Next generation web based live data monitoring for W7-X

Simon Dumke, Heike Riemann, Torsten Bluhm, Robil Daher, Michael Grahl, Martin Grün, Andreas Holtz, Jon Krom, Georg Kühner, Heike Laqua, Marc Lewerentz, Anett Spring, Andreas Werner and the W7-X team

Max-Planck-Institut für Plasmaphysik, Teilinstitut Greifswald, Wendelsteinstraße 1, 17491 Greifswald, Germany

The Wendelstein 7-X stellarator experiment by its nature has the need for continuous live data monitoring. New technical and security related requirements make a redesign of the monitoring architecture, originating in the first years of the century, necessary. Besides that, new developments in technologies, user interfaces and generally IT supported work flows, generate new expectations at the system user level, including live remote participation.

To answer these upcoming demands, the CoDaC team at Wendelstein 7-X has designed a new architecture for monitoring data management, distribution and observation. As a very flexible user interface, a client web application was developed, utilizing modern technologies like WebSockets and WebGL for high performance live data visualization. A touch friendly interface design offers both broad support for new access tools (like tablets, smart phones, smart TV sets) and a very flexible and intuitive work flow.

This paper describes the new MonA-LISA monitoring architecture and gives a short overview over the options provided by the web based monitoring client.

Keywords: Wendelstein 7-X, continuous operation, remote participation, user tools, online data monitoring, CoDaC, Web technologies

1. Introduction

The Wendelstein 7-X experimental stellarator differs from most other fusion experiments in one respect: Instead of relatively short pulses for which measured data can be stored locally during the experiment, then collected centrally and analyzed afterwards, W7-X aims for steady state operation and already works with multiple discharges preconfigured as one experiment program with a duration of up to 10 minutes. This results in many special project requirements – one of them being continuous monitoring of experimental measurement data as well as operational machine data. Handling both kinds with one single tool and supporting life visualization even of 24/7 data signals reliefs scientific experimenters as well as responsible personnel from the obligation to work blindly during the experiment run. But on the other hand, this requires monitoring data to be immediately[1] and continuously distributed over computer networks for inspection

1.1 Requirements

During a live data monitoring session, the group of end users is quite varying in respect to numbers of clients, diversity and numbers of monitored data signals as well as use cases and usage environments. Combined with an ever growing number of targeted client tool platforms from desktop computers and various mobile devices to large-scale control room video wall installations, the need arises for a next generation network-based, platform independent data monitoring architecture. The resulting system has to be reliable[2], highly scalable (for future long term experiments even *during* a measurement session), light and unobtrusive on the network infrastructure and to offer users a maximum of flexibility in usage and platform support. In addition, international cooperation brings the requirement for remote participation capabilities.

1.2 Definition of Signals

A signal (see also [21]) in the context of this paper is a stream of data that provides one data sample per time stamp. A sample can be a scalar value (time series signal like the current through a specific coil), a vector of scalar values (e.g. a spectral energy distribution), or even a multi-dimensional matrix of values. For special kinds of signals a sample might also be an image (video signal like the images from a video camera), a text string (textual message signal like Xcontrol's current program name [12] or a log message) or even data-less events[3].

2. Related Work

Live monitoring of measurement data is not yet a heavily active field in fusion research, because most experiments are presently working with short plasma pulses (shots), during which measurements are recorded and reviewed and analyzed afterwards.

2.1 Monitoring at W7-X

At W7-X, the goal of continuous operation has from the beginning been kept foremost in the minds of the system architects and CoDaC researchers [1][2][3][4].

[1] Requirements demand for them to reach the monitoring screen within 100 milliseconds (where technically possible).

[2] Reliable in the sense of getting back up to running conditions after a (unlikely) full drop out within 10 minutes.

[3] Event signals like the start of a new Xcontrol program segment, where the time stamp itself is the relevant information

author's email: simon.dumke@ipp.mpg.de

This also led to much research and development in the field of online data monitoring [5], driven by a sophisticated requirements analysis [6].

Specifically, Hennig et al. described "A concept of online monitoring for the Wendelstein 7-X experiment" in 2004 [7], thereby laying the base for years of research, work, usage and studies, culminating in the new W7-X monitoring architecture described herein. Most concepts and analysis from that time are still valid and have heavily influenced this current work, although the whole system has evolved over the years. The most profound differences from this early design are (or are caused by) the necessary drop of the UDP multicast reliance and the technological step forward to modern web technologies on the client side.[4]

2.2 Other Applications of Online Data Monitoring

While data monitoring is not yet a hot topic in fusion research, other fields of science and business do handle similar requirements and demands with similar solutions.

Konno et al. [8], designed their monitoring system for the Double Chooz reactor neutrino experiment (presented at CHET2010). Their goal was the creation of a general software framework for online data monitoring, although they were limited to a very specific kind of data (histogram data in this case). The system they came up with has many similarities with the new W7-X monitoring architecture described in this paper, although it can neither handle the flexible data types needed within the W7-X project nor offer the required scalability because of its reliance on a single server for data collection and distribution.

In times of growing process and home automation, online data monitoring is also a topic heavily discussed outside the purely scientific world [9], and many tools and libraries exist to assist in the process of implementing an online data monitoring system [10].

3. MonA-LISA Architecture

To meet the demands and requirements of the Wendelstein 7-X stellarator fusion project, the Control, Data Acquisition and Communication (CoDaC) group at W7-X developed and implemented the new data monitoring architecture MonA-LISA described in this paper, comprising four primary components:

- **Distributed acquisition**: A specialized data source component for the on-site data acquisition framework
- **Flexible distribution**: A runtime-scalable, stackable network of data multiplexers, managing client subscriptions
- **Centralized knowledge**: A central repository of knowledge about all available data signals
- **Comfortable access**: A highly flexible client web application allowing subscription and visualization of measured data signals, events, experiment progress etc.

[4] Web technologies were chosen for their platform independency, their broad support and their probability of broad and long term availability.

3.1 Context

The MonA-LISA architecture is embedded into the existing CoDaC environment at W7-X. The scope of the architecture begins at the point where monitoring data comes in and ends with the visual delivery to the client user's screen. As opposed to previous definitions [7], this scope does not include any data analysis features (producing derived data from measured inputs) anymore. Instead, this functionality is now provided by the CoDaStation framework [13] and the W7-X Online Analysis framework (see Fig. 1). The architectures data source (MonA, see 3.2) is therefore integrated into a CoDaStation, which supplies the data to finally be visualized for the user.

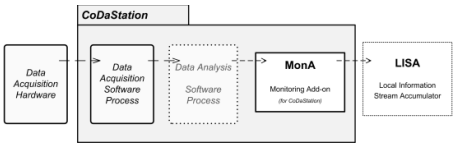

Fig. 1 - Data Analysis within the CoDaStation Framework

Information about the available data signals resides within the configuration database [14]. This information is (more or less) static, as it reflects the current state of the W7-X machine. From it, the MonA-LISA architecture derives its knowledge about those signals.

Most diagnostic systems do only supply live data during the experiment run, or even only for a small portion of it. This non-static information is derived on the fly by the components of the MonA-LISA architecture themselves.

3.2 Architecture Overview

The MonA-LISA architecture consists of four primary components (Fig. 2):

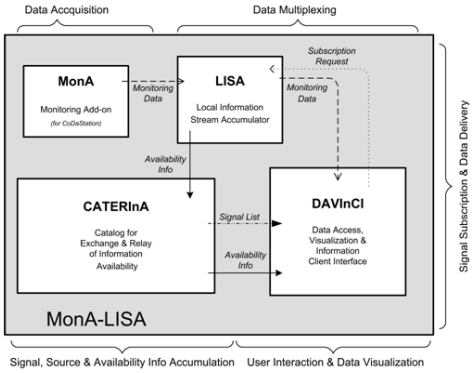

Fig. 2 - Components of the MonA-LISA Architecture

MonA (Monitoring Add-on) is a CoDaStation add-on for the acquisition of monitoring data in the CoDaStation environment. When provided with data by the CoDaStation infrastructure, MonA re-formats those data, splits them into signals[5] and packages those data for transfer to a preconfigured LISA.

LISA (Local Information Stream Accumulator) is a light weight data stream multiplexer and subscription handler for signal data produced by MonA(s) and asked

[5] SignalConsumers in the CoDaStation world really do consume groups of signals (often referred to as streams in reference to the analogous ArchiveDB concept)

for by DAVInCI(s). When a MonA connects to a LISA via TCP and offers monitoring data for a specific signal, LISA registers itself with CatERInA as a source for the specific signal. Clients (like DAVInCIs) looking for that signal will then find the LISA in the central data base (CatERInA), connect to it and subscribe the signal.

All data packets LISA receives from a MonA are forwarded immediately and unchanged to all subscribing clients. MonA and LISA are working together following a Producer Consumer Pattern [15].

CatERInA (Catalog for Exchange and Relay of Information Availability) is the central information repository in the MonA-LISA network. It offers and maintains a list of all signals that are (possibly) available for monitoring through the MonA-LISA network. LISAs supply information about the signals that can be subscribed from them to CatERInA. The W7-X configuration database acts as a background information source for CatERInA, which regularly updates the list of all possibly existing signals.

DAVInCI (Data Access, Visualization and Information Client Interface) is the web and browser based user interface to the MonA-LISA network. It offers the user options to select and visualize an arbitrary number of signals from the list of all available monitoring data. To achieve this, DAVInCI interacts with LISA and CatERInA following the Publisher Subscriber Pattern [11]: DAVInCI handles the un-/subscription and reception of the signals' data from the MonA-LISA network (more precisely, from the corresponding LISAs) and regularly asks CatERInA[6] for an update of the signals and sources list.

3.3 LISA as the core of flexibility

The core of the MonA-LISA architecture's flexibility that makes it lightweight, scalable and adaptable is LISA, the general purpose monitoring data multiplexer.

In a straight forward and simple deployment set, one or multiple MonAs would deliver all their monitoring data to a single LISA instance, which would than distribute them to all interested parties (Fig. 3).

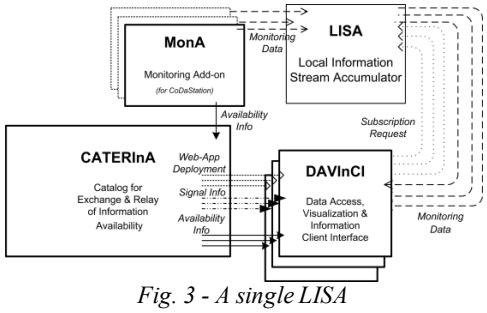

Fig. 3 - A single LISA

In a more demanding environment (with lots of data, many clients etc.), multiple LISA instances can be deployed to each handle a subset of all available data streams to distribute the load among them. In that scenario, each specific signal will be available through one specific LISA[7] while each LISA can be responsible for an arbitrary number of signals (Fig. 4).

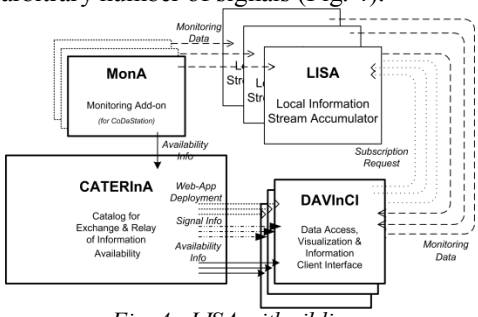

Fig. 4 - LISA with siblings

If such siblings are not enough to handle the load (e.g. one high bandwidth signal like a video stream is subscribed by a great many clients), LISA can also be stacked (or get child nodes): A stacked LISA subscribes one signal (or multiple signals, or all) from its parent LISA and thereby becomes a source for that signal itself. This means, it informs CatERInA that it can offer data for that signal, accepts client subscriptions for it and forwards all data packets from its parent to the registered subscribers. A fully deployed MonA-LISA in the W7-X environment is depicted in Fig. 5.

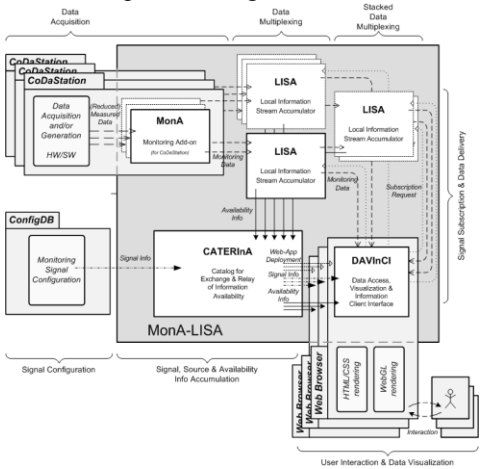

Fig. 5 - MonA-LISA – fully deployed at W7-X

By repeatedly applying the orthogonal concepts of stacked and sibling LISAs, very complex and flexible networks can be created.

The stacked LISA functionality can also be used to deploy domain specific LISAs. An example for this use case is remote participation via the W7-X extranet. A LISA process running on a computer within the extranet can subscribe all (relevant) signals from the primary LISA instances and offers its services via CatERInA. Extranet clients are then allowed to access the web server on CatERInA (e.g. via reverse proxy server) and the domain specific LISA. All signal subscriptions from within the extranet will then be handled by this single LISA[8], removing the remote participation load from the internal systems[9].

[6] CatERInA (being web server, the single-source-of-truth in the MonA-LISA network and a must-be-known for all clients) also acts as a web server for the static files of the web based DAVInCI client application.

[7] CatERInA collects the information about which LISA distributes which signals and makes this knowledge available to clients

[8] The extranet-LISA is not reachable by internal clients and is therefore ignored as a potential data source. In the same manner, extranet clients can solely reach that LISA.

[9] More precisely, limiting the load to that generated by one local client.

4. Implementation and operation

In accordance with established W7-X CoDaC procedures, the infrastructure components MonA, LISA and CatERInA are implemented using the Java programming language. While MonA (as a CoDaStation Add-On) has to be Java anyway, for LISA and CatERInA the same language was chosen in order to get maximum advantage of the CoDaC tool chain [20].

Since this approach introduces some risk of reduced system performance, the overall system load (focused at the multiplexer node LISA) is minimized by making LISA as thin an interface layer as possible[10] and through the heavy use of performance optimized asynchronous I/O via the Netty networking framework [16].

This leaves a single heavy duty module within the MonA-LISA system: The data visualization part of the DAVInCI client.

4.1 DAVInCI – web client design considerations

In order to achieve a maximum of flexibility in target device and platform support and a reduced complexity in the software deployment, the DAVInCI client of the MonA-LISA architecture was designed as a browser-based web application. This point took heavy influence into two main decision points of the implementation:

4.1.1 Data distribution technology

Excluding the historically important possibility of HTTP long polling [17][11], two technological possibilities for the reception of asynchronous or streaming data are supported in current web browsers: WebSockets[18] and WebRTC[19].

While WebSockets are an extension of the traditional HTTP protocol aiming to support bidirectional, stream oriented binary or textual data communication between web server and client, WebRTC is a new technology with the purpose to allow direct peer-to-peer communication between web application instances (each running inside a web browser on any internet connected device).

WebRTC's orientation for real time data streaming, as well as its flexibility in terms of stream/message orientation and reliability make the technology a seemingly optimal candidate for the task. But further research into this field reveals two critical points standing against it, and finally tips the scale towards WebSockets:

- **Immaturity:** WebRTC is not yet fully standardized, browser support is lacking and "server side" software implementations/libraries outside the browser world are very few and limited.
- **Complexity:** The WebRTC stack targets browser-to-browser P2P communication and therefore also handles fields like NAT[12] traversal, signaling etc. These features are all unnecessary inside the MonA-LISA architecture but introduce a lot of complexity in both implementation and application.

In order to keep the option to switch to WebRTC (or any other future alternative technology) when maturity and library support have grown, the application layer protocol used for MonA-LISA has been designed as self containing, message oriented, binary and carrier independent, and the system as a whole was created with a focus on modularity and extensibility.

4.1.2 Data visualization framework

The most important part of the online data monitoring system (besides the distribution of monitoring data) is the visualization of data. To realize this inside a web browser environment, two and a half technologies are generally referred:

- **Scalable Vector Graphics (SVG) [22]** is a XML-based open markup language format for vector graphics. It has strong support throughout modern web browsers and can be freely created, modified and formatted from JavaScript using the established Document Object Model (DOM) [23] API. The specific task of rendering high quality plot graphs using this technique has already been implemented by many groups as diverse open source and/or commercial JavaScript libraries like D3.js [24], Chartist [25], Plotly.js [26][13] and many more.
- The HTML5 **Canvas** Element describes a DOM element type and a powerful JavaScript API that allow the dynamic generation of raster-based images[14]. In short, it allows scripted painting of primitives (like lines, curves, circles, texts etc.) on a rectangular area inside a browser window. As with SVG, many ready-made solutions exist to support the rendering of beautiful data plots of innumerable kinds, e.g. Chart.js [27] and Flot [28].
- For high performance applications, modern browsers also support **WebGL**, an Open GL ES 2.0 based extension for the HTML5 **Canvas**[15] element that allows heavily hardware accelerated graphics to be generated and manipulated through JavaScript. Using this technology would allow to offload most of the heavy lifting of a fluent live data visualization to the distributed computing capability of modern graphics cards. On the other hand, library support for data plot rendering with WebGL is poor, so a full low level implementation would be necessary.

[10] LISA only inspects the first 12 bytes of each incoming data packet, makes some basic checks, dissects the signal ID from there and immediately forwards the untouched data packet to all subscribers. MonA-LISA can deliver data from source (MonA) to sink (DAVInCI) within a measured average of 1.4 msec. Additional latency will be introduced by the data acquisition before reaching MonA and by the screen update rate of the browser/WebGL implementation after reaching DAVInCI.

[11] Long polling is regarded by many as a conceptual breach of the http design principles used to support asynchronous interrupt or streaming data with a technology that did not originally include such support. For this reason, its use is generally discouraged.

[12] Network address translation (NAT) is a method of remapping one IP address space into another by modifying network address information in Internet Protocol (IP) datagram packet headers while they are in transit across a traffic routing device. [29]

[13] Plotly.js is based on D3.js.

[14] Often referred to as „pixel graphics"

[15] Therefore, this is only considered half of a different technology

Intense testing with different JavaScript plotting libraries using SVG and classic Canvas painting as well as custom tailored WebGL rendering led to the conclusion that:

- The two former methods do produce optically great results and offer strong library support. The latter has very limited (plotting-related) library support and does not look quite as good as with the alternative approaches.
- WebGL does, however, offer such a significant performance advantage when rendering large volumes of both scalar and profile data (see 4.2) that it turned out to be the only viable option for the interactive plotting system for MonA-LISA.
- A low level implementation being necessary for WebGL usage anyway, some smart data modeling in the MonA-LISA architecture and data flow can make use of the low level flexibility to allow high grades of performance optimization[16].

4.2 DAVInCI – monitoring client with modern user experience

At the time of this writing, the DAVInCI web client implementation fully supports online visualization of:

- **Scalar signals** (one value per time stamp, e.g. temperature development over time, see Fig. 6 a)
- **Profile signals** (one set of multiple values per time stamp, e.g. a spectral distribution; different view types, see Fig. 6 b)

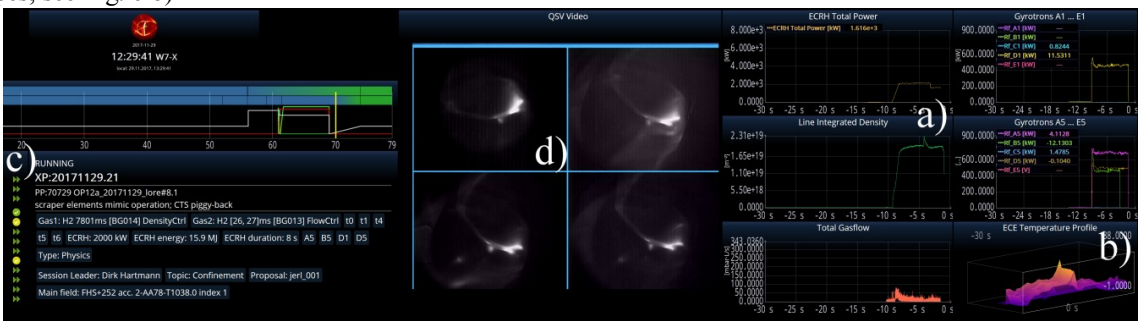

Fig. 6 - DAVInCI – examples of plot types

- **Textual signals** (one textual value, long or short, reflecting a present state, e.g. the name or description of the experiment program currently running, see Fig. 6 c)
- **Video signals** (e.g. an image reflecting the current state of a W7-X component, see Fig. 6 d)

The modern, touch screen ready user interface allows the user to place (using drag and drop techniques) any number of signal plots on his screen (Fig. 7 a), to move and resize them to accommodate his needs (Fig. 7 b) and to add signals for monitoring by dragging them from a list of available signals onto those plots (Fig. 7 c).

A global menu (see Fig. 7 a) and context menu (Fig. 7 d) offer a selection of global as well as plot- and plot-type-specific options like applying themes for printing or large displays, time scaling (1 second to 24 hours), logarithmic scaling of the ordinate axis, different heat map color encodings for profile plots (see Fig. 6 b) etc.

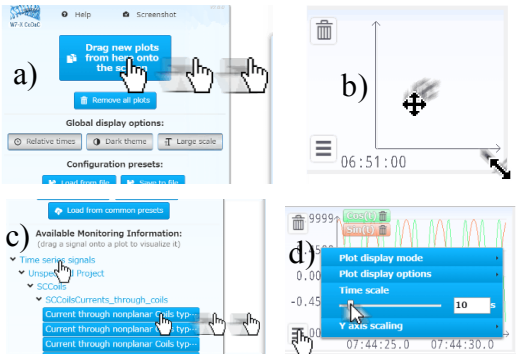

Fig. 7. DAVInCI – Drag & Drop & Interaction

When the user leaves the web application, closes the web browser etc. his current settings[17] (called a preset) are stored in the browsers profile and restored on his return. There is also a set of project-wide general purpose presets available from the menu, and the user can always save a current state as a file and later load it back into a new DAVInCI session[18].

Since the technical works on the W7-X extranet have been completed, the MonA-LISA architecture (with the DAVInCI web client) also offers life data monitoring via that extranet to all members of the W7-X team at:

http://datamonitor.ipp-hgw.mpg.de[19]

5. Conclusions

Based on the MonA-LISA architecture presented in this paper, the DAVInCI web client was realized to enable high performance real time data monitoring using a modern and intuitive user interface. The existing implementation offers plotting options for scalar as well as vectorial profile data (2D and 3D plots), textual event visualization, experiment progress display and video monitoring. The flexibility of the LISA data distribution leaves room for many other signal types like 2d Poincare plots, 3D machine and plasma visualization etc.

[16] Testing and analysis revealed a significant amount of computational power needed purely for data transcription for different ready-made plotting libraries.

[17] The settings include the static plot layout/arrangement and signal assignment, but no fluent signal data.

[18] The new session can also be on a different computer, e.g. after the preset file has been mailed to a colleague.

[19] The link is not publicly available via internet; for extranet access, one needs a W7-X team enrollment and sign a software security policy.

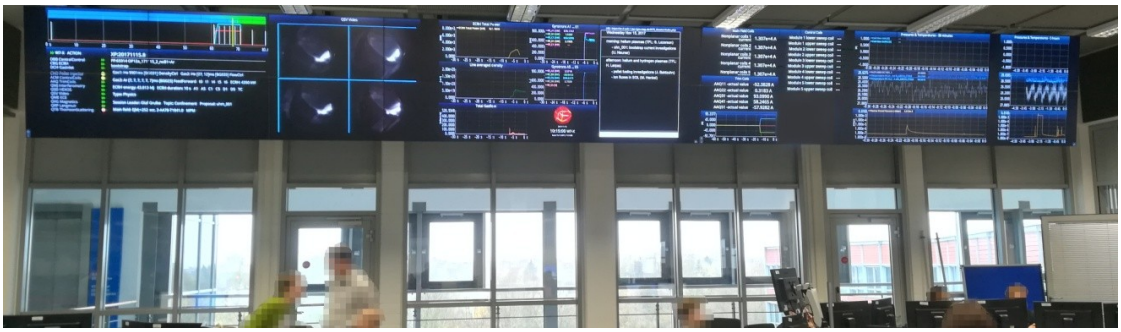

Fig. 8. DAVInCI in the W7-X control room

Inside the IPP, the new system was successfully used during OP 1.2a for data monitoring purposes from the central W7-X HiperWall (see Fig. 8) as well as control room workstations, offices and labs. Up to 50 concurrent DAVInCI clients subscribed over 100 different signals, with the Xcontrol program progress ([3]; state of the running experiment program) and the video stream from inside the plasma vessel attracting the most attention.

Before the beginning of the W7-X operational phase OP 1.2a, the MonA-LISA architecture has been fully deployed using a single LISA instance. As expected, with more and more diagnostic systems supplying data during the operational phase, the load on LISA rose significantly, so a sibling was set up on a second server and the signals split up onto those two machines.

When W7-X extranet came up in September 2017, a domain-specific LISA was stacked above those two. Remote participation partners from EUROfusion, PPPL, ITER etc. where thereby enabled to follow the experiments from outside Greifswald, insulated from the internal productive system. With more increase in signal load, more siblings and stacked LISAs in both domains will be introduced as necessary[20].

6. Acknowledgments

The authors would like to thank all those unnamed (current and former) members of W7-X CoDaC and the W7-X Team who have contributed to the monitoring architecture and/or to the whole W7-X Data Acquisition Framework which gives us things to monitor!

This work has been carried out within the framework of the EUROfusion Consortium and has received funding from the Euratom research and training programme 2014-2018 under grant agreement No 633053. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

7. References

[1] H.-S. Bosch et al.; "Construction of Wendelstein 7-X – Engineering a Steady-State Stellarator" - IEEE Transactions on Plasma Science 38 (3), 265-273 (2010). DoI: 10.1109/TPS.2009.2036918

[2] Ch. Hennig et al.; "Continuous data acquisition with online analysis for the Wendelstein 7-X magnetic diagnostics" - Fusion Engineering and Design 83 (2008) 321–325

[3] A. Werner et al.; "Cutting edge concepts for control and data acquisition for Wendelstein 7-X" - 10.1109/SOFE.2013.6635430

[4] T. Bluhm et al.; "Data access and its implementation at Wendelstein 7-X" - Fusion Engineering and Design 83 (2008) 387–392

[5] Frank Engel; "Architektur für ein Hochleistungs-Monitoring-System in der Fusionsforschung" – Master Thesis – FH Strahlsund – 2012

[6] G. Kühner et al.; "System Requirements - VR-Project: Monitoring System" – Internal RqA IPP Greifswald - 2016

[7] Ch. Hennig et al.; "A concept of online monitoring for the Wendelstein 7-X experiment" - Fusion Engineering and Design Volume 71, Issues 1–4, June 2004, Pages 107–110

[8] Tomoyuki Konno et al.; "Online Data Monitoring Framework Based on Histogram Packaging in Network Distributed Data Acquisition Systems" – CHEP2010 - http://www5.dchooz.org/DocDB/0019/001902/001/chep_online_monitor.pdf

[9] Elegato Systems GmbH – Elegato Eve - https://itunes.apple.com/us/app/elgato-eve/id917695792?mt=8, accessed on 2017-04-03

[10] Marek Otáhal, Olga Štěpánková; "New Tool for Visualization of Time Series and Anomalies in Streaming Data" - Acta Univ. Agric. Silvic. Mendelianae Brun. 2016 - DoI: 10.11118/actaun.2016.64041353P. Eugster et al., The Many Faces of Publish/Subscribe, ACM Computing Surveys, Vol. 35, No.2, June 2003.

[11] P. Eugster et al.; "The Many Faces of Publish/Subscribe" - ACM Computing Surveys, Vol. 35, No.2, June 2003.

[12] A. Spring et al.; "Establishing the Wendelstein 7-X steady state plasma control and data acquisition system during the first operation phase" – Fusion Eng. Des. (2017), http://dx.doi.org/10.1016/j.fusengdes.2017.03.127

[13] T. Bluhm et al., Wendelstein 7-X's CoDaStation: A modular application for scientific data acquisition, Fusion Engineering and Design 89 (2014) 658-662.

[14] G. Kühner et al.; "Editor for system configuration and experiment program specification" - Fusion Engineering and Design 71 (2004) 225–230

[15] M. Grand; "Patterns in Java — Volume 1", John Wiley & Sons, Inc, 1998

[16] The Netty project – http://netty.io – accessed on 2016-09-14

[17] S. Loreto et al.; "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP" - RFC6202 – April 2011

[18] I. Fette et al.; "The WebSocket Protocol" - RFC6455 – 12.2011

[19] W3C WebRTC Working Group; "WebRTC 1.0: Real-time Communication Between Browsers" – Current Version https://www.w3.org/TR/2017/WD-webrtc-20170605/ - 2017-06-05

[20] G. Kühner et al.; "Progress on standardization and automation in software development on W7X" - Fusion Engineering and Design 87 (2012) 2232–2237

[21] J.G. Krom; "Some Ideas about a Storage-Technology-Independent data-access API for the W7X data-archive: The 'Signal Based Access' API" – 2012-03-19 – W7-X CoDaC internal paper

[22] J. Bowler et al.; "Scalable Vector Graphics (SVG) 1.0 Specification" – http://www.w3.org/TR/2001/REC-SVG-20010904/ – accessed 2017-03-24

[23] J. Robie; "What is the Document Object Model?" – https://www.w3.org/TR/WD-DOM/ – accessed 2017-01-13

[24] M. Bostock et al.; "Data Driven Documents" – https://d3js.org – accessed 2016-11-07

[25] G. Kunz et al.; "Chartist.js – Simple Responsive Charts" – https://gionkunz.github.io/chartist-js/ – accessed 2016-11-09

[26] A. C. Johnson et al.; "plotly.js – The open source JavaScript graphing library that powers Plotly" – https://plot.ly/javascript/ – accessed 2016-11-04

[27] N. Downie et al.; "Chart.js – Simple yet flexible JavaScript charting for designers & developers" – http://www.chartjs.org – accessed 2016-11-14

[28] O. Laursen et al.; "Flot – Attractive JavaScript plotting for jQuery" – http://www.flotcharts.org – accessed 2016-11-03

[29] Javvin Technologies Inc.; "Network Protocols Handbook (2 ed.)" - ISBN 9780974094526 - 2005

[20] This is done manually; automated scalability of the network is a topic for future work