A Software Architecture for Simulation and Visualisation based on the Functional Mock-up Interface and Web Technologies

Lars Ivar Hatledal[1] Hans Georg Schaathun[2] Houxiang Zhang[1]

[1]Department of Maritime Technology and Operations, Aalesund University College, Norway, {laht,hozh}@hials.no

[2]Department of Engineering and Natural Sciences, Aalesund University College, Norway, hasc@hials.no

Abstract

This paper presents a software architecture for a collaborative virtual environment (CVE) for simulation and visualisation based on the Functional Mock-up Interface (FMI) for co-simulation and web technologies. FMI has been chosen in order to have a standardised and independent interface to models created in different modelling tools.

The user interface has been implemented using web technologies, which enables a very high degree of flexibility. The Web Graphics Library (WebGL) is used for interactive 3D visualisations, enabling native cross-platform rendering directly in the browser without the need of installing any additional plug-ins. Employing the bi-directional communication capabilities of the WebSocket protocol, multiple users can interact with the same simulation models simultaneously.

A software prototype has been developed in order to demonstrate the applicability of the proposed architecture. As a case study, we have considered virtual prototyping of marine cranes, to illustrate the use on real world problems.

Keywords: Functional Mock-up Interface, WebGL, Virtual Prototyping, Web based Simulation

1 Introduction

Virtual prototyping is a hot topic in many industries. The construction of physical prototypes is costly and time consuming, but has traditionally been necessary to be able to test and evaluate new designs. As computer technology develops it becomes possible to make an increasing part of the necessary tests based on simulations. Modelling and simulation of components has been possible for some time, and good tools exist. Simulation of complex systems is harder, and in most cases it still depends on a costly and time consuming *ad hoc* integration of components. Virtual prototyping refers to a vision where models, or *virtual prototypes*, of complex systems can be developed, tested, and amended with a trial-and-error approach.

As the standards of the web has matured, a growing number of applications have been made accessible from a web browser. The latest version of the HTML standard, HTML5, has brought along new powerful developer Application Programming Interfaces (APIs), previously restricted to desktop applications. WebGL (Marrin, 2011), has made it possible to utilise the 3D rendering capabilities of the GPU from within the browser, without the use of plug-ins. Furthermore, the WebSocket standard (Fette and Melnikov, 2011) allows for low latency bi-directional communication between browsers and web servers.

FMI (Blochwitz et al., 2012) is an open and tool independent standard for model exchange and co-simulation of dynamic models. FMI is currently supported by 73 tools, which indicates a major impact on research and industry. In this architecture, FMI has been chosen in order to have a standardised and independent interface to simulation models created in different modelling tools. These FMI compliant models, called Functional Mock-up Units (FMUs), are distributed on the network and accessed using Remote Procedure Calls (RPC) in order to ensure scalability as the number of concurrent simulations increases. Modules that describes sub-systems (either in the form of FMUs or directly implemented in Java) can be assembled together in order to form more complete systems.

The user interface has been implemented using web technologies, which enables a very high degree of flexibility and low coupling. For instance, modifications to the underlying architecture can be applied without issuing software updates to clients. Furthermore, users can view and interact with live simulations just by opening a browser page. As all computations except rendering are done on a remote server, even less powerful devices such as tablets or smart-phones may interact with running simulations.

This paper is organised as follows. A review of related research work is given in Section 2. Section 3 outlines the software architecture for simulation and visualisation based on the functional mock-up interface and web technologies, while a case study is presented in Section 4. Finally, discussion and conclusion is given in Section 5.

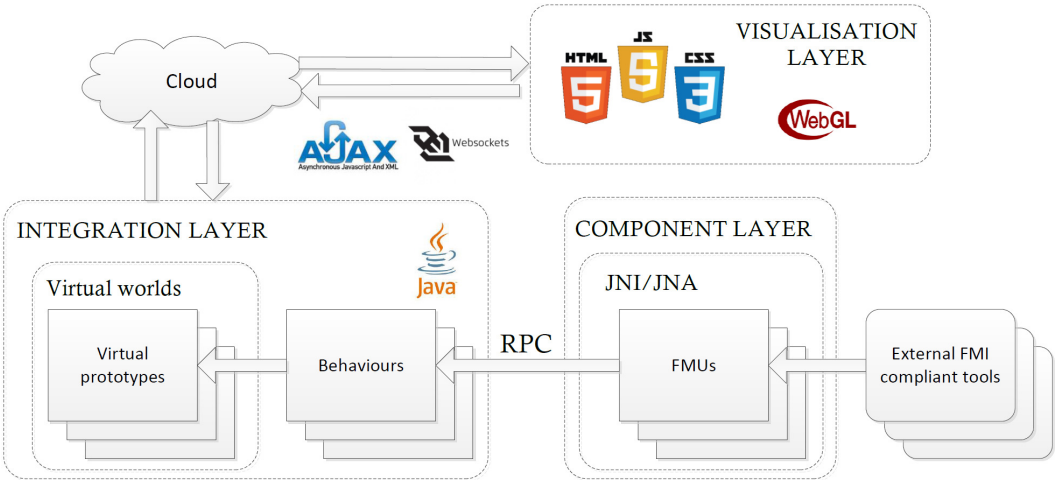

Figure 1. High level software architecture

2 Related work

Component-based simulations is a common topic in the literature, together with other approaches to modularising simulations. After the introduction of FMI, several authors have shown how to use it in modular simulations. A recent example is Neema et al. (2014), who used FMI for component models which where integrated using High-Level Architecture (HLA). Their system supports component models from a range of different modelling tools, in addition to FMI support.

Delafossei et al. (2012) discuss the use of simulation at every stage of the design and development, taking hybrid vehicles as a sample scenario. They start with a high-level model of the complete system, using generic components described by analytic models or lookup tables. When the components are developed later in the process, detailed behavioural models can be integrated into the original system model.

A comprehensive survey of web-based simulation is provided by Byrne et al. (2010). They define web-based simulation as any approach which uses the web browser to provide a user interface for the simulation. They classify different web-based simulation systems according to the role taken by the web browser. The simulation can be client-side, server-side, or hybrid. The latter runs the simulator on the server and the visualisation layer on the client, while the former run both in the web browser or both on the server, respectively. The web browser can also take other roles, such as providing access to a model repository or to documentation.

Several authors have applied WebGL to provide full 3D rendering in web-based simulations. A client-side approach is taken by McMullen et al. (2012), who implemented simulation of abstract models in JavaScript, with visualisation in WebGL for simulation. Pang et al. (2013) took a hybrid approach in a system for interactive e-learning environment for high performance buildings (HPB). They used using a single Functional Mock-up Unit (FMU) as the simulation back-end running on the server.

Several authors have suggested to run simulations in a service oriented architecture (SoA). An early example is zu Eissen and Stein (2006) who proposed a method to simulate Modelica models as a Web Service. Their work is limited to non-distributed, single-user simulations. They used the YANOS simulator for Modelica and wrapped it in SOAP to provide the web service.

Wang and Zhang (2012) have developed a service-oriented and web-based framework for virtual prototyping in a distributed and collaborative environment. They focus on integration of different modelling tools, using High-Level Architecture (HLA) as part of the architecture. System-level modelling is explicitly out of scope. They also give a comprehensive historic overview, which is worth reading.

Zhang et al. (2010) take a model-driven approach to system-level modelling, and the development of a modular simulator. This means that some of the code needed to realise the simulator can be automatically generated from abstract models of the complete system. The works of Zhang et al. (2010) and Wang and Zhang (2012) are not directly applicable to virtual prototyping. They focus on the structured development of a simulator in a waterfall-approach. They do not discuss how domain experts can tinker with a virtual prototype in a trial and error fashion.

More recently, De Filippo et al. (2014) discuss a modular architecture for driving simulators. They give a historic overview of simulation in the automotive industry, discussing shortcomings of previous generations of simulators. Their own approach is called FDMU and is based on a Service-Oriented Architecture (SOA). The objective is similar to that of Distributed Interactive Simulation (DIS) and High-Level Architecture (HLA), but it is claimed that SOA gives looser coupling. SysML is adopted for wrapper design in FDMU. The FDMU set-up includes both a visualisation component and dedicated user interaction hardware.

DOI 10.3384/ecp15119123

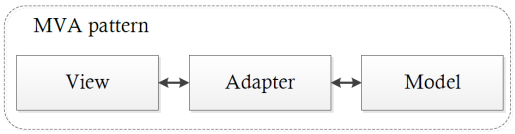

Figure 2. Model View Adapter pattern

3 Software architecture

Fig. 1 shows the high level architecture of the presented software architecture. It is based on the Model-View-Adapter (MVA) pattern and uses a centralised primary architecture (Pečiva, 2008). The MVA pattern is similar to the more well known Model-View-Controller (MVC) pattern, but differs by arranging the model, view and adapter linearly without any connections between the view and the model. This means that the view is completely decoupled from the model, such that the view and model can only interact via a mediating controller or adapter as seen in Fig. 2. As the model is oblivious to it's presentation, the view/user interface can be implemented in any language/-tool without modification of the underlying business logic. It should be noted that in the implemented MVA pattern, the model is oblivious to the adapter as well. In stead of implementing an observer pattern, the adapter reads the state of the model per request. The adapter is implementing the *singleton pattern*. In this way, as the views/user interfaces can only interact with the model through the adapter, it will know when a major user-initialised change has occurred. Allowing it to notify other clients if needed. The adopted MVA pattern is shown in Fig. 3.

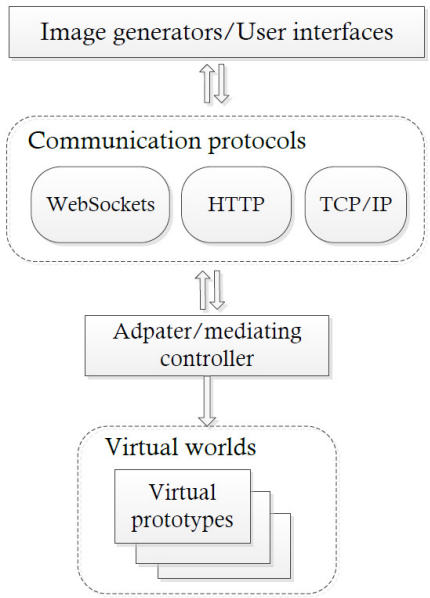

Figure 3. Adopted MVA pattern

3.1 Communication

Real-time communication between clients and the server is handled using the WebSocket protocol, while files are served using asynchronous JavaScript and XML (Ajax). The major benefit of utilising WebSockets, apart from the reduced latency, is the bi-directional communication capability. Before the introduction of WebSockets in HTML5, full-duplex transmissions between client and server was not straightforward. However, some methods for real-time data exchange based on HTTP has been available using *polling*, *long-polling* and *streaming* mechanisms. But these methods involve unnecessary HTTP request and response headers, which introduce latency, and the server itself cannot initiate a connection using the standard HTTP model (Loreto et al., 2011).

Google protocol buffers (Varda, 2008) are used to serialise data transmitted between the client and server. Protocol buffers have been chosen because of the language and platform neutral nature, the small overhead and the well defined message structure. In particular, messages sent using the protocol buffers are pre-defined inside files with a *.proto* extension. *Protobuf.js* (Wirtz, 2013) has been used in order to add support for protocol buffers in JavaScript, while support for C++, Java and Python, as well as some other languages, are bundled together with the protocol buffers.

An example protocol message is given in Listing 1. Actually, this message contains all the information needed to update the browser 3D visuals as the initial *Node* is the top-most node in the scene graph, and each child is also a Node. The optional geometry is only sent once if present, and contains the information necessary for rendering.

Listing 1. example .proto message

```
message Node {
    required int32 ID = 1;
    required string name = 2;
    required Vector3 worldPosition = 3;
    required Quaternion worldQuaternion = 4;
    repeated Node children = 5;
    optional Geometry geometry = 6;
}
```

3.2 Visualisation layer

The visualisation layer consists of a set of HTML5 web pages handling user input, displaying 3D visuals, 2D plots and other types of data presented according to users requirements. In order to preserve state when transitioning between different HTML pages during a user session, Java Server Pages (JSP) has been utilised, which allows variables to be serialised into Java beans.

WebGL is used to render the 3D visuals. In particular, WebGL is a cross-platform Javascript API for rendering 3D graphics inside of an HTML5 <*canvas*> ele-

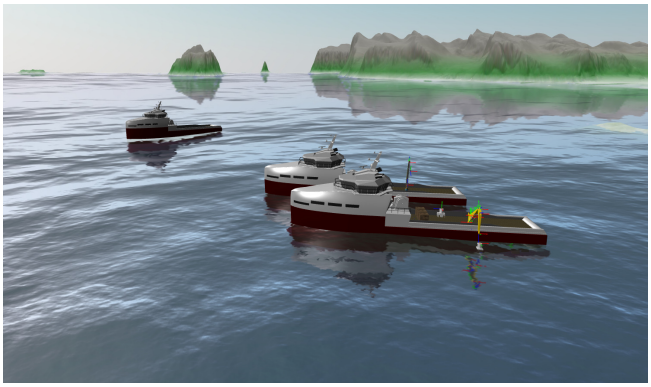

Figure 4. Virtual world rendered using WebGL

Figure 5. Real time plots available through the browser

ment, without the use of plug-ins. WebGL is based on OpenGL ES 2.0, and Version 1.0 of the standard was released in 2011 and is today widely supported in modern browsers, both desktop and mobile versions. Fig. 4 shows a virtual world rendered in the browser using WebGL. The open-source Javascript library *three.js* (Cabello, 2010) has been used to simplify WebGL interaction. *Three.js* supports a number of common 3D formats such as Wavefront .OBJ, Standard Tessellation language (.STL) and Collada (.DAE) to name a few. The 3D models generated by the server side application are transmitted through the WebSocket binary stream, while models available from a remote web server are loaded asynchronously through Ajax calls. However, due to cross-site HTTP requests being subject to restrictions for security reasons, a client-side request made from domain A to load a resource from domain B will be denied by web browsers. A simple way to bypass this restriction is to let domain A pass the request through a server-side resource, acting as a proxy, not bound by the same restrictions.

The plotting library, *CanvasJS*, has been used to simplify the creation of 2D plots, and a web page displaying real-time plots from a running simulation is shown in Fig. 5. Plotting works by having the client poll for updates at a regular interval. These updates contains the most current data at the time of the poll. Caching is done by the client in order to save bandwidth, and to let the view decide how many data points that should be visible.

3.2.1 Terrain

In some cases, the inclusion of real world terrain data can help create a more vivid and realistic simulation environment. Something which is especially true for training simulators. In this system, such data is specified as digital elevation models (DEMs). DEMs of the Norwegian mainland was released to the public in 2013 by the Norwegian Mapping Authority, with a resolution down to 10x10 meters. Using the Geospatial Data Abstraction Library (GDAL) (GDAL Development Team, 2015), the DEM files can be exported to a more lightweight and WebGL friendly format.

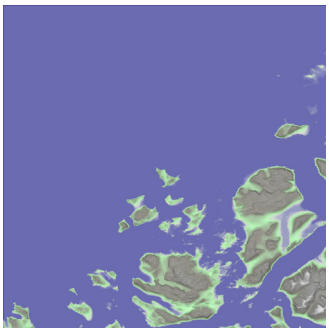

Figure 6. Terrain texture based on DEM data generated by combining hill-shade, hill-slope and color-relief textures

GDAL is also able to produce color-reliefs, hill-shade and hill-slope textures from the DEM source. These are combined into a single texture and mapped onto the terrain. An example of such a generated texture is given in Fig. 6.

3.2.2 Ocean Waves

Ocean waves are created by having the server and any connected clients implement the same wave equation. In this way, time is the only variable needed to be shared and is controlled by the server as seen in fig 7. The benefit of this approach is that views can set the height-map for an arbitrary sized mesh without increasing the load on the server. The server itself only needs to calculate the height-map around vessels and other floating objects.

DOI 10.3384/ecp15119123

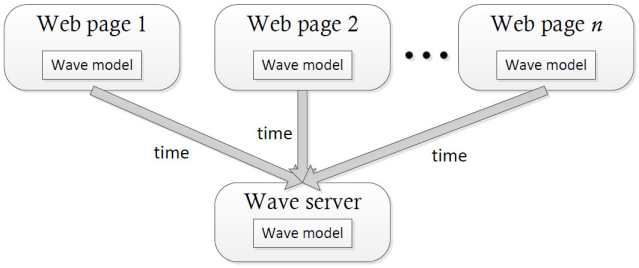

Figure 7. Wave generation principle

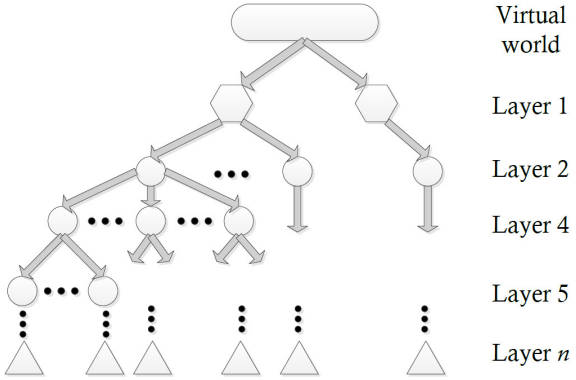

Figure 8. General scene graph layout

3.3 Integration Layer

The integration layer has been written using the Java Platform, Enterprise Edition (EE), which is built on top of the Java Standard Edition (SE) and provides the means for developing and running large-scale and scalable network applications. For instance the WebSocket and Servlet API's used to communicate with the web browsers are bundled with the EE API.

3.3.1 Virtual world

The virtual world, or collaborative virtual environment, is the simulated world in which the *virtual prototypes* exists. Simple objects such as crates etc. can also inhabit this world in order to increase realism, but are not necessarily important for the simulation in question. All objects in the same world are stepped forward using the same time step, such that a simulation may be slowed down or accelerated forward depending on the needs. The virtual world is managed by a scene graph in charge of managing the parent-child relationships and transformations of all the *nodes* in a scene. These nodes are arranged in a tree structure as shown in Fig. 8. A node can have zero or more children, but only one parent. Properties of the nodes include the local and world transformations along with an optional 3D representation. *Dual-quaternions* are used to represent these rigid-body transformations as they have been shown to be the most efficient and compact form of representing rotation and translation in a unified way (Kenwright, 2012).

Virtual worlds are generated by uploading configuration files to the server. These are written using the YAML data serialization format. YAML is similar to JavaScript Object Notation (JSON), but has additional features such as comments and anchors, and has been chosen because it is easy to parse, has a rich feature set and is easy to manually read/edit.

3.3.2 Virtual prototypes

The *Virtual prototypes* are the objects in the virtual world that users can interact with, edit, monitor etc. It could be a crane, winch system, propulsion system etc. In order for it to be a virtual prototype, one should be able to change some physical property of the object in order to see how this change would affect it's performance.

3.3.3 Behaviours

Behaviours alters the state of a virtual prototype over time, and could encapsulate the functionality of zero or more FMUs from the *component layer*. Several behaviours can be attached to a virtual prototype in order to simulate the desired functionality.

3.4 Component Layer

Simulation models can be written in a number of different domain specific tools. Integration of models across these tools are not necessary straightforward, and success depends on the tools having support for the same mechanisms for sharing data.

The integration of simulation models created in various software tools is achieved through the use of the FMI standard, by letting tools export their models as FMUs implementing the *co-simulation* standard. Basically, an FMU is a compressed folder consisting of a combination of compiled C code, describing the model equations, and XML files specifying the variables used. The most notable difference between a *model-exchange* and *co-simulation* FMU is that the latter includes its own solver. The open source library *javaFMI* (javaFMI Development Team, 2013) is used to simplify interaction with the FMUs. In particular, *javaFMI* takes care of unzipping, parsing the attached XML and invoking the pre-complied C code using Java Native Access (JNA).

Functionality from FMUs in this layer is accessed trough RPC calls. In this way, available FMUs can be distributed and computed on some remote resource. Currently, Remote Method Invocation (RMI), which is built into the Java API, is used as the RPC mechanism. As FMUs are invoked remotely, the server hosting them and the client accessing them are not required to run on the same platform or share bitness. That is, an RPC request

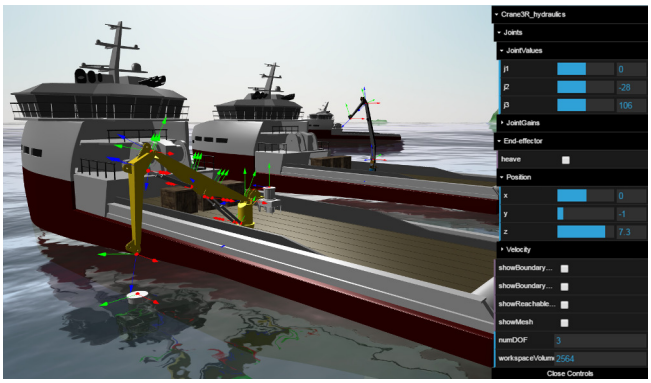

Figure 9. Controlling a crane in the virtual world

originating from a 64-bit windows program can take advantage of an FMU compiled as a 32-bit linux library.

4 Case Study

The presented software architecture has been used to implement a virtual prototyping system prototype for maritime crane design and operation (Yingguang et al., 2015). The crane kinematics and hydraulic properties are defined in a YAML document and uploaded to the server. In this prototype, dynamics has not been considered, but is planned to be added once a physics engine has been integrated. Links and joints are placed in the scene graph hierarchy by parsing the crane kinematics. The hydraulic properties are inputs to the FMUs making up the hydraulic system, which consists of a hydraulic motor for the base and hydraulic cylinders for the jib and boom. These were modelled using bond graphs in the modelling tool *20-sim* (Weustink et al., 1998). Multiple cranes can be defined in the same document, along with other world objects. In Fig. 9, two cranes have been defined. The widget visible to the right is used to interact with the cranes. Changes done to this widget are propagated to the server, which forwards the requested action to the crane model, and other users interacting with the same crane will have their widget updated accordingly. This particular widget uses Cascade Style Sheet (CSS) to overlay itself on-top of the WebGL <*canvas*>, but widgets could also be regular HTML pages. Additionally, real world terrain, multiple vessels and simple geometric entities have been included. Using the approach described in Hatledal et al. (2015), the workspace of the cranes are calculated and saved as a 3D model made accessible to the visualisation.

For a more elaborate description of the case study, refer to Yingguang et al. (2015).

5 Discussion and Conclusion

This paper presents a software architecture for simulation and visualisation based on FMI and web technologies. FMI allows for the integration of simulation models from different domains tools, while the use of web technologies enables a highly flexible way of presenting data. Furthermore, as simulations are carried out server side, no software other than a browser is required to interact with them. As the simulations are centralised on a remote server, multiple user can interact with the same simulations for collaborative purposes.

A key feature of the architecture is that a user does not have to download the models with which he interacts. A proprietary behaviour model can just as well run on the servers of its owner, giving access only to the dynamic and observable variables. A 3D model would have to be downloaded to the client, but needs only to contain the data relevant for visualisation. Thus this is a potential solution to open for collaboration between secretive partners.

There are a number of immediate challenges. As the complexity and size of the world to simulate increases, manually editing the configuration document defining the virtual world becomes cumbersome and inefficient. To simplify this process, a visual editor should be implemented.

Our current focus is to integrate the system with a physics engine, to allow accurate simulation of physical and mechanical laws. After the initial submission of this paper we have made significant progress towards integration of *AgX Dynamics* from Algoryx Simulation AB, which is highly suited for high fidelity maritime simulations. This will be discussed in future papers.

Acknowledgements

The contribution of Chu Yingguang, Deng Yuxiang and Filippo Sanfilippo is highly appreciated by the authors.

The project work of this paper is financially sponsored by a grant from the Norwegian National Research Foundation (Innovation Projects for the Industrial Sector MAROFF, ES486092).

References

Torsten Blochwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, et al. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *9th International Modelica Conference*, 2012.

James Byrne, Cathal Heavey, and P.J. Byrne. A review of web-based simulation and supporting tools. *Simulation Modelling Practice and Theory*, 18(3):253 – 276, 2010. ISSN 1569-190X. doi:http://dx.doi.org/10.1016/j.simpat.2009.09.013. URL `http://www.sciencedirect.com/science/article/pii/S1569190X0900149X`.

Ricardo Cabello. Three. js. *URL: https://github. com/mrdoob/three. js*, 2010.

F De Filippo, A Stork, H Schmedt, and F Bruno. A modular architecture for a driving simulator based on the FDMU approach. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 8(2):139–150, 2014.

Vincent Delafossei, Scott Stantoni, Takayuki Sekisue, and Junsik Yun. A methodology to use simulation at every stage of a hybrid vehicle design. In *IEEE Vehicle Power and Propulsion Conference*, October 2012.

Ian Fette and Alexey Melnikov. The websocket protocol. 2011.

GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, Version 1.11.2*. Open Source Geospatial Foundation, 2015.

Lars I Hatledal, Filippo Sanfilippo, and Zhang Houxiang. A voxel-based numerical method for computing and visualising the workspace of offshore cranes. *ASME 34th International Conference on Ocean, Offshore and Arctic Engineering*, 2015.

javaFMI Development Team. javafmi. *URL: https://bitbucket.org/siani/javafmi/wiki/Home*, 2013.

Ben Kenwright. A beginners guide to dual-quaternions: what they are, how they work, and how to use them for 3d character hierarchies. 2012.

Salvatore Loreto, P Saint-Andre, S Salsano, and G Wilkins. Known issues and best practices for the use of long polling and streaming in bidirectional http. *Internet Engineering Task Force, Request for Comments*, 6202(2070-1721):32, 2011.

Chris Marrin. WebGL specification. *Khronos WebGL Working Group*, 2011.

TH McMullen, KA Hawick, VD Preez, and B Pearce. Graphics on web platforms for complex systems modelling and simulation. In *Proc. International Conference on Computer Graphics and Virtual Reality (CGVR'12)*, pages 83–89, 2012.

Himanshu Neema, Jesse Gohl, Zsolt Lattmann, Janos Sztipanovits, Gabor Karsai, Sandeep Neema, Ted Bapty, John Batteh, Hubertus Tummescheit, and Chandrasekar Sureshkumar. Model-based integration platform for fmi co-simulation and heterogeneous simulations of cyber-physical systems. In *10th International Modelica Conference*, pages 10–12, 2014.

Xiufeng Pang, Raj Dye, Thierry S Nouidui, Michael Wetter, and Joe J Deringer. Linking interactive Modelica simulations to HTML5 using the functional mockup interface for the LearnHPB platform. August 2013.

Jan Pečiva. A summary of active transactions in collaborative virtual environments. In *Proceedings of the Annual Strathmore University ICT Conference 2008*, pages 1–1, 2008. URL `http://www.fit.vutbr.cz/research/view_pub.php?id=8765`.

Kenton Varda. Protocol buffers: Googles data interchange format, 2008.

Hongwei Wang and Heming Zhang. Using collaborative computing technologies to enable the sharing and integration of simulation services for product design. *Simulation Modelling Practice and Theory*, 27:47 – 64, 2012. ISSN 1569-190X. doi:http://dx.doi.org/10.1016/j.simpat.2012.05.002. URL `http://www.sciencedirect.com/science/article/pii/S1569190X12000664`.

PBT Weustink, TJA De Vries, and PC Breedveld. Object-oriented modeling and simulation of mechatronic systems with 20-sim 3.0. *Mechatronics*, 98:873–787, 1998.

D Wirtz. Protobuf. js. *URL: https://github. com/dcodeIO/ProtoBuf. js*, 2013.

Chu Yingguang, Lars I Hatledal, Filippo Sanfilippo, Hans G Schaathun, Vilmar Æsøy, and Zhang Houxiang. Virtual prototyping system for maritime crane design and operation based on functional mock-up interface. *OCEANS: Discovering Sustainable Ocean Energy for a new World*, 2015.

Heming Zhang, Hongwei Wang, David Chen, and Gregory Zacharewicz. A model-driven approach to multidisciplinary collaborative simulation for virtual product development. *Advanced Engineering Informatics*, 24(2):167–179, 2010.

Sven Meyer zu Eissen and Benno Stein. Realization of web-based simulation services. *Computers in Industry*, 57(3):261 – 271, 2006. ISSN 0166-3615. doi:http://dx.doi.org/10.1016/j.compind.2005.12.007. URL `http://www.sciencedirect.com/science/article/pii/S0166361506000169`. Advanced Computer Support of Engineering and Service Processes of Virtual Enterprises Advanced Computer Support Special Issue.