the same regardless of the training data size. For both sensors, the error decreases rapidly until the training data size reaches 80,000; beyond that, little improvements (sometimes worsen) are observed. Based



(a) Sensor 4

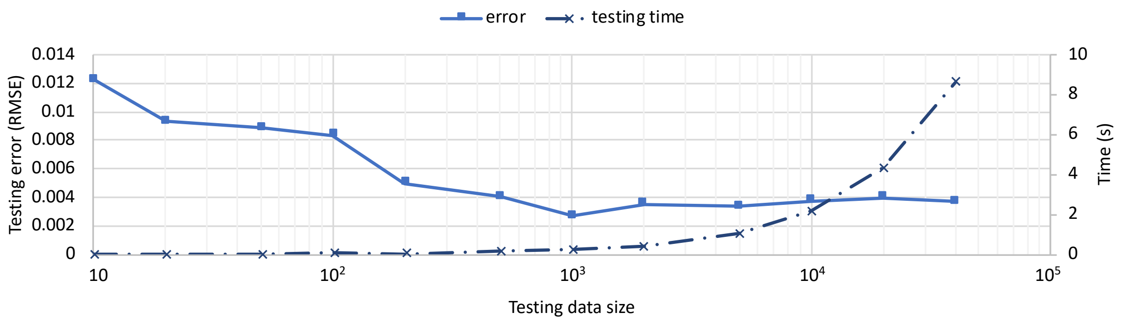


(b) Sensor 9

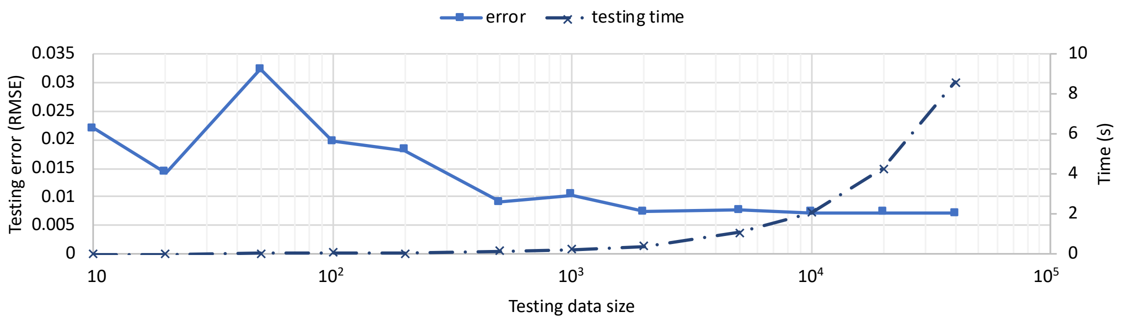
Figure 5.9 Reconstruction error, training time and reconstruction time with respect to different training data sizes

on the results, the training dataset size equal to or higher than 80,000 data points would achieve a good accuracy for sensor data reconstruction.

The last example examines the effect of the size of the testing dataset on data reconstruction. Since the initial hidden units (i.e., $\boldsymbol{h}_0^f$ and $\boldsymbol{h}_{T'+1}^f$) of BRNN models are set to zero vectors, the testing error could be high when the testing dataset is very short. Sensor data reconstructions are performed using the two BRNN models for different testing data sizes ranging from 10 to 40,000. To see the effects due to the zero vectors of the initial hidden units, the testing dataset is selected from the data with relatively high amplitudes (from the data points 200,401 to 240,400). In addition, the two BRNN models are trained with 80,000 data points (from the data points 1 to 80,000) per sensor and a batch size of 200 data points. Figure 5.10 shows the error and the computational time for the testing dataset with respect to the data size. The results show that the errors are relatively consistent



(a) Sensor 4



(b) Sensor 9

Figure 5.10 Reconstruction error and testing time with respect to different testing data sizes

when the testing data size reaches 1,000 or higher for both sensors. It can also be seen that the data reconstruction time increases linearly as the size of testing dataset increases. Based on the results, the testing dataset size equal to or higher than 1,000 data points would be sufficient to mitigate the effect of initial hidden units with zero vectors.

5.3.2 Comparison of BRNN-based sensor data reconstruction with other methods

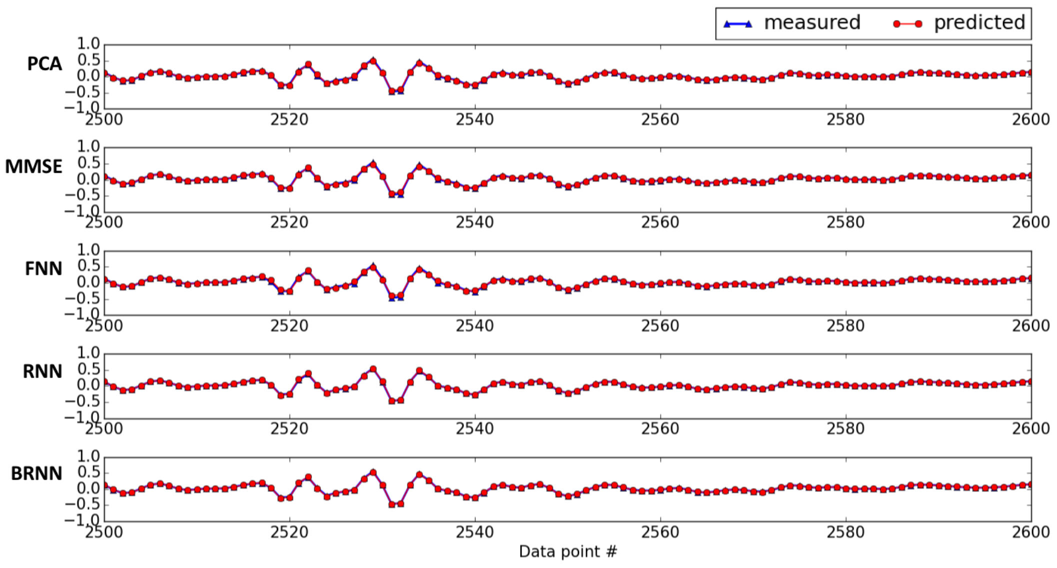
As discussed in the introduction section of this chapter, other data-driven sensor data reconstruction methods, such as principal component analysis (PCA) [137], minimum mean square error (MMSE) estimation [138], feedforward neural network (FNN) [142] and recurrent neural network (RNN) [146], have been proposed. In this section, the BRNN-based sensor data reconstruction method is evaluated by comparing it with the existing methods. Again, two BRNN models for reconstructing the data of Sensor 4 and Sensor 9 are created for testing purpose. For comparison, PCA, MMSE, FNN and RNN models corresponding to the two BRNN models are also created. As listed in Table 5.2, the FNN- and RNN-based models are created with the same hyperparameters of the BRNN model.

All the models are trained with 80,000 data points (from the data points 1 to 80,000) per sensor. For evaluation, the testing dataset has 40,000 data points per sensor and is selected from the data points 80,001 to 120,000. Figure 5.11 shows the samples of sensor data reconstruction results using the different methods. It should be noted that Figure 5.11 shows only 100 data points out of 40,000 data points of entire reconstructed data so that the plots can clearly shows the comparison between the original data and the reconstructed data.

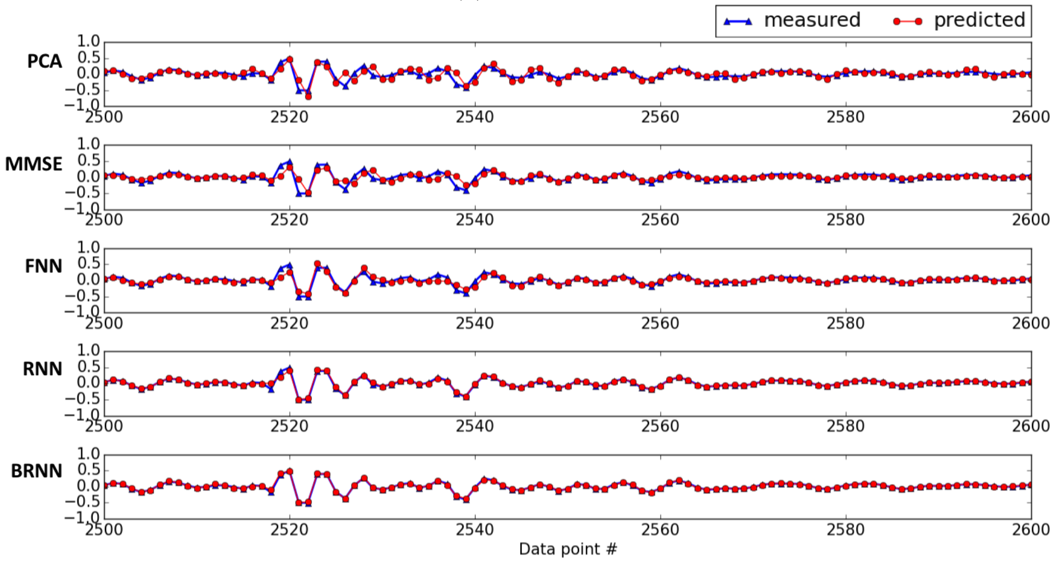
As can be seen in Figure 5.11(a), all methods work well for Sensor 4 which is located at the center of the bridge. For Sensor 9, which is located near the span support, the RNN and BRNN methods work better than the other methods as shown in Figure 5.11(b). Table 5.3 provides detailed information on the reconstruction results, including the computation time and the testing errors. A notable aspect shown in the results is that the testing error for the sensor at the center of the bridge (i.e., Sensor 4) is generally lower than the sensor located near the support of the bridge (i.e., Sensor 9). This is because the sensor located near the

Table 5.2 Data-driven sensor data reconstruction models for comparison

	FNN	**RNN**
Number of hidden layers	1	1
Number of hidden units	100	100
Transfer function (hidden layer)	Hyperbolic tangent function	Hyperbolic tangent function
Transfer function for output layer	Linear function	Linear function
Optimization algorithm	Adam	Adam
Loss function	MSE	MSE
Learning rate	10^{-3}	10^{-3}
Maximum number of epochs	200	200



(a) Sensor 4



(b) Sensor 9

Figure 5.11 Simulation: sensor data reconstruction with the different methods

Table 5.3 Simulation: computing time and reconstruction error

	PCA	MMSE	FNN	RNN	BRNN
Training time per epoch (sec/epoch)	-	-	≈ 0.51	≈ 34.19	≈ 86.21
Total training time (sec)	≈ 0.053	≈ 0.026	≈ 102	≈ 6,838	≈ 17,242
Testing time (sec)	≈ 2.155	≈ 0.006	≈ 0.049	≈ 7.445	≈ 8.3971
Testing RMSE for Sensor 4	0.0154	0.0151	0.0125	0.0034	0.0035
Testing RMSE for Sensor 9	0.0658	0.0432	0.0308	0.0105	0.0070

support has only one adjacent sensor, which means that there exists less spatial information for estimating the sensor data. Nevertheless, BRNN models yield much smaller testing error for Sensor 9 than all other methods, because BRNN takes both spatial and bidirectional temporal correlation into account. One tradeoff of the BRNN models' accuracy is the amount of training time required. But once the BRNN model is trained, the reconstruction process can be executed very efficiently even though the computing time remains higher than the other methods.

5.3.3 Sensor validation

This section demonstrates the utilization of the sensor data reconstruction method for sensor validation, including fault detection and isolation. Among the widely used fault detection and isolation (FDI) methods is the analytical redundancy approach that determines anomalies by comparing system measurements with analytically estimated information [139, 160, 161]. The basic idea of sensor validation is that the existence of a faulty sensor will lead to some "measurable" difference between the measured data and the reconstructed data.

For the demonstration, three types of datasets are defined: a training dataset, a set of baseline datasets and a faulty dataset. The training dataset is a good quality (fault-free) dataset for training the BRNN models. 80,000 data points (from the data points 1 to 80,000) per sensor are used as the training dataset. The baseline datasets are also fault-free datasets for obtaining statistical information of the testing error when no anomaly exists. Ten baseline datasets, each with 40,000 data points selected from the data points 80,001 to 480,000, are defined. The faulty dataset is the dataset that contains some anomalies (i.e., faults). For testing purposes, the faulty dataset contains 40,000 data points (selected from the data points from 480,001 to 520,000) per sensor. In the faulty dataset, the data of Sensors 4 and 6 are manipulated to emulate faulty sensor data. Specifically, for the faulty dataset, random noises s are introduced to the data, denoted by x_t^n, of Sensor 4 and 6 as:

$$x_t^n \leftarrow x_t^n \times s, \qquad s \in N(1, \sigma^2) \tag{5.11}$$

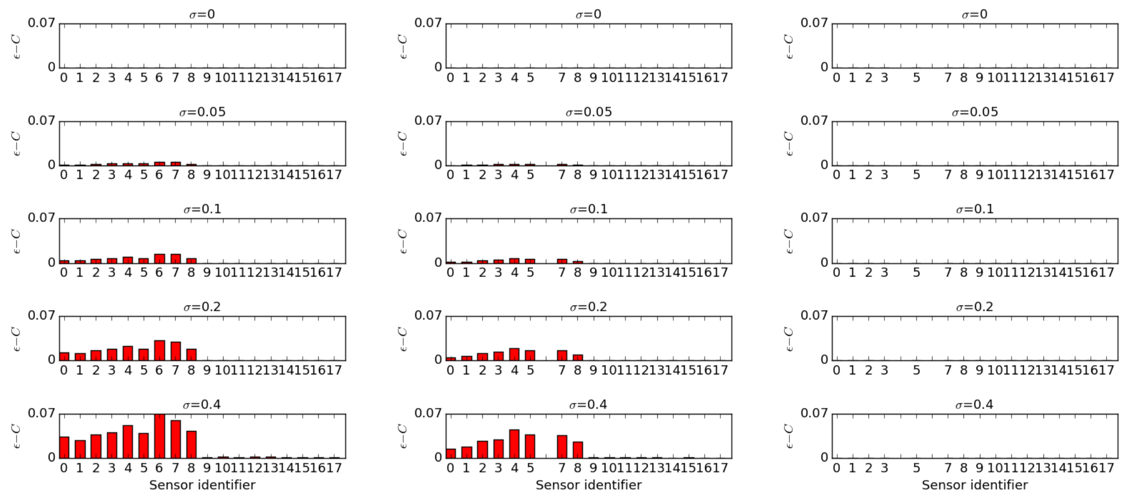
where σ is the level of noise which is set by varying $\sigma = 0, 0.05, 0.1$.

The goal of fault detection is to determine the existence of a faulty sensor in a system. The experimental test for fault detection is conducted as follows:

(1) For each sensor, a BRNN model that has that sensor as the output and all other sensors as input is created.
(2) All created BRNN models are trained using the training dataset.
(3) Sensor data reconstruction is performed for the 10 baseline datasets using the trained BRNN models.
(4) For each sensor, the 95% confidence interval of the testing error is computed based on the testing errors from the 10 baseline datasets. The 95% confidence interval serves as the threshold for sensor validation.
(5) Sensor data reconstruction is performed for the faulty dataset using the trained BRNN models.
(6) For each sensor, the testing error from the faulty dataset is computed and compared with the threshold for sensor validation.

Figure 5.12(a) shows the difference between the testing error ϵ from the faulty dataset that exceeds the threshold of the 95% confidence interval. As shown in Figure 5.12(a), the testing errors exceed the thresholds when the noise level $\sigma \geq 0.05$. As the noise level increases, the discrepancy between the testing error from the faulty data and the thresholds of the baseline model increases. It can also be seen that the discrepancies appear on all sensors (1-8) along the same girder with Sensors 4 and 6. This result is probably due to higher correlations among the sensor channels on the same girder. While the anomaly is detected by comparing the testing errors and thresholds, further analysis is needed to identify the location of the faulty sensor.

While anomaly detection focuses on determining if any faulty sensors exist in the sensor network, the goal of anomaly identification is to localize the faulty sensor in a system. A



(a) Anomaly detection with all sensors (b) Anomaly detection after excluding Sensor 6 (c) Anomaly detection after excluding Sensor 4 and 6

Figure 5.12 Simulation: anomaly detection and identification for the anomaly due to noise at Sensor 4 and 6

simple approach for anomaly identification would be to infer the sensors with testing errors exceeding their thresholds as faulty sensors. However, this approach is prone to false positives because a faulty sensor can affect the testing error not only for the faulty sensor itself, but also on the other (input) sensors. Another approach is to consider the sensor with the largest testing error in comparison to the threshold used for the anomaly detection as the faulty sensor. However, this approach is prone to false negatives, in particular, when there exist multiple faulty sensors. To avoid the likelihood for false positive and false negative errors, this work employs an elimination approach. The basic idea is that the faulty sensors do not affect the measurement of other sensors and, thus, the anomaly disappears if all faulty sensors are removed from the sensor network.

To identify the faulty sensor, the dataset for Sensor 6, which has the highest discrepancy between the testing errors from the faulty dataset and the threshold, is assumed to be faulty and removed. For the remaining 17 sensors, the procedures of model construction, model training, computing the thresholds and calculating the testing errors with the faulty dataset are repeated. Figure 5.12(b) shows the results after removing the data of Sensor 6. The results, however, still show detected anomalies, where Sensor 4 has the highest discrepancy between the testing errors and its threshold. Therefore, the anomaly identification procedure is continued by removing the datasets for Sensors 4 and 6. For the remaining 16 sensors, the procedures for model training, computing the threshold and calculating the testing error is repeated. Figure 5.12(c) shows the results without the datasets from Sensors 4 and 6. As no anomaly is detected after removing the datasets for Sensors 4 and 6, this implies that Sensors 4 and 6 are faulty. This example shows that the BRNN-based method can potentially help detect and identify faulty sensors.

5.4 Data Analysis Pipeline based on Cyberinfrastructure Platform

This section describes the pipeline for implementing the BRNN-based sensor data reconstruction procedure with the cyberinfrastructure platform developed in the previous chapter. Typically, it is time-consuming to train an artificial neural network. High-performance computers can be used to speed up the training process. However, in general, high performance computers are relatively expensive and may not be cost-effective for less computationally demanding task. In this work, we train the BRNN model for sensor data reconstruction on a high-performance computer and store and share the trained sensor data reconstruction model on the cyberinfrastructure platform. During the sensor data reconstruction phase, the BRNN model is imported to a local computer for execution. This section describes in detail the overall data and computational flow for the data-driven machine learning process.

Figure 5.13 describes the overall organization for utilization of the cyberinfrastructure platform for long-term data analysis process. There are three main components involved in the process: a high-performance computer (on the premise or on the cloud), the cyberinfrastructure platform and a local computer. The high-performance computer, typically equipped with one or more GPUs, is employed for the computationally

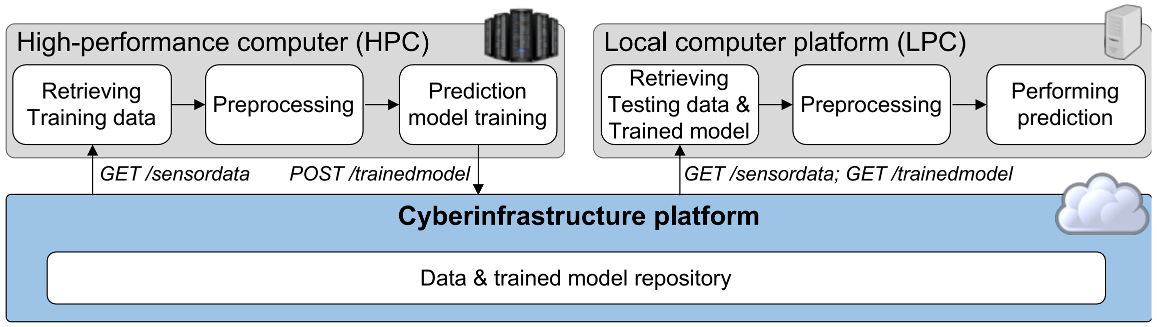

High-performance computer (HPC)
Retrieving Training data
Preprocessing
Prediction model training
Local computer platform (LPC)
Retrieving Testing data & Trained model
Preprocessing
Performing prediction
GET /sensordata
POST /trainedmodel
GET /sensordata; GET /trainedmodel
Cyberinfrastructure platform
Data & trained model repository

Figure 5.13 Overview of data analysis process for sensor data reconstruction

demanding task for training the sensor data reconstruction model. The cyberinfrastructure platform serves as a data hub which manages the sensor data, as well as the trained models. The local computer platform (such as a desktop or laptop computer, or a tablet) is employed to perform sensor data reconstruction using the trained models. This section describes the computational processes on each of components and their interactions.

5.4.1 Training of BRNN model on high-performance computer

For the computationally intensive training of the BRNN-based sensor data reconstruction model, high-performance computer, which can be an on-premise machine or a virtual machine deployed on cloud platform, is desirable. For training the BRNN model, the following assumptions are made:

- The time period during which the sensor data with good quality is collected is known.
- The list of input and output sensors pairing for the BRNN predictive model are selected.
- The hyperparameters (e.g., number of hidden units) of the BRNN-based sensor data reconstruction model, as described in Section 5.2, are pre-defined.

Training a sensor data reconstruction model consists of three basic steps: retrieval of the training data set, training of the model, and uploading and storing the trained model.

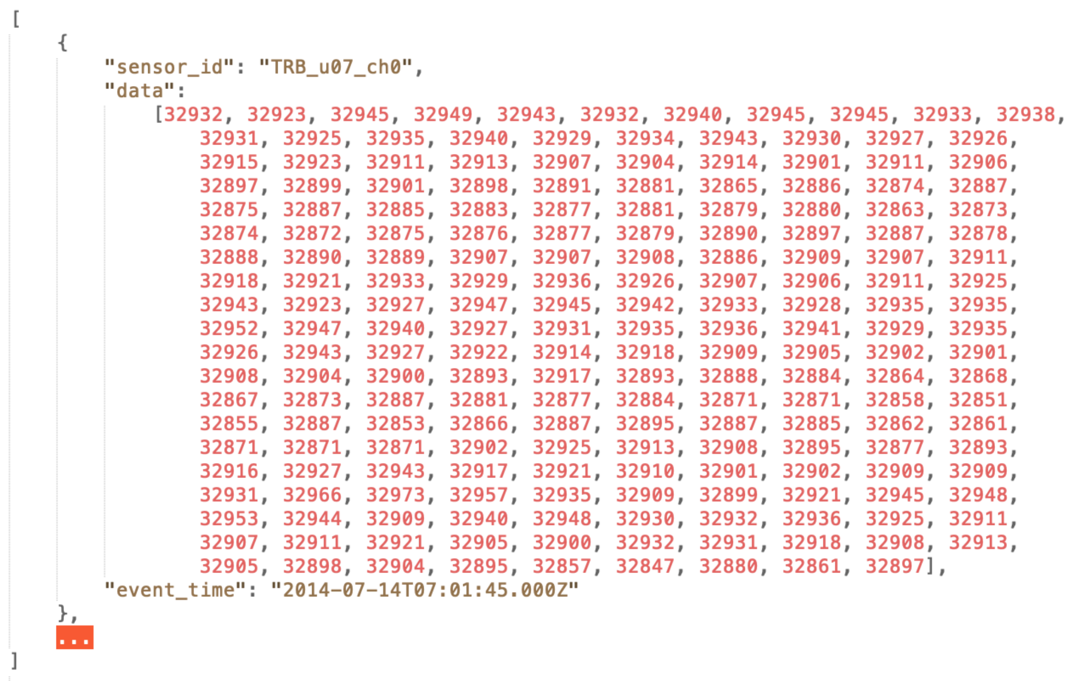
First, the training dataset is retrieved and loaded on the high-performance computer from the cyberinfrastructure platform over the Internet as a web service. To retrieve the sensor data of a specific sensor, the application service implemented on the high-performance computer invokes the sensor data retrieval service on the cyberinfrastructure platform by sending an HTTP request:

```
GET /sensordata/TRB_u07_ch0?
event_time_begin=2014-07-14T00:00:00.000z&
event_time_end=2014-07-14T23:59:59.000z HTTP/1.1
HOST: <cyberinfrastructure platform address>
```

The query includes parameters, such as sensor ID (`TRB_u07_ch0`) and the time period (from `2014-07-14T00:00:00.000z` to `2014-07-14T23:59:59.000z`) when the sensor data is collected. The cyberinfrastructure platform then returns corresponding sensor data encoded in JSON format, as shown in Figure 5.14. The data retrieval request is repeated for each of the input and output sensors. The retrieved sensor data is then mapped to a matrix object (e.g., Numpy matrix object) that can be read by a machine learning tool.

With the retrieved data, a high-performance computer is employed to train the sensor data reconstruction model, following the forward and backward passes and the parameter update procedure, as described in Section 5.2.3.

Once trained, the BRNN model and its metadata are uploaded and stored to the cyberinfrastructure platform. The parameters (i.e., the weights and biases) and the metadata of the trained model is mapped into a JSON format. For example, Figure 5.15 shows a JSON object containing a BRNN model, which has an output sensor `TRB_u191_ch0` and 11 input sensors (`TRB_u07_ch0`, `TRB_u131_ch0`, …), and its metadata. In the current

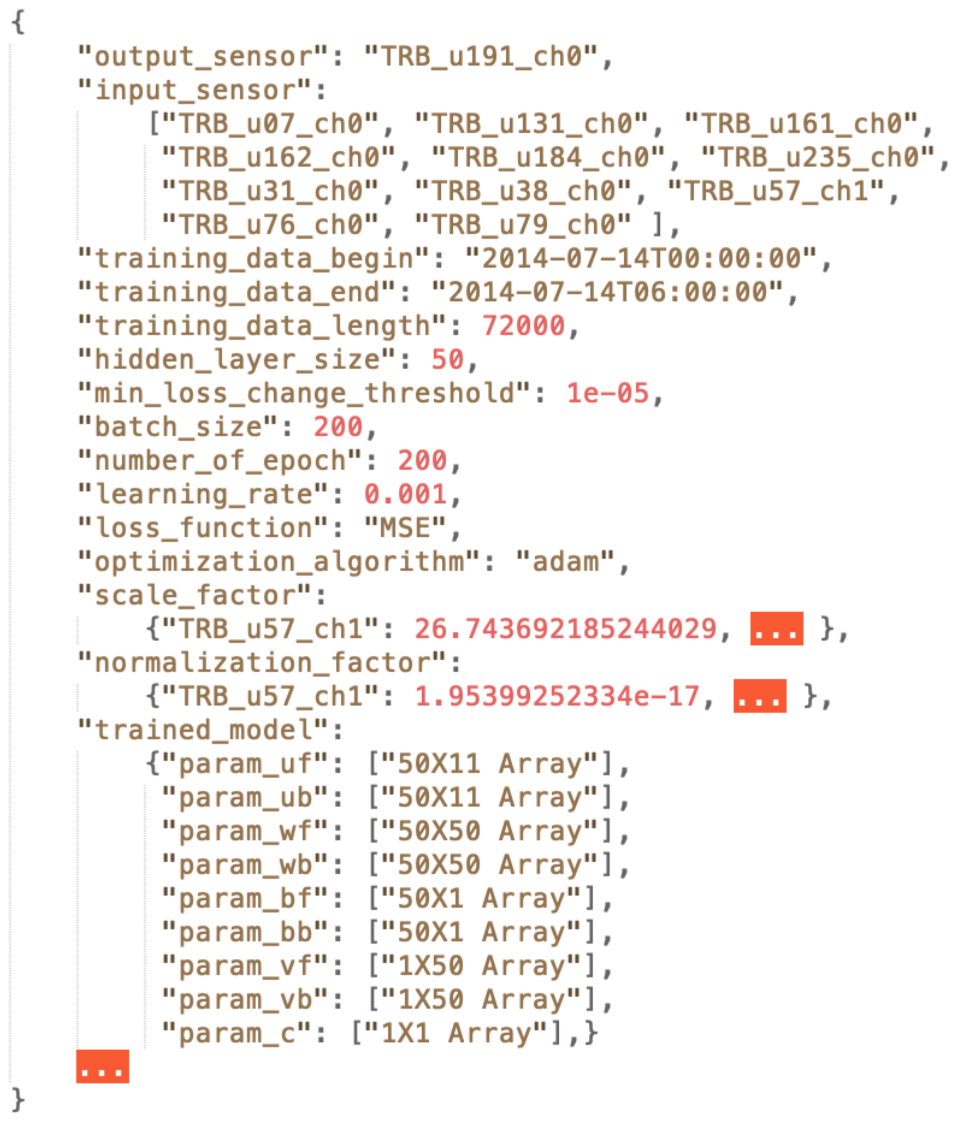


```
[
    {
        "sensor_id": "TRB_u07_ch0",
        "data":
            [32932, 32923, 32945, 32949, 32943, 32932, 32940, 32945, 32945, 32933, 32938,
                32931, 32925, 32935, 32940, 32929, 32934, 32943, 32930, 32927, 32926,
                32915, 32923, 32911, 32913, 32907, 32904, 32914, 32901, 32911, 32906,
                32897, 32899, 32901, 32898, 32891, 32881, 32865, 32886, 32874, 32887,
                32875, 32887, 32885, 32883, 32877, 32881, 32879, 32880, 32863, 32873,
                32874, 32872, 32875, 32876, 32877, 32879, 32890, 32897, 32887, 32878,
                32888, 32890, 32889, 32907, 32907, 32908, 32886, 32909, 32907, 32911,
                32918, 32921, 32933, 32929, 32936, 32926, 32907, 32906, 32911, 32925,
                32943, 32923, 32927, 32947, 32945, 32942, 32933, 32928, 32935, 32935,
                32952, 32947, 32940, 32927, 32931, 32935, 32936, 32941, 32929, 32935,
                32926, 32943, 32927, 32922, 32914, 32918, 32909, 32905, 32902, 32901,
                32908, 32904, 32900, 32893, 32917, 32893, 32888, 32884, 32864, 32868,
                32867, 32873, 32887, 32881, 32877, 32884, 32871, 32871, 32858, 32851,
                32855, 32887, 32853, 32866, 32887, 32895, 32887, 32885, 32862, 32861,
                32871, 32871, 32871, 32902, 32925, 32913, 32908, 32895, 32877, 32893,
                32916, 32927, 32943, 32917, 32921, 32910, 32901, 32902, 32909, 32909,
                32931, 32966, 32973, 32957, 32935, 32909, 32899, 32921, 32945, 32948,
                32953, 32944, 32909, 32940, 32948, 32930, 32932, 32936, 32925, 32911,
                32907, 32911, 32921, 32905, 32900, 32932, 32931, 32918, 32908, 32913,
                32905, 32898, 32904, 32895, 32857, 32847, 32880, 32861, 32897],
        "event_time": "2014-07-14T07:01:45.000Z"
    },
    ...
]
```

Figure 5.14 Sensor data (JSON format) retrieved from the cyberinfrastructure platform by invoking sensor data retrieval service

implementation, the metadata stored along with the BRNN model includes the following information:

- Input and output: the ID of the target output sensor (`output_sensor`) and IDs of the input sensors (`input_sensor`).
- Training dataset information: the time period when the training dataset is collected (`training_data_begin` and `training_data_end`), the number of data points per sensor (`training_data_length`) and scaling and normalization factors (`scaling_factor` and `normalization_factor`).
- Hyper-parameters relevant to training procedure: the number of hidden units (`hidden_layer_size`), size of the batch (`batch_size`), stopping criteria (`number_of_epoch` and `min_loss_change_threshold`), learning rate (`learning_rate`), loss function (`loss_function`) and optimization procedure (`optimization_algorithm`).



```
{
    "output_sensor": "TRB_u191_ch0",
    "input_sensor":
        ["TRB_u07_ch0", "TRB_u131_ch0", "TRB_u161_ch0",
         "TRB_u162_ch0", "TRB_u184_ch0", "TRB_u235_ch0",
         "TRB_u31_ch0", "TRB_u38_ch0", "TRB_u57_ch1",
         "TRB_u76_ch0", "TRB_u79_ch0" ],
    "training_data_begin": "2014-07-14T00:00:00",
    "training_data_end": "2014-07-14T06:00:00",
    "training_data_length": 72000,
    "hidden_layer_size": 50,
    "min_loss_change_threshold": 1e-05,
    "batch_size": 200,
    "number_of_epoch": 200,
    "learning_rate": 0.001,
    "loss_function": "MSE",
    "optimization_algorithm": "adam",
    "scale_factor":
        {"TRB_u57_ch1": 26.743692185244029, ... },
    "normalization_factor":
        {"TRB_u57_ch1": 1.95399252334e-17, ... },
    "trained_model":
        {"param_uf": ["50X11 Array"],
         "param_ub": ["50X11 Array"],
         "param_wf": ["50X50 Array"],
         "param_wb": ["50X50 Array"],
         "param_bf": ["50X1 Array"],
         "param_bb": ["50X1 Array"],
         "param_vf": ["1X50 Array"],
         "param_vb": ["1X50 Array"],
         "param_c": ["1X1 Array"],}
    ...
}
```

Figure 5.15 Trained BRNN model and metadata encoded in JSON format

The parameters of the BRNN model are mapped into JSON format using double-precision array data type. For example, the weights $\mathbf{U}^{\mathrm{f}}, \mathbf{U}^{\mathrm{b}}, \mathbf{W}^{\mathrm{f}}, \mathbf{W}^{\mathrm{b}}, \boldsymbol{v}^{f}$ and $\boldsymbol{v}^{b}$ of the BRNN model are mapped into the arrays `param_uf`, `param_ub`, `param_wf`, `param_wb`, `param_vf` and `param_vb`, respectively. The biases $\boldsymbol{b}^{f}, \boldsymbol{b}^{b}$ and c of the BRNN model are mapped into the arrays `param_bf`, `param_bb` and `param_c`, respectively.

The JSON-encoded data is transmitted to the cyberinfrastructure platform by invoking a model storing service via an HTTP request structured as:

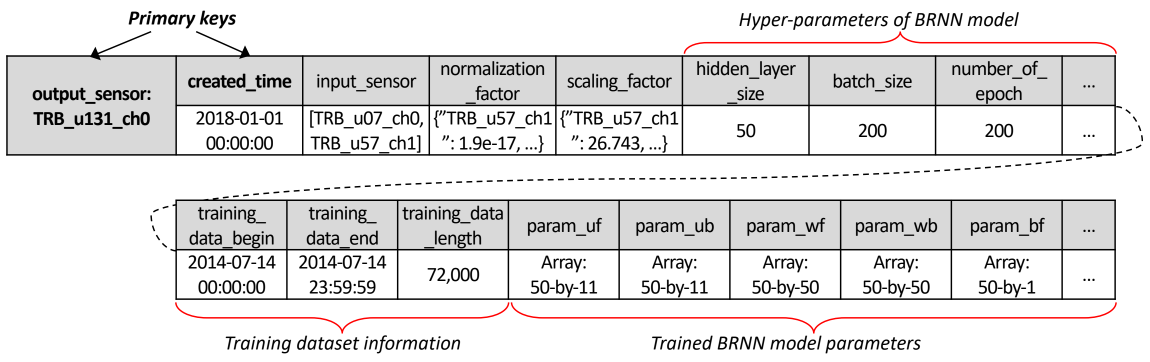
```
POST /trainedmodel HTTP/1.1
HOST: <cyberinfrastructure platform address>
<A blank line separating header and body>
<Attachment: JSON data>
```

This request includes query the JSON-encoded data as an attachment. Once the data is transmitted, the cyberinfrastructure platform parses the BRNN model and its metadata and stores them in a model database, as discussed next in Section 5.4.2.

5.4.2 Storing the trained model on cyberinfrastructure platform

The trained models and their metadata are managed on the cyberinfrastructure platform so that the model can be shared and used by different machines and devices. Web services are implemented on the cyberinfrastructure platform to facilitate storing and retrieving the trained models. Receiving the HTTP request for storing trained model, the corresponding web service is invoked and parses the JSON data attached in the request. Furthermore, a database schema is defined for managing the trained models. The web service generates a query statement to store the parsed data in the database (Cassandra database in current implementation) on the cyberinfrastructure platform.

Figure 5.16 shows the database schema defined for storing trained model (i.e., trained weights and biases) along with metadata needed for querying and utilization of the model. In this schema, the list of input sensors and the output sensor are recorded to identify the input-output sensors of the model. The normalization and scaling factors are stored for the processing of sensor data in the sensor reconstruction phase. In addition, the BRNN model's hyper-parameters (e.g., hidden layer size, batch size, number of epochs, minimum loss change threshold, learning rate, loss function and optimization algorithm) and


Primary keys
Hyper-parameters of BRNN model
Training dataset information
Trained BRNN model parameters

output_sensor: TRB_u131_ch0	created_time	input_sensor	normalization_factor	scaling_factor	hidden_layer_size	batch_size	number_of_epoch	...
	2018-01-01 00:00:00	[TRB_u07_ch0, TRB_u57_ch1]	{"TRB_u57_ch1": 1.9e-17, ...}	{"TRB_u57_ch1": 26.743, ...}	50	200	200	...

training_data_begin	training_data_end	training_data_length	param_uf	param_ub	param_wf	param_wb	param_bf	...
2014-07-14 00:00:00	2014-07-14 23:59:59	72,000	Array: 50-by-11	Array: 50-by-11	Array: 50-by-50	Array: 50-by-50	Array: 50-by-1	...

Figure 5.16 Data schema for storing sensor data reconstruction model

information on the training dataset (e.g., the period when the training dataset is collected and the length of the training dataset) are stored to provide the information about the trained model. For different versions of the models (for example using different time periods for the same input and output sensors), the created time attribute is defined as a primary key for the model version. Each of the parameters of BRNN model is stored in a column, as shown in Figure 5.16. Two-dimensional double-precision array data type is employed to store the weights, while one-dimensional double-precision array data type is used to store the biases.

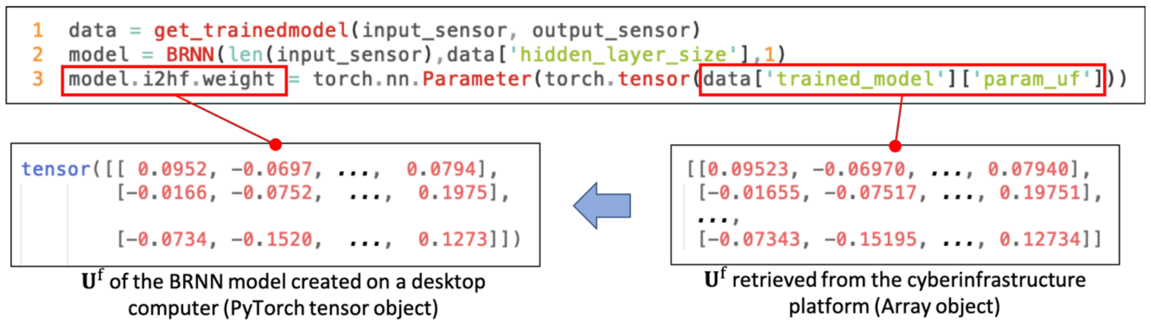
Once the trained model is stored in the database, the model stored can be retrieved and used by different client devices by invoking the model retrieval service, which is discussed in the next section.

5.4.3 Sensor data reconstruction on local computer

With the trained sensor data reconstruction model archived on the cyberinfrastructure platform, the model can be retrieved as needed by any devices for reconstruction of the data for the output sensors. Here, a desktop computer is employed to illustrate the sensor data reconstruction process. For retrieving the trained model, the desktop computer first invokes the model retrieval service by sending an HTTP request to the cyberinfrastructure platform as follows:

```
GET /trained_model?output_sensor=TRB_u131_ch0?
input_sensor=TRB_u07_ch0&
input_sensor=TRB_u57_ch1&
input_sensor= ... <repeat for all input sensors>
HTTP/1.1
HOST: <cyberinfrastructure platform address>
```

This request includes query parameters, such as the list of the input sensors and the target output sensor. The model retrieval service retrieves the trained model corresponding to the query parameters from the database and returns the trained model with the metadata encoded in the same JSON format used when the model is uploaded (see Figure 5.15). The BRNN model parameters (i.e., weight and bias) stored in a double-precision array data type are mapped into a BRNN model object that can be read by the scoring engine to be created on the desktop for reconstructing the data of the output sensor. Figure 5.17, for example, shows a snippet of Python script that maps the weight $\mathbf{U}^f$ of the retrieved BRNN model into the weight $\mathbf{U}^f$ of a BRNN model object created on desktop computer using PyTorch [159]. The first line of the script retrieves the trained model and the metadata from the cyberinfrastructure platform by invoking the model retrieval service. The second line of the script creates a BRNN model object based on the number of input sensors and the number of hidden units. The third line of the script converts $\mathbf{U}^f$ (i.e.,



```
1  data = get_trainedmodel(input_sensor, output_sensor)
2  model = BRNN(len(input_sensor),data['hidden_layer_size'],1)
3  model.i2hf.weight = torch.nn.Parameter(torch.tensor(data['trained_model']['param_uf']))
```

```
tensor([[ 0.0952, -0.0697, ...,  0.0794],
        [-0.0166, -0.0752,  ...,  0.1975],

        [-0.0734, -0.1520,  ...,  0.1273]])
```

```
[[0.09523, -0.06970, ..., 0.07940],
 [-0.01655, -0.07517, ..., 0.19751],
 ...,
 [-0.07343, -0.15195, ..., 0.12734]]
```

$\mathbf{U}^f$ of the BRNN model created on a desktop computer (PyTorch tensor object)
$\mathbf{U}^f$ retrieved from the cyberinfrastructure platform (Array object)

Figure 5.17 Data mapping from the retrieved BRNN parameters to a BRNN model object created using PyTorch [153]

`data['trained_model']['param_uf']`) of the retrieved BRNN model to a PyTorch Parameter object and copies it to $\mathbf{U}^{f}$ (i.e., `new_model.i2hf.weight`) of the BRNN model object created on desktop computer. In this way, a BRNN model created on a desktop computer can duplicate the weights and biases of the trained model retrieved from the cyberinfrastructure platform.

To perform sensor data reconstruction using the retrieved model, the testing dataset collected for the desired period needs to be downloaded from the cyberinfrastructure platform. For each sensor, sensor data can be retrieved by invoking the sensor data retrieval service of the cyberinfrastructure platform as follows:

```
GET /sensordata/TRB_u07_ch0?
event_time_begin=2014-07-15T00:00:00.000z&
event_time_end=2014-07-15T23:59:59.000z HTTP/1.1
HOST: <cyberinfrastructure platform address>
```

This web service request specifies the sensor ID and the desired time period. Once retrieved, the testing dataset is scaled and normalized using the scaling and normalization factors retrieved with the BRNN model. The testing dataset is then fed into the retrieved sensor data reconstruction model, which returns the reconstructed sensor data of the output sensor.

In summary, with the cyberinfrastructure platform developed in this study, the BRNN-based sensor data reconstruction can be flexibly conducted in a decentralized manner. Computationally intensive training process is conducted on a high-performance (but more expensive) computer, while less computationally demanding sensor data reconstruction task is performed on any device by retrieving the trained model from the cyberinfrastructure platform. Machine learning model management services are built to store and to share the trained model via the cyberinfrastructure platform.

5.5 Application on a Bridge in Service

This section demonstrates the BRNN-based sensor data reconstruction method using the sensor data collected on the Telegraph Road Bridge (Monroe, Michigan) [156, 157]. For this study, the vibration response datasets collected during the month of July 2014, which includes 700,000 data points per sensor, is downloaded from the cyberinfrastructure platform and used (as shown in Figure 5.18). It should be noted that Sensors A4 and A12 are down most of the time on that month.

The BRNN models are created with the same hyperparameters (i.e., number of forward hidden units and backward hidden units, the activation functions, loss function, learning rate and the maximum number of epochs) and optimization schema as described in Section 5.3. In addition to the evaluation of BRNN-based sensor data reconstruction, this section describes the faulty data recovery process for the missing data of Sensors A4 and A12. Furthermore, this section investigates environmental effects (e.g., temperature change) to the BRNN-based sensor data reconstruction. For the training of BRNN models, a computer

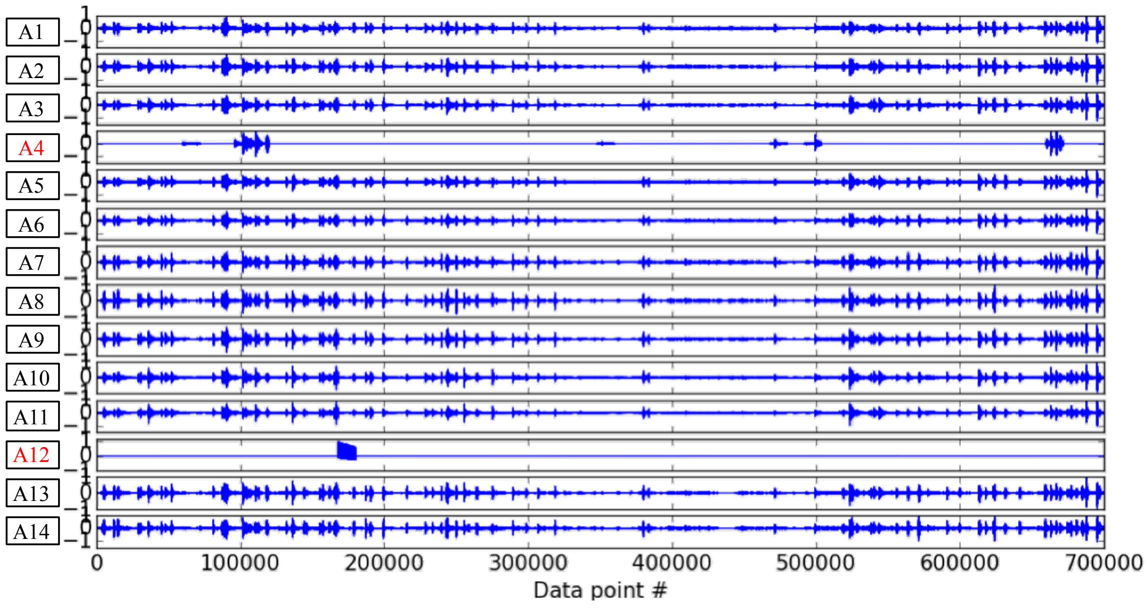


Figure 5.18 Vertical acceleration measurement of the Telegraph Road Bridge (July 2014)

with high-performance CPU (Intel Core i7-7820X 3.60GHz) and a high-performance GPU (GTX 1080 Ti) is used. For reconstructing sensor data, a laptop computer with Intel Core i7-4870HQ 2.5 GHz is employed.

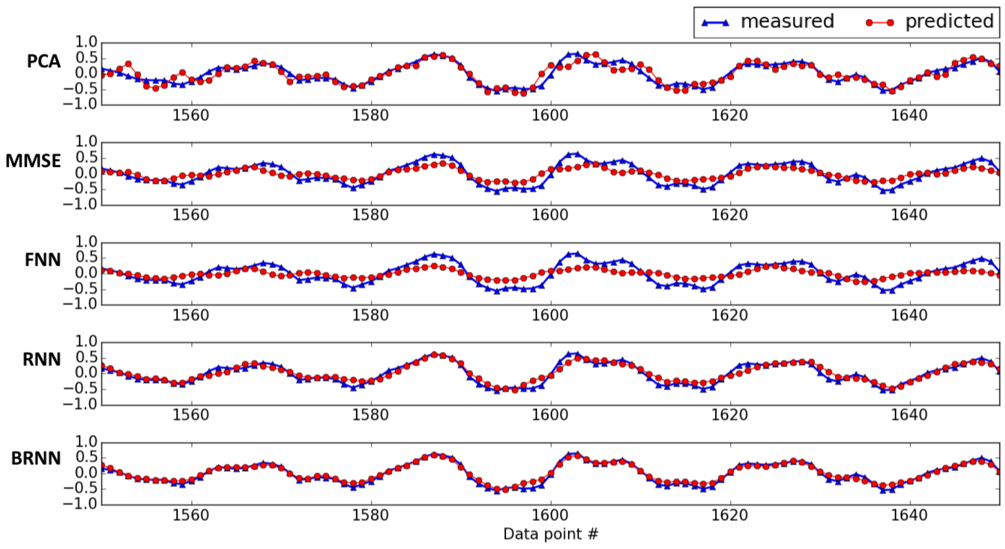
5.5.1 Comparison of BRNN-based sensor data reconstruction with other methods

For testing purpose, two BRNN models are created to reconstruct data of Sensor A1 (located near the support of the leftmost girder) and Sensor A11 (located at the middle of the rightmost girder), respectively, using all other sensors as input sensors. In addition, other data-driven models (including PCA, MMSE, FNN and RNN) corresponding to the BRNN models are created for comparison. All the models are trained with a training dataset containing 100,000 data points (from the data points 1 to 100,000) per sensor. The trained models are evaluated with a testing dataset, which contains 40,000 data points (from the data points 100,001 to 140,000) per sensor.

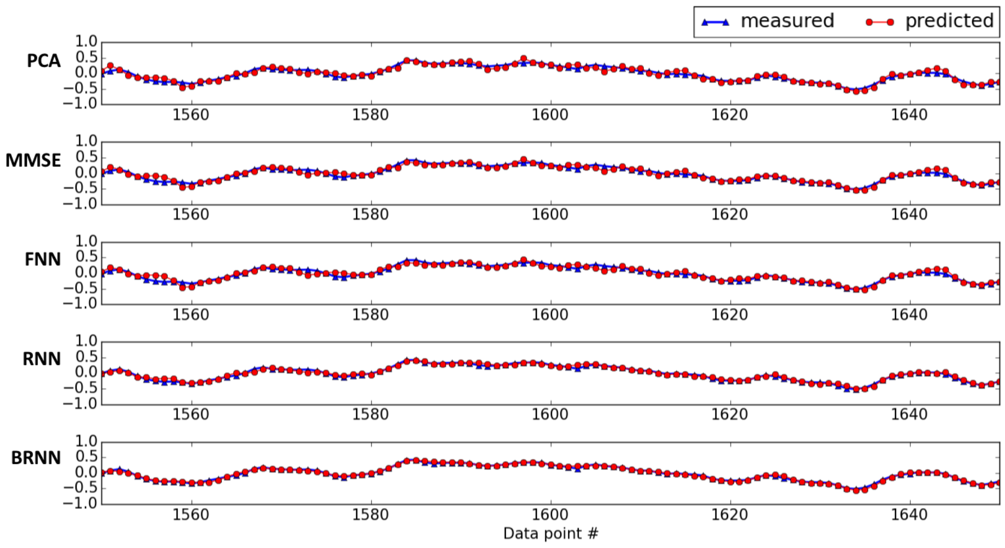
Figure 5.19 shows the results of data reconstruction using the different methods. As shown in Figure 5.19(a), the BRNN model reconstructs the data for Sensor A1 most accurately among the tested methods. For Sensor A11, all methods work well, but the BRNN- and RNN-based method show slightly better fit comparing to the other methods. Table 5.4 shows the testing errors of different sensor data reconstruction methods. Similar to the numerical simulation results presented earlier, the testing error for the sensor (i.e., Sensor A1) located near the support of the bridge is generally higher than the sensor (i.e., Sensor A11) located at the center of the bridge because the spatial correlation is weak for the sensor at the end of the bridge span. Generally, the results show that the BRNN-based method outperforms other methods in terms of data reconstruction accuracy.

Table 5.4 Bridge structure: testing error

	PCA	**MMSE**	**FNN**	**RNN**	**BRNN**
Testing RMSE for Sensor A1	0.085	0.052	0.053	0.035	0.029
Testing RMSE for Sensor A11	0.030	0.028	0.030	0.018	0.014



(a) Sensor A1



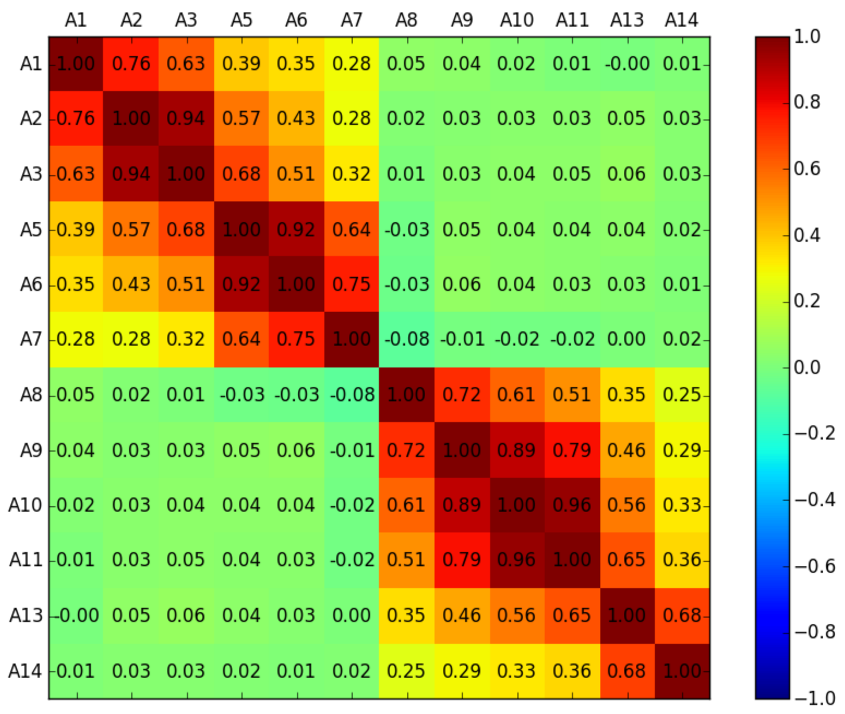
(b) Sensor A11

Figure 5.19 Bridge structure: sensor data reconstruction with the different methods

5.5.2 Effects of using different set of input sensors

So far, the sensor data reconstruction is demonstrated using all available sensors as input sensors, which assumes that all sensors can contribute to the reconstruction of output sensor's data. However, in reality, only some subset of sensors would contribute on the reconstruction. Therefore, it would be more practical to use the subset of sensors with high contribution because this can reduce the size of the BRNN model, particularly, when there are many sensors in the sensor network. Furthermore, by using a subset of sensors as input sensors, sensor data reconstruction model can still be used even when some of non-input sensors in the network are faulty. To use a subset of sensors as input sensors, it is very important to choose an appropriate subset that can reconstruct sensor data as accurate as when using all sensors as input sensors. Since the BRNN-based sensor data reconstruction is based on the spatiotemporal correlation, one plausible approach is to choose sensors that have high correlation with the output sensors as the input sensors.

Figure 5.20 shows the covariance matrix among sensor data collected from twelve sensors at the Telegraph Road Bridge. Here, the covariance matrix is computed using 100,000 data points (from the data points 1 to 100,000) per sensor. From the covariance matrix, it can be seen that the data from sensors along the same girder are highly correlated, while the data from the sensors on different girder are not correlated well. The matrix also shows that



	A1	A2	A3	A5	A6	A7	A8	A9	A10	A11	A13	A14
A1	1.00	0.76	0.63	0.39	0.35	0.28	0.05	0.04	0.02	0.01	-0.00	0.01
A2	0.76	1.00	0.94	0.57	0.43	0.28	0.02	0.03	0.03	0.03	0.05	0.03
A3	0.63	0.94	1.00	0.68	0.51	0.32	0.01	0.03	0.04	0.05	0.06	0.03
A5	0.39	0.57	0.68	1.00	0.92	0.64	-0.03	0.05	0.04	0.04	0.04	0.02
A6	0.35	0.43	0.51	0.92	1.00	0.75	-0.03	0.06	0.04	0.03	0.03	0.01
A7	0.28	0.28	0.32	0.64	0.75	1.00	-0.08	-0.01	-0.02	-0.02	0.00	0.02
A8	0.05	0.02	0.01	-0.03	-0.03	-0.08	1.00	0.72	0.61	0.51	0.35	0.25
A9	0.04	0.03	0.03	0.05	0.06	-0.01	0.72	1.00	0.89	0.79	0.46	0.29
A10	0.02	0.03	0.04	0.04	0.04	-0.02	0.61	0.89	1.00	0.96	0.56	0.33
A11	0.01	0.03	0.05	0.04	0.03	-0.02	0.51	0.79	0.96	1.00	0.65	0.36
A13	-0.00	0.05	0.06	0.04	0.03	0.00	0.35	0.46	0.56	0.65	1.00	0.68
A14	0.01	0.03	0.03	0.02	0.01	0.02	0.25	0.29	0.33	0.36	0.68	1.00

Figure 5.20 Covariance matrix of sensor data collected from the Telegraph Road Bridge

the data from adjacent sensors (e.g., Sensors A1 and A2) on the same girder have higher correlations than the data from distant sensors (e.g., between Sensors A1 and A7) even on the same girder.

To study the effect of using only the highly correlated sensors, sensor data reconstruction for sensor A1 is performed with two BRNN models. The first model has all available sensors (i.e., Sensors A2, …, A14) as inputs, while the second model has only the highly correlated sensors (i.e., Sensors A2, …, A7) as inputs. Both models are trained using 100,000 data points (from the data points 1 to 100,000) per sensor. The testing data is selected from the data points 100,001 to 140,000. Figure 5.21 shows data reconstructed for sensor A1 by each model.

Table 5.5 shows the reconstruction error of the two models. The results show that the model constructed using the data from the highly correlated subset of sensors reconstructs the data as accurate as the model constructed using all the sensors. This result shows that it is appropriate to select the input sensors by examining their correlation with the output sensors.

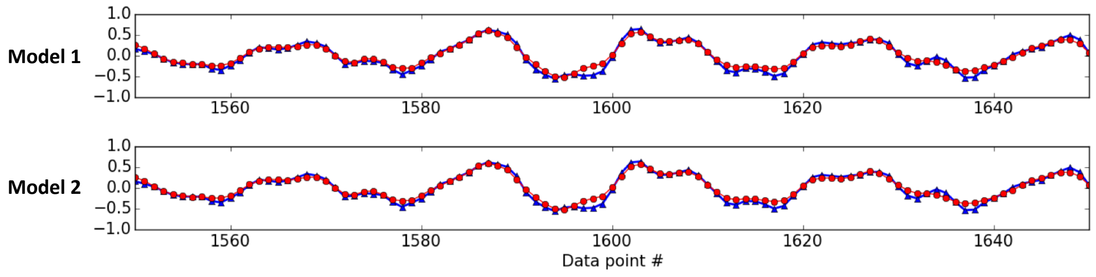


Figure 5.21 Sensor data reconstruction with different inputs

Table 5.5 Testing error for different input combinations

	Model 1	**Model 2**
Input Sensors	All sensors (A2, …, A14)	Highly correlated sensors (A2, …, A7)
Testing RMSE for Sensor A1	0.0288	0.0292

5.5.3 Environmental effect

For bridges which are in service, environmental condition (e.g., temperature) varies throughout the year [162]. A sensor data reconstruction method needs to be robust against the environmental effects. In this section, a study is conducted to see whether the BRNN models trained with data of a certain month can be used to reconstruct the data for other months. For the study, a BRNN model with Sensors A7, A9 and A10 as input sensors and Sensor A8 as the output sensor is constructed using the hyperparameters defined in Section 5.3.

Figure 5.22 shows the temperature change measured by a thermistor on the Telegraph Road Bridge from October 2014 to May 2015. Ten BRNN models are created for each selected month as highlighted in Figure 5.22. Each BRNN model is trained with 100,000 data points per sensor collected from the corresponding month. In addition, a yearly model is created and trained using the training data containing 100,000 data points per sensor by concatenating the five datasets collected from five different months, including April, July, October and December of 2014 and February of 2015. Using the 11 BRNN models (i.e.,

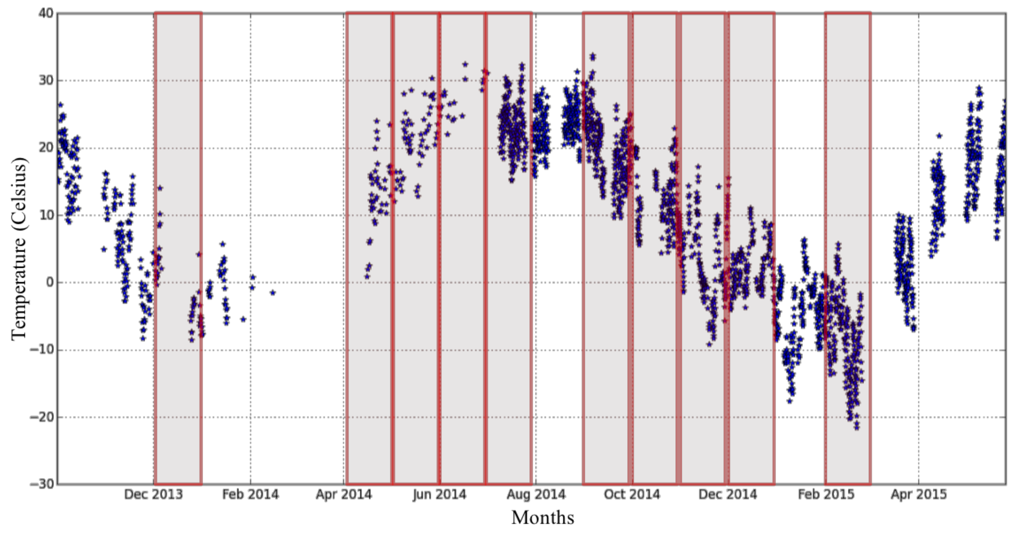


Figure 5.22 Temperature change measured on the Telegraph Road Bridge from October 2014 to May 2015 (Months selected for analysis are highlighted with boxes)

10 monthly models and one yearly model), sensor data reconstructions are performed for the 10 testing datasets, each of which has 40,000 data points per sensor collected from one of the 10 selected months. Figure 5.23 shows the testing errors of the 11 BRNN models for the 10 testing datasets. It can be seen that the testing errors of all 11 BRNN models are similar for each month. This result shows that, for the data considered at this bridge site, the BRNN-based method is robust against the change of environmental conditions.

5.5.4 Reconstruction of missing sensor data

Sensor data reconstruction can be used for recovering missing data when some sensors are faulty. As shown in Figure 5.18, Sensors A4 and A12 were not working properly in the month of July 2014 and the data is severely erroneous. The missing data can be recovered using BRNN-based sensor data reconstruction method. The basic idea is to train a BRNN model using good quality datasets collected from other dates.

Figure 5.24 depicts the sensor data reconstruction process for recovering the missing data of July 2014 by creating and training two BRNN models corresponding to the two target sensors A4 and A12, respectively. The first BRNN model is constructed with input sensors A2, A3, A5, A6 and A7, which are located on the same girder as the output sensor, and an output Sensor A4. The trained model is constructed using 60,000 data points per sensor, collected in October 2014 when all of input and output sensors are operating properly.

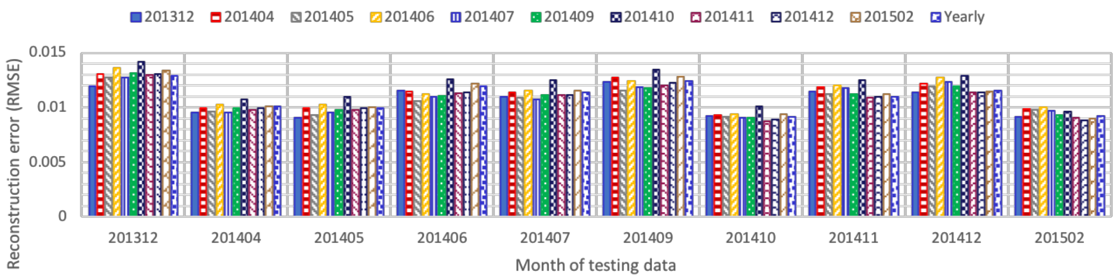


Figure 5.23 Reconstruction errors for testing sets collected from different months: 10 BRNN models trained with monthly data and one BRNN model trained with yearly data are used

Once the first BRNN model is trained, the data (700,000 data points per sensor) collected by Sensors A2, A3, A5, A6 and A7 collected in July 2014 is inputted to the first model to reconstruct the data of Sensor A4 in July 2014. The recovered dataset of Sensor A4 is shown in Figure 5.24. Similarly, the second BRNN model is constructed with input sensors A8, A9, A10, A11, A13 and A14 and an output Sensor A12. The second BRNN model is trained using 60,000 data points per sensor, collected in July 2012 when the input and output sensors are operating properly. The data (700,000 data points per sensor) collected by Sensors A8, A9, A10, A11, A13 and A14 collected in July 2014 is then inputted to the second model to recover the data of Sensor A12 in July 2014. The recovered dataset of Sensor A12 is also shown in Figure 5.24.

The recovered data, which now includes the data for all 14 sensors, can be further analyzed to check the integrity of the target system. For example, the bridge's operational mode shapes can now be computed using the recovered dataset from July 2014 by system identification, for example, using the frequency domain decomposition (FDD) method

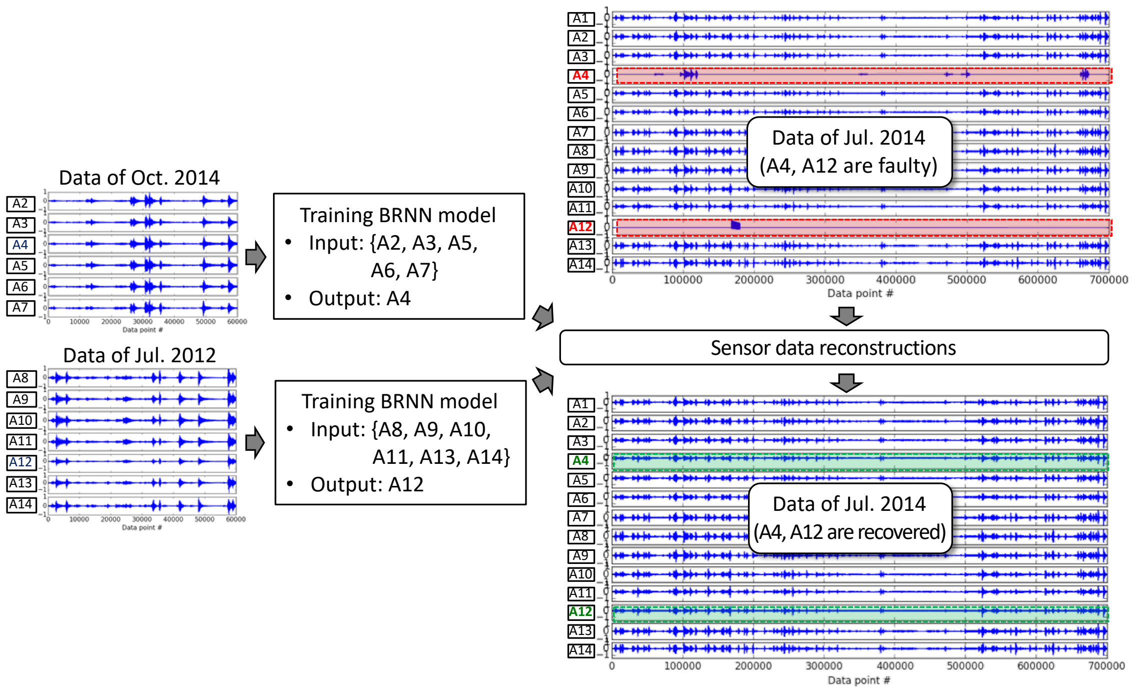
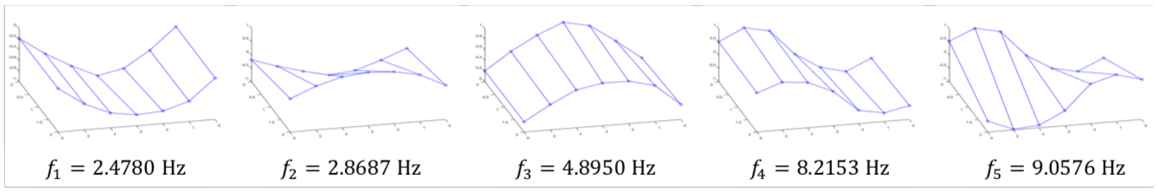

Data of Oct. 2014
A2
A3
A4
A5
A6
A7
Data point #
Training BRNN model
• Input: {A2, A3, A5, A6, A7}
• Output: A4
Data of Jul. 2012
A8
A9
A10
A11
A12
A13
A14
Training BRNN model
• Input: {A8, A9, A10, A11, A13, A14}
• Output: A12
Data of Jul. 2014 (A4, A12 are faulty)
Sensor data reconstructions
Data of Jul. 2014 (A4, A12 are recovered)
Data point #

Figure 5.24 Reconstruction of missing data using BRNN models

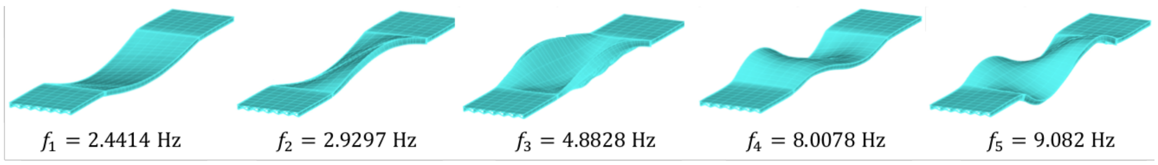
[163]. Figure 5.25 shows the first five modal frequencies and operational mode shapes computed with the recovered sensor data along with the corresponding modal frequencies and operational mode shapes experimentally measured and reported in [164]. As shown in Figure 5.25, the modal frequencies and operational mode shapes computed using the recovered datasets match very well with the bridge's original mode shape and frequencies obtained experimentally.

5.6 Summary

This chapter describes a data-driven sensor data reconstruction method and its implementation using the cyberinfrastructure platform. Sensor data reconstruction is an important task for the operation of sensor network to recover missing or faulty sensor data and detect anomalies. For the precise sensor data reconstruction, this study presents a data-driven sensor data reconstruction approach based on bidirectional recurrent neural network (BRNN). The BRNN-based sensor data reconstruction method is a supervised regression



(a) Modal frequency and operational mode shapes computed from the reconstructed dataset



(b) Modal frequency and operational mode shapes obtained from experiments [161]

Figure 5.25 First five modal frequencies and operational mode shapes of the Telegraph Road Bridge

problem that reads input sensors' data and reconstructs the target output sensor's data. Specifically, the BRNN-based method reconstructs sensor data based on the spatiotemporal correlation among the sensor data collected by a distributed network of sensors. The BRNN-based sensor data reconstruction procedure consists of two phases: training phase and reconstruction phase. In the training phase, a BRNN model is trained using a training dataset by adjusting the weights and biases of the model to minimize the difference between the reconstructed data and the measured data of the target output sensor. In the reconstruction phase, the target output sensor's data is reconstructed by feeding the input sensors data to the trained BRNN model. To validate the BRNN-based sensor data reconstruction approach, demonstrations are conducted using the bridge vibration response data collected from numerical simulations. The demonstration results show that the BRNN-based sensor data reconstruction yields more accurate estimation than other existing data reconstruction methods (e.g., PCA, MMSE, FNN and RNN). Furthermore, the BRNN-based sensor validation can capture and localize anomalies due to faulty sensors.

While the use of an artificial neural network is likely to improve the data reconstruction accuracy, the training of the artificial neural network is typically computationally demanding. For the effective data-driven sensor data reconstruction, this study presents a data analysis pipeline that automates data and computational flow over high-performance computer and client devices via the cyberinfrastructure platform. In the training phase, the high-performance computer retrieves a training dataset from the cyberinfrastructure platform and trains the sensor data reconstruction models. The trained models are uploaded and stored using the cyberinfrastructure platform. In the reconstruction phase, client devices (e.g., desktop computer) downloads a testing dataset, as well as the trained data reconstruction model from the cyberinfrastructure platform, and performs data reconstruction. In this way, a computationally demanding model training task can be performed on a high-performance computer for fast learning process, whereas client devices can perform only a less computationally demanding reconstruction task. The data analysis pipeline is demonstrated with the sensor data collected from the Telegraph Road Bridge.

While the validation results are promising, there is still much room for research for the sensor data reconstruction problem. The demonstrations in this work are conducted using only vibration data. The use of BRNN-based method for reconstructing different types of data (e.g., wind data collected from anemometer) should be investigated. In addition, the demonstrations in this work mainly focus on bridge monitoring application. The proposed method can be potentially useful to different sensing applications wherein sensors have spatial and temporal correlations.

Chapter 6

Conclusions

6.1 Summary of Results and Contributions

As microelectromechanical systems (MEMS), semiconductor and network communication technologies have greatly enhanced the developments of sensors and sensor networks that are now being deployed widely in engineering systems, advanced data management and software technologies can further extend the functionalities of engineering systems. Engineering applications are typically data-intensive and computationally demanding services that require not only continuing advances in engineering modeling and sensor development but also data and software frameworks to facilitate workflow and resource management. This thesis presents a cyberinfrastructure platform tailored to civil infrastructure monitoring applications. The cyberinfrastructure platform is designed to meet the data management requirements, which are scalability, data integration, interoperability, standardized interfaces and flexibility.

In Chapter 2, an information modeling framework for bridge monitoring application is described. Bridge monitoring 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. In current practice, the different types of information are typically managed by isolated systems, which ends up with inefficient data utilization and integration. In order to address the data sharing issue and guarantees data interoperability, this chapter presents a bridge information modeling framework for supporting bridge monitoring applications. The framework employs the OpenBrIM data model [81] as the base model, and enrich the base model by defining data entities for the representation of engineering model and sensor description. Specifically, the framework draws on the data entities of CSiBridge [79] for the representation of engineering model and SensorML [80] for the representation of sensor information. The demonstration shows that the proposed BrIM data schema can capture bridge information, including geometry, engineering model and sensor description, about a bridge structure instrumented with structural monitoring system.

In Chapter 3, a NoSQL-based data management system for the scalable management of sensor data and relevant information is presented. Since engineering 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. Based on the data management requirements defined for civil infrastructure monitoring, Apache Cassandra database and MongoDB are selected as the backend database systems of the cyber bridge monitoring framework. Apache Cassandra is a column family database that is suitable for large-scale distributed database, while MongoDB is a document-oriented database that has advantages on the schema-less data structure and fast performance. Data schemas for Cassandra database and MongoDB are defined following the bridge and sensor information models. The demonstration results show that the NoSQL-based data management platform enables effective data management for a large amount of bridge monitoring data and supports query capability for retrieving various types of data residing in the data management system.

In Chapter 4, a prototype cyberinfrastructure platform is designed and implemented on a cloud computing environment. The cyberinfrastructure platform handles data store and retrieval through three layers: communication layer, mapping layer and storage layer. The communication layer offers standardized, interoperable web-based interfaces to allow various devices can access the platform to store and retrieve different types of data. Specifically, RESTful web services are implemented to support store and retrieval of sensor data and bridge information. The mapping layer handles data mapping processes involved in the management of information models. The storage layer offers scalable data storage by employing a highly scalable distributed database system. The prototype platform leverages the Infrastructure as a Service (IaaS) cloud service model for scalability, reliability and portability. In addition, the platform is implemented on hybrid cloud computing environment to enable decentralized data management where sensitive data is managed by private data center. The demonstration results show that the proposed platform facilitates the data utilization and integration by providing platform-neutral interfaces which can be accessed by various applications on different types of devices.

In Chapter 5, a data-driven sensor data reconstruction method and its implementation using the cyberinfrastructure platform are described. For the accurate sensor data reconstruction, a data-driven method based on the bidirectional recurrent neural network (BRNN) is proposed to reconstruct sensor data based on the spatiotemporal correlation among sensors. The BRNN-based sensor data reconstruction method is tested with vibration response data collected from numerical simulations. The results show that the BRNN-based method reconstructs sensor data more accurately than other existing data-driven sensor data reconstruction methods. For effective learning process, the BRNN-based sensor data reconstruction method is implemented based on a data analysis pipeline that can automate data and computational flow over high-performance computer and client devices through the cyberinfrastructure platform. In this data analysis pipeline, a high-performance computer retrieves training dataset from the cyberinfrastructure platform, trains the BRNN-based model and uploads the trained model to the cyberinfrastructure platform. Client devices (e.g., desktop computers) can download the trained model and the testing dataset from the cyberinfrastructure platform to perform sensor data reconstruction. In this way, computationally demanding training tasks can be handled by high-performance computer, whereas less computationally demanding data reconstruction task can be performed by affordable client devices.

6.2 Future Research Recommendation

The importance of effective data management is growing for facilitating data integration, sharing and utilization. While the proposed cyberinfrastructure platform is designed to handle a large amount of heterogeneous data involved in engineering applications, the platform can be further enhanced. The following discusses several future research directions:

- While the cyberinfrastructure platform is design for the applications of civil infrastructure monitoring, the platform can be easily modified and extended to support data management of other engineering domains. For example, the data management services offered by the platform can be used for the management of factory monitoring data by enriching data schema to represent domain-specific information (e.g., machine information). Therefore, future research may extend the proposed cyberinfrastructure platform for efficient data management for a broad range of different engineering domains.
- 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. Therefore, the information model schema needs to be enhanced to meet the data requirement of other applications and domains.
- Data security is another important topic that was not discussed in this thesis. For secure data management, the proper use of security schemes is indispensable.

Therefore, the proposed cyberinfrastructure platform needs to be enhanced by adopting authentication, authorization and cryptography methodologies.

- While the validation results are promising, there is still much room for research for the sensor data reconstruction problem. The demonstrations in this work are conducted using only vibration data. The use of BRNN-based method for reconstructing different types of data (e.g., wind data collected from anemometer) should be investigated. In addition, the demonstrations in this work mainly focus on bridge monitoring application. The proposed method can be potentially applied to different sensing applications wherein sensors have spatial and temporal correlations.

Bibliography

[1] J. Lynch and K. Loh, "A summary review of wireless sensors and sensor networks for structural health monitoring,," *Shock and Vibration Digest,* vol. 38, no. 2, pp. 91-130, 2006.

[2] G. Zhou and T. Yi, "Recent developments on wireless sensor networks technology for bridge health monitoring," *Mathematical Problems in Engineering,* vol. 2013, pp. 33, 2013.

[3] H. Lim, H. Sohn and P. Liu, "Binding conditions for nonlinear ultrasonic generation unifying wave propagation and vibration," *Applied Physics Letters,* vol. 104, no. 21, pp. 214103, 2014.

[4] E. Cross, K. Koo, J. Brownjohn and K. Worden, "Long-term monitoring and data analysis of the Tamar Bridge," *Mechanical Systems and Signal Processing,* vol. 35, no. 1, pp. 16-34, 2013.

[5] N. Dervilis, K. Worden and E. Cross, "On robust regression analysis as a means of exploring environmental and operational conditions for SHM data," *Journal of Sound and Vibration,* vol. 347, pp. 279-296, 2015.

[6] K. Law, K. Smarsly and Y. Wang, "Sensor data management technologies for infrastructure asset management," in *Sensor Technologies for Civil Infrastructures: Applications in Structural Health Monitoring*, vol. 2, M. Wang, J. P. Lynch and H. Sohn, Eds., Cambridge, Woodhead Publishing, pp. 3-32, 2014.

[7] A. Zaslavsky, C. Perera and D. Georgakopoulos, "Sensing as a service and big data," in *International Conference on Advances in Cloud Computing (ACC-2012)*, Bangalore, 2013.

[8] D. Agrawal, S. Das and A. Abbadi, "Big data and cloud computing: current state and future opportunities," in *The 14th International Conference on Extending Database Technology (EDBT/ICDT '11)*, New York, NY, 2011.

[9] K. Grolinger, W. A. Higashino, A. Tiwari and M. A. Capretz, "Data management in cloud environments: NoSQL and NewSQL data stores," *Journal of Cloud Computing: Advances, Systems and Applications,* vol. 2, no. 22, 2013.

[10] P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal,* vol. 1, no. 1-2, pp. 35-46, 2016.

[11] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering,* vol. 57, no. 3, pp. 221-224, 2015.

[12] S. Kim, C. Kim and J. Lee, "Monitoring results of a self-anchored suspension bridge," in *Sensing Issues in Civil Structural Health Monitoring*, Dordrecht, Springer, pp. 475-484, 2005.

[13] R. Hou, S. Jeong, Y. Wang, K. Law and J. Lynch, "Camera-based triggering of bridge structural health monitoring systems using a cyber-physical system framework," in *International Workshop on Structural Health Monitoring*, Stanford, CA, 2017.

[14] C. Eastman, P. Teicholz, R. Sacks and K. Liston, BIM handbook: a guide to building information modeling for owners, managers, designers, engineers, and contractors, Hoboken, NJ: John Wiley & Sons, Inc., 2011.

[15] S. Chen and A. Shirolé, "Integration of information and automation technologies in bridge engineering and management: Extending the state of the art," *Transportation Research Record,* vol. 1976, no. 1, pp. 3-12, 2006.

[16] M. Beyer, "Gartner says solving 'Big Data' challenge involves more than just managing volumes of data," 27 June 2011. [Online]. Available: https://www.gartner.com/newsroom/id/1731916. [Accessed 1 June 2018].

[17] M. Chen, S. Mao and Y. Liu, "Big data: A survey," *Mobile Networks and Applications,* vol. 19, no. 2, pp. 171-209, 2014.

[18] K. Wong, "Instrumentation and health monitoring of cable-supported bridges," *Structural Control and Health Monitoring,* vol. 11, no. 2, pp. 91-124, 2004.

[19] New Civil Engineer, "Project profile | Queensferry Crossing," 13 September 2017. [Online]. Available: https://www.newcivilengineer. com/tech-excellence/project-profile-queensferry-crossing/10023399.article. [Accessed 11 December 2017].

[20] U. Lieske, A. Dietrich, L. Schubert and B. Frankenstein, "Wireless system for structural health monitoring based on Lamb waves," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, San Diego, CA, 2012.

[21] B. Lin and V. Giurgiutiu, "Development of optical equipment for ultrasonic guided wave structural health monitoring," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, San Diego, CA, 2014.

[22] S. Chen and A. Shirolé, "Implementation roadmap for bridge information modeling (BrIM) data exchange protocols," University at Buffalo, State University of New York, Buffalo, NY, 2013.

[23] C. Eastman, "Building product models: computer environments, supporting design and construction," CRC press, 1999.

[24] M. Lenzerini, "Data integration: A theoretical perspective," in *The Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002.

[25] Amazon Web Services, Inc., "AWS IoT services overview - Amazon Web Services," [Online]. Available: https://aws.amazon.com/iot/. [Accessed 1 February 2018].

[26] Microsoft, "Azure IoT Hub | Microsoft Azure," [Online]. Available: https://azure.microsoft.com/. [Accessed 1 February 2018].

[27] Google Cloud, "Cloud IoT Core," [Online]. Available: https://cloud.google.com/iot-core/.

[28] IBM, "IoT Platform - IBM Watson IoT," [Online]. Available: https://www.ibm.com/internet-of-things/spotlight/watson-iot-platform. [Accessed 1 February 2018].

[29] AT&T, "AT&T IoT Platform - Build solutions for the Internet of Things," [Online]. Available: https://iotplatform.att.com/. [Accessed 1 February 2018].

[30] PTC, "ThingWorx industrial innovation platform | PTC," [Online]. Available: https://www.ptc.com/en/products/iot/thingworx-platform. [Accessed 1 February 2018].

[31] Autodesk, Inc., "Autodesk Fusion Connect. Enterprise IoT software platform," [Online]. Available: https://autodeskfusionconnect.com/. [Accessed 1 February 2018].

[32] G. Yang, L. Xie, M. Mäntysalo, X. Zhou, Z. Pang, L. Xu, S. Kao-Walter, Q. Chen and L. Zheng, "A health-IoT platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box," *IEEE Transactions on Industrial Informatics,* vol. 10, no. 4, pp. 2180-2191, 2014.

[33] C. Doukas and I. Maglogiannis, "Bringing IoT and cloud computing towards pervasive healthcare," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012.

[34] A. Krylovskiy, M. Jahn and E. Patti, "Designing a smart city Internet of Things platform with microservice architecture," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Rome, 2015.

[35] R. Lea and M. Blackstock, "City hub: A cloud-based IoT platform for smart cities," in *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, Singapore, 2014.

[36] G. Suciu, A. Vulpe, S. Halunga, O. Fratu, G. Todoran and V. Suciu, "Smart cities built on resilient cloud computing and secure Internet of Things," in *2013 19th International Conference on Control Systems and Computer Science*, Bucharest, 2013.

[37] P. Jayaraman, D. Palmer, A. Zaslavsky and D. Georgakopoulos, "Do-it-Yourself digital agriculture applications with semantically enhanced IoT platform," in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Singapore, 2015.

[38] J. Brownjohn, A. Zasso, G. Stephen and R. Severn, "Analysis of experimental data from wind-induced response of a long span bridge," *Journal of Wind Engineering and Industrial Aerodynamics,* vol. 54, pp. 13-24, 1995.

[39] C. Farrar, P. Cornwell, S. Doebling and M. Prime, "Structural health monitoring studies of the Alamosa Canyon and I-40 bridges," No. LA-13635-MS, Los Alamos National Lab., NM (US), 2000.

[40] K. Wong, C. Lau and A. Flint, "Planning and implementation of the structural health monitoring system for cable-supported bridges in Hong Kong," in *Nondestructive Evaluation of Highways, Utilities, and Pipelines*, 2000.

[41] H. Li, J. Ou, X. Zhao, W. Zhou, H. Li, Z. Zhou and Y. Yang, "Structural health monitoring system for the Shandong Binzhou Yellow River highway bridge," *Computer-Aided Civil and Infrastructure Engineering,* vol. 21, no. 4, pp. 306-317, 2006.

[42] M. Fraser, A. Elgamal, X. He and J. Conte, "Sensor network for structural health monitoring of a highway bridge," *Journal of Computing in Civil Engineering,* vol. 24, no. 1, pp. 11-24, 2009.

[43] K. Y. Koo, N. Battista, D. and J. Brownjohn, "SHM data management system using MySQL database with MATLAB and web interfaces," in *5th International Conference on Structural Health Monitoring of Intelligent Infrastructure (SHMII-5)*, Cancún, 2011.

[44] K. Smarsly, K. Law and D. Hartmann, "Multiagent-based collaborative framework for a self-managing structural health monitoring system," *Journal of Computing in Civil Engineering,* vol. 26, no. 1, pp. 76-89, 2012.

[45] M. Stonebraker, S. Madden, D. Abadi, S. Harizopoulos, N. Hachem and P. Helland, "The end of an architectural era (it's time for a complete rewrite)," in *The 33rd International Conference on Very Large Data Bases*, 2007.

[46] R. Hecht and S. Jablonski, "NoSQL evaluation: A use case oriented survey," in *2011 International Conference on Cloud and Service Computing (CSC)*, 2011.

[47] J. Han, E. Haihong, G. Le and J. Du, "Survey on NoSQL database," in *The 6th International Conference on Pervasive Computing and Applications*, 2011.

[48] R. Padhy, M. Patra and S. Satapathy, "RDBMS to NoSQL: Reviewing some next-generation non-relational databases," *International Journal of Advanced Engineering Science and Technologies,* vol. 11, no. 1, pp. 15-30, 2011.

[49] R. Nagel, W. Braithwaite and P. Kennicott, "Initial graphics exchange specification IGES version 1.0," National Bureau of Standards, Washington DC, 1980.

[50] ISO, "ISO 10303:1994 – Industrial automation systems and integration – Product data representation and exchange," International Organization for Standardization (ISO), Geneva, Switzerland, 1994.

[51] buildingSMART, "Industry foundation classes version 4 - Addendum 1," [Online]. Available: http://www.buildingsmart-tech.org/ifc/IFC4/Add1/html/. [Accessed 20 May 2016].

[52] V. Samec, J. Stamper, H. Sorsky and T. W. Gilmore, "Long span suspension bridges – bridge information modeling," in *7th International Conference of Bridge Maintenance, Safety and Management*, 2014.

[53] M. Marzouk and M. Hisham, "Bridge information modeling in sustainable bridge management," in *International Conference on Sustainable Design and Construction 2011: Integrating Sustainability Practices in the Construction Industry*, 2011.

[54] N. Yabuki, E. Lebegue, J. Gual and T. Shitani, "International collaboration for developing the bridge product model IFC-Bridge," in *11th International Conference on Computing in Civil and Building Engineering*, 2006.

[55] FHWA, "Open BrIM standards," [Online]. Available: https://collaboration.fhwa.dot.gov/dot/fhwa/ascbt/brim/default.aspx. [Accessed 23 April 2018].

[56] P. Mell and T. Grance, The NIST definition of cloud computing, 2011.

[57] Q. Zhang, L. Cheng and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications,* vol. 1, no. 1, pp. 7-18, 2010.

[58] T. Gillis, "Cost wars: Data center vs. public cloud," 2015. [Online]. Available: https://www.forbes.com/sites/tomgillis/2015/09/02/cost-wars-data-center-vs-public-cloud/#60c91b11923f. [Accessed December 5 2017].

[59] G. Deckler, "Cloud vs. on-Premises – Hard dollar costs," 2016. [Online]. Available: https://www.linkedin.com/pulse/cloud-vs-on-premises-hard-dollar-costs-greg-deckler/?trk=pulse_spock-articles. [Accessed 5 December 2017].

[60] T. Le, S. Kim, M. Nguyen, D. Kim, S. Shin, K. E. Lee and R. da Rosa Righi, "EPC information services with No-SQL datastore for the Internet of Things," in *2014 IEEE International Conference on RFID (IEEE RFID)*, 2014.

[61] T. Li, Y. Liu, Y. Tian, S. Shen and W. Mao, "A storage solution for massive IoT data based on NoSQL," in *2012 IEEE International Conference on Green Computing and Communications*, 2012.

[62] T. Thantriwatte and C. Keppetiyagama, "NoSQL query processing system for wireless ad-hoc and sensor networks," in *International Conference on Advances in ICT for Emerging Regions*, 2011.

[63] K. Law, J. Cheng, R. Fruchter and R. Sriram, "Engineering applications of the cloud," in *Encyclopedia of Cloud Computing*, S. Murugesan and I. Bojanova, Eds., Chichester, John Wiley & Sons, Ltd, 2016.

[64] C. Cheng, *SC Collaborator: A service oriented framework for construction supply chain collaboration and monitoring*, Ph.D. Thesis, Stanford University, 2009.

[65] C. Pautasso, O. Zimmermann and F. Leymann, "Restful web services vs. “Big” web services: making the right architectural decision," in *The 17th International Conference on World Wide Web*, Beijing, China, 2008.

[66] R. Fielding, *Architectural styles and the design of network-based software architectures*, Ph.D. Thesis, University of California, Irvine., 2000.

[67] H. Zhao and P. Doshi, "Towards automated restful web service composition," in *2009 IEEE International Conference on Web Services*, Los Angeles, CA, 2009.

[68] G. Mulligan and D. Gracanin, "A comparison of SOAP and REST implementations of a service based interaction independence middleware framework," in *2009 Winter Simulation Conference (WSC)*, Austin, TX, 2009.

[69] F. Belqasmi, J. Singh, S. Melhem and R. Glitho, "Soap-based vs. restful web services: A case study for multimedia conferencing," *IEEE Internet Computing,* vol. 16, no. 4, pp. 54-63, 2012.

[70] S. Jeong, R. Hou, J. Lynch, H. Sohn and K. Law, "An information modeling framework for bridge monitoring," *Advances in Engineering Software,* vol. 114, pp. 11-31, 2017.

[71] S. Jeong, Y. Zhang, S. O'Connor, J. Lynch, H. Sohn and K. Law, "A NoSQL data management infrastructure for bridge monitoring," *Smart Structures and Systems,* vol. 17, no. 4, pp. 669-690, 2016.

[72] S. Jeong, J. Byun, D. Kim, H. Sohn, I. H. Bae and K. H. Law, "A data management infrastructure for bridge monitoring," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, San Diego, CA, 2015.

[73] S. Jeong, R. Hou, J. Lynch, H. Sohn and K. Law, "A scalable cloud-based cyberinfrastructure platform for bridge monitoring," *Structure and Infrastructure Engineering,* vol. 15, no. 1, pp. 82-102, 2019.

[74] S. Jeong and K. Law, "An IoT platform for civil infrastructure monitoring," in *The 42nd IEEE Computer Society Signature Conference on Computers, Software & Applications (COMPSAC 2018)*, Tokyo, 2018.

[75] S. Jeong, Y. Zhang, R. Hou, J. Lynch, H. Sohn and K. Law, "A cloud-based information repository for bridge monitoring applications," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, Las Vegas, NV, 2016.

[76] S. Jeong, R. Hou, J. Lynch, H. Sohn and K. Law, "A distributed cloud-based cyberinfrastructure framework for integrated bridge monitoring," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, Portland, OR, 2017.

[77] S. Jeong, M. Ferguson and K. Law, "Sensor data reconstruction and anomaly detection using bidirectional recurrent neural network," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, Denver, CO, 2019.

[78] M. Bartholomew, B. Blasen and A. Koc, "Bridge information modeling (BrIM) using open parametric objects," No. FHWA-HI F -16-010, Federal Highway Administration, 2015.

[79] Computers & Structures, Inc. , "Structural bridge design software | CSiBridge," [Online]. Available: https://www.csiamerica.com/products/csibridge. [Accessed 20 April 2016].

[80] Open Geospatial Consortium, "OGC® SensorML: Model and XML encoding standard," 2014. [Online]. Available: http://www.opengeospatial.org/standards/sensorml. [Accessed 20 April 2016].

[81] OpenBrIM, "OpenBrIM V3," [Online]. Available: https://openbrim.appspot.com/schema.xsd?for=openbrim&v=3&format=xsd&version=1.1. [Accessed 10 November 2016].

[82] ParamML, "ParamML author's guide," [Online]. Available: https://sites.google.com/a/redeqn.com/paramml-author-s-guide/home. [Accessed 20 April 2016].

[83] W3C, "XML schema," 15 October 2014. [Online]. Available: http://www.w3.org/2001/XMLSchema. [Accessed 23 April 2018].

[84] W3Schools, "XML schema reference," [Online]. Available: https://www.w3schools.com/xml/schema_elements_ref.asp. [Accessed 23 April 2018].

[85] "Liquid Studio," Liquid Technologies, [Online]. Available: https://www.liquid-technologies.com/xml-studio. [Accessed 13 January 2017].

[86] "XML Editor: XML Spy," Altova, [Online]. Available: https://www.altova.com/xmlspy.html. [Accessed 13 January 2017].

[87] ISO, "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, 2000.

[88] K. Lee, "IEEE 1451: A standard in support of smart transducer networking," in *IEEE Instrumentation and Measurement Technology Conference*, 2000.

[89] J. Pschorr, C. Henson, H. Patni and A. Sheth, "Sensor discovery on linked data," 2010. [Online]. Available: http://corescholar.libraries.wright.edu/knoesis/780. [Accessed 2 March 2016].

[90] S. O'Connor, J. Lynch, M. Ettouney, G. vander Linden and S. Alampalli, "Cyber-enabled decision making system for bridge management using wireless monitoring systems: Telegraph Road Bridge demonstration project," in *Structural Materials Technology 2012*, 2012.

[91] S. O'Connor, Y. Zhang, J. Lynch, M. Ettouney and P. Jansson, "Long-term performance assessment of the Telegraph Road Bridge using a permanent wireless monitoring system and automated statistical process control analytics," *Structure and Infrastructure Engineering,* vol. 13, no. 5, pp. 604-624, 2017.

[92] H. Koh, W. Park and H. Kim, "Recent activities on operational monitoring of long-span bridges in Korea," in *The 6th International Conference on Structural Health Monitoring of Intelligent Infrastructure*, 2013.

[93] S. O'Connor, Y. Zhang, J. Lynch, M. Ettouney and G. van der Linden, "Automated analysis of long-term bridge behavior and health using a cyber-enabled wireless monitoring system," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, San Diego, CA, 2014.

[94] H. Sohn, H. Lim and S. Yang, "A fatigue crack detection methodology," in *Smart Sensors for Health and Environment Monitoring*, Dordrecht, Springer, pp. 233-253, 2015.

[95] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing*, 2013.

[96] A. Moniruzzaman and H. S.A., "NoSQL database: New era of databases for big data analytics - classification, characteristics and comparison," *International Journal of Database Theory and Application,* vol. 6, no. 4, pp. 1-14, 2013.

[97] D. McNeill, "Data management and signal processing for structural health monitoring of civil infrastructure systems," in *Structural Health Monitoring of Civil Infrastructure Systems*, V. Karbhari and F. Ansari, Eds., Boca Raton, FL: CRC Press, pp. 283-304, 2009.

[98] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes and R. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS),* vol. 26, no. 2, pp. 4, 2008.

[99] The Apache Software Foundation, "Apache Cassandra," [Online]. Available: http://cassandra.apache.org/. [Accessed 1 February 2018].

[100] MongoDB, Inc., "MongoDB for giant ideas | MongoDB," [Online]. Available: http://www.mongodb.com/. [Accessed 28 May 2018].

[101] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *Operating Systems Review (ACM),* vol. 44, no. 2, pp. 35-40, 2010.

[102] K. Chodorow, MongoDB: The definitive guide: Powerful and scalable data storage., O'Reilly Media, Inc., 2013.

[103] D. Hows, P. Membrey and E. Plugge, MongoDB basics, Berkely, CA: Apress, 2014.

[104] DataStax, "Introduction to Cassandra Query Language," [Online]. Available: http://docs.datastax.com/en/cql/3.1/cql/cql_intro_c.html. [Accessed 20 April 2016].

[105] DataStax, "Python Cassandra Driver," [Online]. Available: https://datastax.github.io/python-driver/. [Accessed 20 April 2016].

[106] "Python Imaging Library (PIL)," [Online]. Available: http://www.pythonware.com/products/pil/. [Accessed 7 December 2017].

[107] E. Hewitt, Cassandra: the definitive guide, O'Reilly Media, Inc., 2010.

[108] Python Software Foundation, "19.7. xml.etree.ElementTree — The ElementTree XML API," [Online]. Available:

https://docs.python.org/2/library/xml.etree.elementtree.html. [Accessed 1 February 2018].

[109] MongoDB, "PyMongo 3.6.1 documentation," [Online]. Available: https://api.mongodb.com/python/current/. [Accessed 1 June 2018].

[110] MathWorks, Inc., "MATLAB API for Python," [Online]. Available: https://www.mathworks.com/help/matlab/matlab-engine-for-python.html. [Accessed 24 June 2018].

[111] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg and J. Vanderplas, "Scikit-learn: Machine learning in Python," *The Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[112] L. Gautier, "Documentation for rpy2," [Online]. Available: https://rpy2.readthedocs.io/en/version_2.8.x/. [Accessed 24 June 2018].

[113] S. O'Connor, Y. Zhang and J. Lynch, "Automated outlier detection framework for identifying damage states in multi-girder steel bridges using long-term wireless monitoring data," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, 2015.

[114] State of Michigan, "MDOT - Mi Drive interactive map," [Online]. Available: https://mdotnetpublic.state.mi.us/drive/. [Accessed 1 February 2018].

[115] R. Hou, Y. Zhang, S. O’Connor, Y. Hong and J. P. Lynch, "Monitoring and identification of vehicle-bridge interaction using mobile truck-based wireless sensors," in *11th International Workshop on Advanced Smart Materials and Smart Structures Technology*, Urbana-Champaign, IL, 2015.

[116] G. E. and C. C., "openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files," 2016. [Online]. Available: http://openpyxl.readthedocs.org/en/default/ . [Accessed 20 April 2016].

[117] Python Software Foundation, "pywin32 214 : Python package index," [Online]. Available: https://pypi.python.org/pypi/pywin32. [Accessed 7 October 2016].

[118] Python Software Foundation, "7.5. StringIO — Read and write strings as files," [Online]. Available: https://docs.python.org/2/library/stringio.html. [Accessed 7 October 2016].

[119] R. Baheti and H. Gill, "Cyber-physical systems," *The Impact of Control Technology,* vol. 12, pp. 161-166, 2011.

[120] Node.js Foundation, "Node.js," [Online]. Available: https://nodejs.org/. [Accessed 1 February 2018].

[121] H. Haas and A. Brown, "Web services glossary," 2004. [Online]. Available: https://www.w3.org/TR/ws-gloss/. [Accessed 30 January 2017].

[122] D. Guinard, V. Trifa and E. Wilde, "A resource oriented architecture for the Web of Things," in *2010 Internet of Things (IoT)*, Tokyo, 2010.

[123] Q. Z. Sheng, X. Qiao, A. Vasilakos, C. Szabo, S. Bourne and X. Xu, "Web services composition: A decade's overview," *Information Sciences,* vol. 280, pp. 218-238, 2014.

[124] C. Pautasso, "BPEL for REST," in *International Conference on Business Process Management*, 2008.

[125] F. Rosenberg, F. Curbera, M. Duftler and R. Khalaf, "Composing RESTful services and collaborative workflows: A lightweight approach," *IEEE Internet Computing,* vol. 12, no. 5, pp. 24–31, 2008.

[126] C. Pautasso, "Composing RESTful services with JOpera," in *International Conference on Software*, Zurich, Switzerland, 2009.

[127] "JOpera for Eclipse," JOpera.org, 2013. [Online]. Available: http://www.jopera.org/. [Accessed 7 December 2017].

[128] C. Pautasso and G. Alonso, "The JOpera visual composition language," *Journal of Visual Languages and Computing,* vol. 16, no. 1-2, pp. 119–152, 2005.

[129] "Maps JavaScript API," Google Developers, [Online]. Available: https://developers.google.com/maps/documentation/javascript/. [Accessed 7 December 2017].

[130] J. Smith and R. Nair, "The architecture of virtual machines," *Computer,* vol. 38, no. 5, pp. 32-38, 2005.

[131] "Microsoft Azure," Microsoft, [Online]. Available: https://azure.microsoft.com/. [Accessed 7 December 2017].

[132] X. Huang and X. Du, "Efficiently secure data privacy on hybrid cloud," in *2013 IEEE International Conference on Communications (ICC)*, Budapest, 2013.

[133] T. Dillon, C. Wu and E. Chang, "Cloud computing: issues and challenges," in *24th IEEE International Conference on Advanced Information Networking and Applications (AINA)*, 2010.

[134] P. Overschee, "Subspace Identification for Linear Systems," 2002. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/2290-subspace-identification-for-linear-systems. [Accessed 27 June 2017].

[135] S. Qin and W. Li, "Detection, identification, and reconstruction of faulty sensors with maximized sensitivity," *AIChE Journal,* vol. 45, no. 9, pp. 1963-1976, 1999.

[136] S. Wang and Y. Chen, "Sensor validation and reconstruction for building central chilling systems based on principal component analysis," *Energy Conversion and Management,* vol. 45, no. 5, pp. 673-695, 2004.

[137] G. Kerschen, P. De Boe, J. Golinval and K. Worden, "Sensor validation using principal component analysis," *Smart Materials and Structures,* vol. 14, no. 1, pp. 36-42, 2005.

[138] J. Kullaa, "Sensor validation using minimum mean square error estimation," *Mechanical Systems and Signal Processing,* vol. 24, no. 5, pp. 1444-1457, 2010.

[139] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Transactions on Industrial Informatics,* vol. 9, no. 4, pp. 2226–2238, 2013.

[140] X. Xu, J. Hines and R. Uhrig, "Sensor validation and fault detection using neural networks," in *Maintenance and Reliability Conference (MARCON 99)*, 1999.

[141] I. Eski, S. Erkaya, S. Savas and S. Yildirim, "Fault detection on robot manipulators using artificial neural networks," *Robotics and Computer-Integrated Manufacturing,* vol. 27, no. 1, pp. 115–123, 2011.

[142] K. Smarsly and K. Law, "Decentralized fault detection and isolation in wireless structural health monitoring systems using analytical redundancy," *Advances in Engineering Software,* vol. 73, pp. 1–10, 2014.

[143] K. Dragos and K. Smarsly, "Distributed adaptive diagnosis of sensor faults using structural response data," *Smart Materials and Structures,* vol. 25, no. 10, pp. 105019–15, 2016.

[144] K. Law, S. Jeong and M. Ferguson, "A data-driven approach for sensor data reconstruction for bridge monitoring," in *2017 World Congress on Advances in Structural Engineering and Mechanics*, 2017.

[145] J. Kullaa, "Distinguishing between sensor fault, structural damage, and environmental or operational effects in structural health monitoring," *Mechanical Systems and Signal Processing,* vol. 25, no. 8, pp. 2976–2989, 2011.

[146] A. I. Moustapha and R. R. Selmic, "Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection," *IEEE Transactions on Instrumentation and Measurement,* vol. 57, no. 5, pp. 981–988, 2008.

[147] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing,* vol. 45, no. 11, pp. 2673-2681, 1997.

[148] Y. LeCun, L. Bottou, G. Orr and K. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*, Berlin, Heidelberg., Springer, pp. 9-50, 1998.

[149] A. Ng, J. Ngiam, C. Foo, Y. Mai, C. Suen, A. Coates, A. Maas, A. Hannun, B. Huval, T. Wang and S. Tandon, "UFLDL tutorial," [Online]. Available: http://deeplearning.stanford.edu/tutorial/. [Accessed 15 January 2018].

[150] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *The Thirteenth International Conference on Artificial Intelligence and Statistics*, 2010.

[151] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE,* vol. 78, no. 10, pp. 1550-1560, 1990.

[152] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *The 3rd International Conference for Learning Representations*, 2015.

[153] M. Hagan, H. Demuth, M. Beale and O. De Jess, Neural network design, 2014.

[154] R. Williams and J. Peng, "An efficient gradient-based algorithm for on-line training of recurrent network trajectories," *Neural Computation,* vol. 2, no. 4, pp. 490-501, 1990.

[155] I. Sutskever, *Training recurrent neural networks.*, Ph.D. Thesis, Department of Computer Science, University of Toronto, 2013.

[156] Y. Zhang, S. O'Connor, G. van der Linden, A. Prakash and J. Lynch, "SenStore: a scalable cyberinfrastructure platform for implementation of data-to-decision frameworks for infrastructure health management," *Journal of Computing in Civil Engineering,* vol. 30, no. 5, pp. 04016012, 2016.

[157] S. O'Connor, Y. Zhang, J. Lynch, M. Ettouney and P. O. Jansson, "Long-term performance assessment of the Telegraph Road Bridge using a permanent wireless monitoring system and automated statistical process control analytics," *Structure and Infrastructure Engineering,,* vol. 13, no. 5, pp. 604-624., 2017.

[158] American Association of State Highway and Transpor, Standard specifications for highway bridges, 17th ed., Washington D.C.: American Association of State Highway and Transportation Officials, 2002.

[159] PyTorch, "PyTorch," [Online]. Available: http://pytorch.org/. [Accessed 1 February 2018].

[160] A. Pouliezos and G. Stavrakakis, "Analytical redundancy methods," in *Real Time Fault Monitoring of Industrial Processes*, Dordrecht, Springer, 1994.

[161] A. Trunov and M. Polycarpou, "Automated fault diagnosis in nonlinear multivariable systems using a learning methodology," *IEEE Transactions on Neural Networks,* vol. 11, no. 1, pp. 91–101, 2000.

[162] H. Sohn, "Effects of environmental and operational variability on structural health monitoring," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences,* vol. 365, no. 1851, pp. 539-560, 2007.

[163] R. Brincker, L. Zhang and P. Andersen, "Modal identification of output-only systems using frequency domain decomposition," *Smart Materials and Structures,* vol. 10, no. 3, pp. 441-445, 2001.

[164] A. Mosavi, H. Sedarat, S. O'Connor, A. Emami-Naeini, V. Jacob, A. Krimotat and J. Lynch, "Finite element model updating of a skewed highway bridge using a multi-variable sensitivity-based optimization approach," in *SPIE Smart Structures and Materials + Nondestructive Evaluation and Health Monitoring*, San Diego, CA, 2012.