20th International Conference on the Application of Computer
Science and Mathematics in Architecture and Civil Engineering
K. Gürlebeck and T. Lahmer (eds.)
Weimar, Germany, 20–22 July 2015

A GENERIC FRAMEWORK SUPPORTING DISTRIBUTED COMPUTING IN ENGINEERING APPLICATIONS

J.Wiggenbrock*[1,2] and K. Smarsly[1]

[1]*Bauhaus University Weimar*
Coudraystr. 7, 99423 Weimar, Germany
Email: jens.wiggenbrock@uni-weimar.de

[2]*South Westphalia University of Applied Sciences*
Lindenstraße 53, 59872 Meschede, Germany

Keywords: Data Modeling, Distributed Engineering Applications, Message-oriented Middleware, Computing in Civil Engineering.

Abstract. *Modern distributed engineering applications are based on complex systems consisting of various subsystems that are connected through the Internet. Communication and collaboration within an entire system requires reliable and efficient data exchange between the subsystems. Middleware developed within the web evolution during the past years provides reliable and efficient data exchange for web applications, which can be adopted for solving the data exchange problems in distributed engineering applications. This paper presents a generic approach for reliable and efficient data exchange between engineering devices using existing middleware known from web applications. Different existing middleware is examined with respect to the suitability in engineering applications. In this paper, a suitable middleware is shown and a prototype implementation simulating distributed wind farm control is presented and validated using several performance measurements.*

1 INTRODUCTION

A general trend in engineering applications are interconnected subsystems that in total form a distributed engineering system. Every subsystem may include sensors and actuators supporting the overall task of the engineering system. A common general requirement of distributed engineering systems is the need for reliable and efficient data exchange between the interconnected subsystems. Examples of such engineering systems are industrial assembly lines, home automation, wired or wireless structural health monitoring [1] [2] or, more specifically, wind farms being composed of single wind turbines representing interconnected subsystems.

There exists a number of proprietary and enterprise-specific approaches for reliable and efficient data exchange between interconnected and spatially distributed subsystems. The approaches provide data exchange protocols, network topologies and, also, several specifications for additional cabling that must be considered when implemented in terms of an engineering system required to connect the subsystems. Disadvantages of these approaches are high expenses because of the additional cabling. Furthermore, the engineering system is technology-dependent and, as such, hardly expandable.

Modern, collaborative web applications, such as groupware and social networks, also require reliable and efficient data exchange between the subsystems. Here, the web servers and the web browsers are considered subsystems forming the entire system "web" that is connected through the Internet. In the area of web applications, there exists well-established middleware for data exchange between the subsystems (i.e. web servers and web browsers). This middleware can advantageously be adopted to implement reliable and efficient data exchange in distributed engineering applications.

In this paper, a generic framework for reliable and efficient data exchange between engineering devices is presented, using existing middleware usually deployed in web applications. Starting with a brief overview of modern collaborative web applications, the principles of existing middleware are elucidated. Then, selection criteria for suitable middleware with respect to data exchange in engineering applications are defined, and existing middleware is examined. Based on the selection results, the middleware being most appropriate to be adopted to engineering applications is taken as a basis for a prototype implementation. Finally, the prototype implementation is validated using several performance measurements.

2 MESSAGE-ORIENTED MIDDLEWARE FOR ENGINEERING APPLICATIONS

"Message-oriented middleware" is a generic term describing a software that operates as message exchange service. The software can be divided into a server, which provides the exchange service, and participating clients, which require message exchange between each other. A message refers to all types of data and command packets being exchanged. The following paragraphs describe state-of-the-art message-oriented middleware for engineering applications.

2.1 Principles of message-oriented middleware

Today, most devices that are connected to the Internet are located behind a router with an integrated firewall. As shown in Figure 1, the router grants Internet access to the devices behind the router. Vice versa, the devices behind the router cannot be accessed from the

Internet directly. To give an example, the mobile phone shown in Figure 1 cannot establish a connection to the public web server, and the web server cannot establish a connection to any device, such as the mobile phone mentioned above.

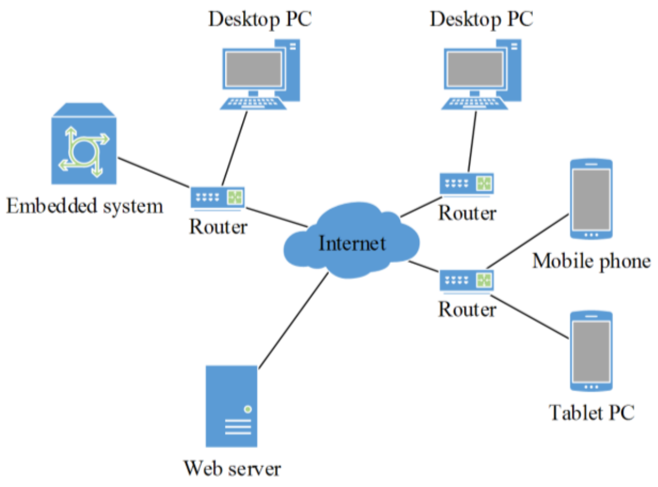

Figure 1: Topology of various devices connected through the Internet

In summary, the topology shown in Figure 1 shields the devices behind the router and basically protects the devices from external access. However, considering engineering applications, establishing a connection to a device behind a router can become a serious issue if the connection is needed for the engineering application.

A well-known solution for reliable and efficient data exchange between devices connected through the Internet is the establishment of a public server, based e.g. on the File Transfer Protocol (FTP). The common, unidirectional and indirect way to transfer a file between the devices via FTP is shown in Figure 2. The desktop PC connects with the public FTP server and uploads a file (green arrows). Once the file is uploaded, the mobile phone connects with the public FTP server and downloads the file (red arrows). Technically, the FTP server operates as a middleware for file exchange.

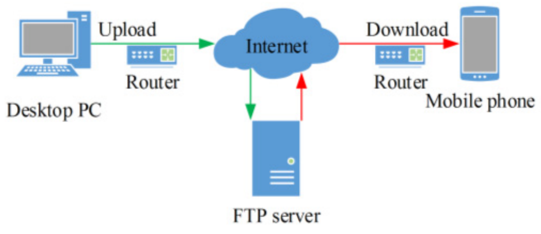

Figure 2: Indirect, unidirectional data exchange over a public FTP server

Both operations, upload and download, are executed independently from each other and must explicitly be initiated by the respective client device, i.e. the upload process must be finished before the download process is started. Unfortunately, traditional file exchange protocols, such as FTP, generally do not provide any possibility to notify participating clients when the upload process is finished or when changes at the server occur. Thus, additional means for notifying participating clients, such as email or phone calls, are needed, which, however, leads to delays in the data exchange process.

Eradicating the drawbacks illustrated above, message-oriented middleware (MOM) enables data exchange between various devices based on a different concept. Negotiating on a central server allows near real-time data exchange between participating clients, as shown in Figure 3 and in Figure 4 [3]. Message-oriented middleware provides different communication protocols, such as “message passing”, “message queueing”, and “publish/subscribe”. While “message passing” and “message queueing” are mostly used for concurrent programming in local applications, publish/subscribe is a well-established model for asynchronous distributed computing considered herein.

A public server provides topic-oriented communication channels and takes on their central mediation. First, all participating clients connect to the public server. Then, the clients register with the server and “subscribe” to a specific topic. Finally, the server compiles a list of all clients and topics, and it manages the communication. To receive data, a client sends a message with the topic of interest to the server. The server processes the list of clients and topics, and it forwards the message to the clients subscribed to the topic of interest. Figure 3 and Figure 4 show different publish/subscribe architectures. Figure 3 presents a simple architecture of one client publishing in one topic, a server, and two subscriber clients. Figure 4 presents a simple architecture of two clients publishing in the same or in different topics, a server, and one subscriber client.

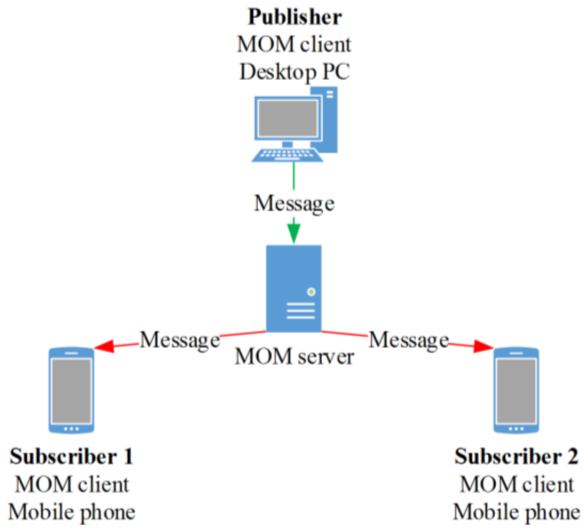

Figure 3: Publish/subscribe architecture with one publisher and two subscribers

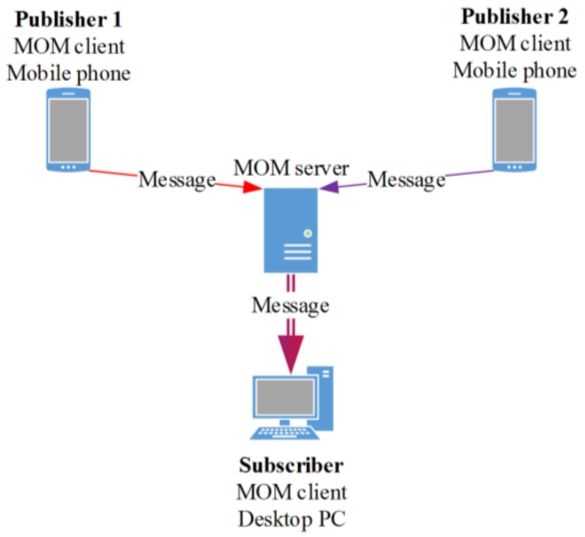

Figure 4: Publish/subscribe architecture with two publishers and one subscriber

2.2 Selection criteria for message-oriented middleware for engineering applications

Automated, distributed engineering applications are widely used in a number of areas. The most critical component in distributed engineering applications is the communication system [4]. Essential features, which every distributed engineering application must provide, are defined as follows:

- Fast data exchange: Data packets have to be delivered with low and predictable latency
- Reliable data exchange: Data must be equipped with error-correcting code
- Durability: The lifetime of engineering applications may be several decades; the engineering systems have to be maintained over the lifetime

When using message-oriented middleware in engineering applications, specific requirements must be met. The following criteria are identified for selecting suitable message-oriented middleware for distributed engineering applications. Based on these criteria, Table 1 summarizes the middleware examined in this study.

- Hosting: Does the middleware provide self-hosting on an independent server?
- Data security: Does the middleware provide authentication and encryption or is a plugin available?
- Future security: Is the development within the next years ensured by a vivid community?
- Usability: Does the middleware provide a programming interface, which can easily be integrated into existing engineering applications?

Table 1: Middleware examined in this study

Middleware	**Host**	**Data security**	**Future security**	**Usability**	**Remarks**
Redis http://redis.io/	Self-host	**Not integrated**	Server and clients are open source	Server: Independent application on Linux/Windows Clients: Supporting many programming languages	Popular NoSQL-database with publish/subscribe function
Google Cloud pub/sub API https://cloud.google.com/pubsub/docs	**Cloud-host**	HTTPS	**Beta version, potential for future Google services**	Server: Managed service Clients: Supporting .NET, Java and JavaScript	First (beta) release in 03/2015
Apache Kafka http://kafka.apache.org/	Self-host	SSL implemented in last version	Server and clients are open source	Server: Independent application on Linux/Windows Clients: Supporting several progr. languages	Originally developed by LinkedIn
PUSHER https://pusher.com/	**Cloud-host**	SSL implemented	**Server not available, Clients are open source**	Server: Managed service Clients: Supporting .NET, Java and JavaScript	Describes itself as “Leader in realtime technologies”
Socket.io + NODE.js http://socket.io/	Self-host	SSL implemented	**Server and clients are open source**	**Server and client only available in JavaScript**	**Popular event-driven JavaScript server architecture**
RabbitMQ http://www.rabbitmq.com	Self-host	SLL implemented but depends on Erlang crypto application	Server and clients are open source.	Server: Independent application on Linux/Windows written in Erlang Clients: Supporting several progr. languages	Implements the open Advanced Message Queuing Protocol enabling own client developments
ASP.NET SignalR http://signalr.net	Self-host	Over IIS-Server / OWIN	Open source	Server: Integrates in existing C# or ASP.NET appl’s on Linux/Windows Clients: Supporting C#, Java, JavaScript	Started as open source project; now core feature of ASP.NET

As can be seen from Table 1, “Apache Kafka”, “RabbitMQ” and “SignalR” are matching all defined selection criteria. In addition, SignalR provides a so called server module that can be integrated directly into existing engineering applications [5]; this enables the server to perform further processing on the messages and to distribute the messages according to additional filter rules. Finally, SignalR is chosen in this study as an appropriate basis for the prototype implementation simulating decentralized wind farm control.

3 A GENERIC FRAMEWORK SUPPORTING REAL-TIME DATA EXCHANGE FOR DECENTRALIZED WIND FARM CONTROL

Representing an illustrative example of a distributed engineering application, decentralized collaborative control of a wind farm is chosen as a proof of concept of the proposed framework. Today, wind turbines in a wind farm are usually operated without considering wake effects between the wind turbines. Figure 5 shows a wind farm with highlighted wake fields affecting wind turbines lying behind other wind turbines. Minimizing the wake effects increases the wind farm power efficiency. In recent years, different approaches towards wind farm power efficiency optimization have been proposed. Park et al., for example, propose a cooperative control strategy, adjusting the yaw control of the nacelle, the pitch control of the rotor blades and the induction factor of the generator to alter the wake field of each wind turbine [6, 7]. It is evident that automated real-time control of the wind turbines in a collaborative way can substantially increase the overall performance of a wind farm in terms of power efficiency. In this paper, collaborative control of a wind farm is simulated, serving as a proof of concept of the proposed framework.

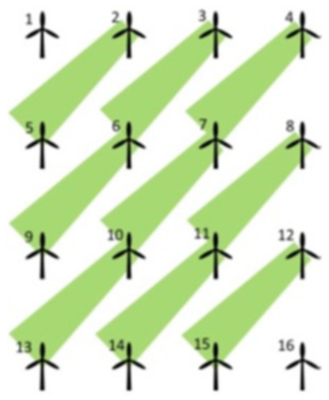

Figure 5: Wind farm with highlighted wake fields affecting wind turbines lying behind other wind turbines (figure source: [4])

It is assumed that for cooperative control, the location (i.e. latitude and longitude) of each wind turbine within the wind farm, the actual settings (i.e. yaw angle, rotor blade pitch, and induction factor), the power output, and the environmental data (i.e. wind direction and wind speed) of each wind turbine is needed to calculate the optimum settings for each wind turbine relevant to collaborative wind farm power maximization. The optimum settings of each wind turbine contain improved yaw angle, rotor blade pitch, and induction factor. It is further assumed that real-time computational optimization is done by a central control unit within a procedure carried out in regular intervals. Each wind turbine sends the described data sets to the central control unit. In this study, the central control unit runs an engineering application simulating the wind farm optimization model proposed by Park et al. The calculated data sets containing the optimum settings are sent to the respective wind turbine in through messages.

Assuming that the messages, which contain the data sets described above, are sent in human readable text format (JSON or XML), one message has a size of less than 100 bytes (about 10 bytes per value). In this study, data exchange between the central control unit and the wind turbines 10 times per minute will be sufficient for wind farm control.

The SignalR middleware used in this study has originally been developed to integrate real-time communication in ASP.NET web applications [8]. A SignalR server is based on the programming language C# and provides server and client integration in underlying applications running on Windows and Linux. Clients for Java and JavaScript are also available. The ability of SSL encryption is a further feature of SignalR. Figure 6 presents the architecture of the proposed generic framework, which basically consists of one server that represents the central control unit, and two clients that represent two wind turbines. Both, server and clients, include specific SignalR libraries that provide a hub class. The hub class is the interface to handle all SignalR communications within the distributed engineering application. Furthermore, the hub class provides a programmable interface for external access, which keeps the main part of the application encapsulated. Data exchange between subsystems is done by invoking hub class methods that pass the data sets of the wind turbines as method parameters.

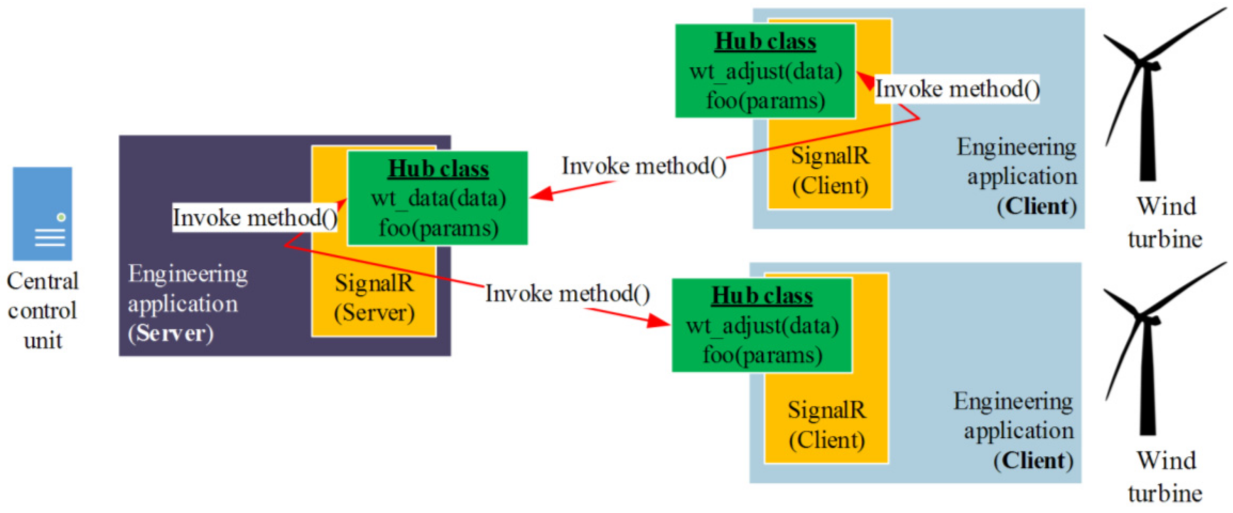

Figure 6: SignalR-based example application for wind farm control

As mentioned earlier, the wind turbines technically act as the clients and the central control unit acts as the server. Figure 7 shows the wind farm control as a closed control circuit. The clients, in pre-defined intervals, collect the environmental data and the wind turbine settings required for real-time optimization. Once having collected all data sets, the clients invoke the server method “wt_data”, passing the collected data sets to the server.

As shown in Figure 7, the “wt_data” server method passes the collected data sets to the server engineering application, which then calculates the optimum settings for each wind turbine. After the calculations are done, the server engineering application invokes the client method “wt_adjust” on every client, passing the optimum settings for adjusting the respective wind turbine. Thus, the optimum settings are transmitted to the server. Finally, the “wt_adjust”-method passes the optimum settings to the wind turbine, which adopts the optimum settings to adjust the wind turbine actuators.

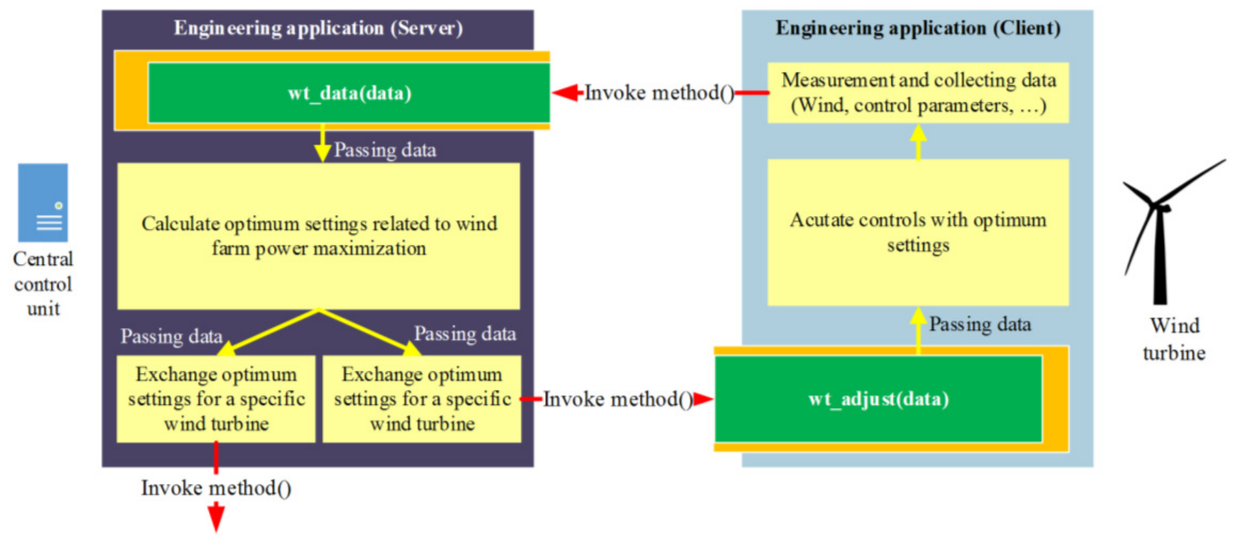
Engineering application (Server)
wt_data(data)
Invoke method()
Engineering application (Client)
Measurement and collecting data (Wind, control parameters, …)
Passing data
Central control unit
Calculate optimum settings related to wind farm power maximization
Acutate controls with optimum settings
Wind turbine
Passing data
Passing data
Passing data
Exchange optimum settings for a specific wind turbine
Exchange optimum settings for a specific wind turbine
Invoke method()
wt_adjust(data)
Invoke method()

Figure 7: Closed control circuit enabling wind farm optimization

The above described framework has been implemented into a prototype application simulating distributed wind farm control. The prototype application has been written in the programming language C#. As a result, the program is executable on different computer systems and operating systems, entailing an easy integration into existing systems, such as existing wind turbine control systems. Performance measurements have been conducted on the prototype application to validate the reliability and the efficiency of the data exchange. The test environment defined for the performance measurement is shown in Figure 7, measuring the message response times from client to server and back to the client. The time difference, also called latency, is an acknowledged performance indicator used to compare different system variations. Table 2 presents the performance measurement results for different system variations.

		Client		
		Desktop PC, Windows 7	**Desktop PC, Linux Ubuntu 14.10**	**Raspberry Pi 1, Debian Raspbian**
Server	**Desktop PC, Windows 7**	2 ms	3 ms	20 ms
	Desktop PC, Linux Ubuntu 14.10	270 ms	70 ms	90 ms
	Raspberry Pi 1, Debian Raspbian	730 ms	520 ms	550 ms

Table 2: Performance measurement results

Summarizing the performance measurement results, it can be concluded that the proposed framework is usable for reliable and efficient data exchange supporting distributed computing in engineering applications, specifically for decentralized wind farm control as simulated herein. However, the different response times on desktop PCs running Linux as server are unexpected and may open a field for further research.

4 SUMMARY AND CONCLUSIONS

This paper has presented a generic framework for reliable and efficient data exchange in engineering applications. Principles of existing middleware, well-established in the field of web applications, have been adopted to distributed engineering applications. Specifically, selection criteria for reliable and efficient data exchange in engineering applications have been defined and matched with existing middleware. Based on the selection criteria, SignalR has been examined in detail for a prototype implementation simulating distributed wind farm control. As a result, the prototype implementation has shown that SignalR is universally applicable to different engineering systems as a reliable and efficient middleware for data exchange. Performance measurements have demonstrated that the performance strongly depends on the roles of the subsystems (e.g. server or client) as well as on the computer systems and operating systems used (e.g. Windows and the Microsoft .NET Framework or Linux and the Mono .NET Framework).

5 ACKNOWLEDGEMENTS

The authors would like to gratefully acknowledge the generous support offered by Professor Stefan Breide and Professor Jürgen Willms (South Westphalia University of Applied Sciences, Germany).

REFERENCES

[1] K. Smarsly, K. Lehner and D. Hartmann, "Structural Health Monitoring based on Artificial Intelligence Techniques," in *Proceedings of the International Workshop on Computing in Civil Engineering*, Pittsburgh, PA, USA, 2007.

[2] K. Smarsly, K. H. Law and M. König, "Autonomous Structural Condition Monitoring based on Dynamic Code Migration and Cooperative Information Processing in Wireless Sensor Networks," in *Proceedings of the 8th International Workshop on Structural Health Monitoring 2011*, Stanford, CA, USA, 2011.

[3] M. Qusay and E. Curry, Middleware for Communications, New Jersey, USA: John Wiley & Sons, Ltd, 2004.

[4] K. H. Law, K. Smarsly and Y. Wang, "Sensor Data Management Technologies for Infrastructure Asset Management," in *Sensor Technologies for Civil Infrastructures*, Sawston, UK, Woodhead Publishing, Ltd., 2014, pp. 3-32.

[5] J. S. Lang and J. R. Irving, "Creating a Prototype Web Application for Spacecraft Real-Time Data Visualization on Mobile Devices," in *SpaceOps 2014 International Conference on Space Operations*, Pasadena,CA, USA, 2014.

[6] J. Park, S. Kwon and K. H. Law, "Wind Farm Power Maximization Based On A Cooperative Static Game Approach," in *Proceedings of the SPIE Smart Structures/NDE Conference*, San Diego, CA, USA, 2013.

[7] J. Park and K. H. Law, "A Bayesian optimization approach for wind farm power maximization," in *Proceedings of the SPIE Smart Structures/NDE Conference*, San Diego, CA, USA, 2015.

[8] G. A. Valdez, "SignalR: Building real time web applications," Microsoft, 17 12 2012. [Online]. Available: http://blogs.msdn.com/b/webdev/archive/2012/12/17/signalr-building-real-time-web-applications.aspx. [Accessed January 9, 2015].