PLANNING OF DEPENDABLE REMOTE HANDLING CONTROL SYSTEM ARCHITECTURE FOR ITER

P. Alho[1], A. Hahto[1], J. Mattila[1], L. Aha[1]

[1] *Dept. of Intelligent Hydraulics and Automation, Tampere University of Technology, Finland*

pekka.alho@tut.fi

Abstract. The experimental ITER fusion reactor will feature a number of remote handling (RH) systems that perform maintenance and replacement operations in the reactor. RH control systems must be fail-safe and recoverable, since no humans are allowed in the reactor and a failure could cause major economic losses or setbacks for the research program. ITER is a complex system and requirements for RH system operations are demanding, including radiation tolerance, limits for available space, heavy objects etc. The RH control software must be dependable and operate in real-time, while supporting the changes introduced during the multidecadal lifetime of the ITER plant. Although the fundamentals of implementing teleoperation systems are well-known, the application remains demanding because of the environment, dependability and interoperability requirements. Fault-tolerance techniques based on replication of components can be costly and introduce additional complexity in the system. To achieve the goal cost-efficiently, the research aims to find and utilize the most useful methods from fault prevention, fault removal, fault tolerance and fault forecasting methodologies, all of which are necessary to ensure required dependability. These will be combined in a lean systems engineering framework that will include a reference architecture, hardware and software modules, processes and recommendations for development practices.

INTRODUCTION

ITER is an international experimental nuclear fusion reactor, being built in the South of France. Remote handling (RH) has an important role in the ITER plant, because reactor operation produces high energy neutrons which are absorbed by the components inside the reactor vessel, leaving them activated. Since humans will not have an access to the reactor, all tasks are performed indirectly through remotely operated robots, thus making RH critical for the operation of the ITER.

The focus of this research is in the development of dependable and fail-safe software architecture for the ITER RH control system, used to control a *teleoperated bilateral master-slave manipulator* (Fig. 1). Teleoperated robots can be used in nuclear applications e.g. for inspections and decommission [1]. Although not covered by this research, a mobile robotic manipulator could be used in nuclear catastrophes for repairs, reconnaissance etc. A manipulator can be used to grasp and move objects, or carry tools to perform a variety of tasks like bolting or welding. However, the remote handling systems used to perform these tasks are complex and expensive to design and implement. The development could benefit from the ability to use ready-made components, open standards and other new hardware and software technologies [2]. Moreover, to be able to function with all other systems and machines present in nuclear facilities, the remote handling system needs to be designed to be interoperable.

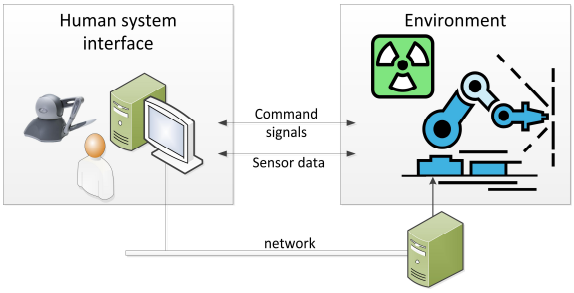

Fig. 1 – Teleoperated bilateral master-slave manipulator system.

Since the robots operate in a hazardous environment, the system must be designed to be *fault tolerant* and *fail-safe*, i.e. it can be guided to a safe state. A shared feature with ITER RH and Mars pathfinders, satellites etc. is that operations are performed independently in an isolated environment. However, unlike in space and foreign planets, robots in the ITER core can be recovered for maintenance, if properly designed, so there is no need for large-scale replication of components. Avoiding solutions resorting to extensive redundancy gives benefits of smaller size, less complexity and better affordability. Less complexity makes the system easier to specify, design, develop and verify. Nevertheless, even though the energy density in the ITER reactor cannot cause a catastrophic failure, the economic losses in the case of an operation failure could be significant – thus the RH system is considered *safety-critical* and needs to take the dependability requirements into consideration in the RH system design. For example, the system should not immediately drop all functionality in the presence of faults, as low availability of control systems leads to increased downtime,

and consequently increased operational costs. Instead, the system should try to isolate the fault and degrade gracefully, so that the fail-safe state is a last resort when nothing else can be done to prevent failure – basic implementation for this functionality is a so-called limp-home mode. In this paper we will present solutions to some of these problems in a systems engineering framework, including architecture, supervisory system and use of off-the-shelf (OTS) components.

BACKGROUND

This chapter presents the concept of dependability in more detail, describes the target application (and its requirements) and the research problem.

DEPENDABILITY

Dependability is defined as the "ability of the system to deliver service that can justifiably be trusted". Different techniques to attain dependability are categorized as fault prevention, fault removal, fault forecasting, and fault tolerance. [3]. To achieve dependability, we need a combination of techniques from these four areas, although this paper focuses mostly in the fault tolerance techniques, as they are most closely related to the system architecture. The most efficient combination of methods depends from the situation and system constraints in terms of cost, complexity and effectiveness [4]. In order to achieve dependability cost-efficiently, the required level of dependability must be correctly captured in the requirements so that the system may be implemented to this correct level.

Related to dependability is the concept of *security*; both share common attributes like availability and integrity of the correct service [3]. Although security is not a key requirement in this research, it definitely should not be neglected as e.g. STUXNET worm and recent outage of the PlayStation Network have effectively demonstrated. A minimum security recommendation for supervisory systems would be to separate them from the Internet and restricting user access.

Fault-tolerance techniques in computer systems are a well-known research topic as it has been researched for several decades – e.g. Randell presents some fundamental concepts still in use today in his 1975 article about software fault tolerance [5]. Fault-tolerance methods have been developed and applied in demanding safety-critical applications like space, aviation, nuclear, railways, etc. Techniques used in these applications usually rely on replication of components, but it is important to notice the difference between replication and redundancy. Although all fault tolerance is based on use of additional resources, i.e. redundancy, this does not necessarily mean replication of software and/or hardware components.

Like most aviation and nuclear systems, ITER RH is also a safety-critical application, because of a risk of economic losses in the case of a failure in the control systems. However, ITER RH systems do not necessarily need fault tolerance based on extensive replication because there is no risk of a catastrophic failure or loss of human life. Instead, the fault tolerance could be implemented with single-version fault tolerance techniques [6]. Single version fault tolerance techniques are cost-efficient way to achieve safety with possible compromises to reliability when compared to replication-based techniques. Examples of single version fault tolerance include system structuring, atomic actions, error detection and exception handling, among others [7].

Some common replication-based reference architectures used for safety-critical applications include e.g. triple modular redundancy, recovery blocks, retry blocks and N-version-programming. With software systems, diversity of design is needed in addition to simple replication of software units, since every copy would have the same faults. Even though the development costs of diverse N-variant software are less than N times non-fault-tolerant software [8], it still presents a major cost increase for the development when compared to basic software or single version fault tolerance techniques. The problem related to redundancy is that programmers tend to make same kinds of mistakes so the different versions may have same types of mistakes, regardless of independent development. Faults in the redundant components do not need to be identical, it is sufficient that they are coexistent [9].

A solution that uses existing devices to implement a diversified fault-tolerant approach for fault-tolerant automobile steer-by-wire system is presented in [10]. The solution uses existing electronic stability control and driving-torque distribution devices as backups, to be used to maintain steering in degraded modes under component failures. The steer-by-wire system avoids structural complication and increased costs by not using diversified mechanical designs. This is an example of using graceful degradation to keep offering limited operations, or a form of limp-home-mode, in the presence of faults.

Another approach is presented in [11] using a simple well-tested and highly reliable alternative subsystem to make sure that the higher-performing but more complex primary control subsystem stays inside safe limits. If the high-performance subsystem breaches these limits, the alternative backup takes control until the

conditions become normal again. This enables the system to keep providing some level of service even in the presence of faults.

A similar idea is used in the monitor-actuator architecture, where an independent monitoring channel maintains a watch on the actuation channel looking for an indication that the control of the system should be passed to a separate safety channel that will bring the system to a safe state [12].

Finally, it should be noted that since most of faults in software systems are due to faulty requirements [4], the whole process must support the development of a dependable system. Without correct requirements, the right fault tolerance techniques cannot be chosen. Dependable systems cannot be built without appropriate combination of fault prevention, fault removal, fault forecasting, and fault tolerance techniques [4].

TARGET APPLICATION

The application for our RH control system is a teleoperated bilateral master-slave manipulator system, where the operator controls a remote manipulator working in a hazardous environment, as presented in the figures 1 and 2. As a whole, ITER RH systems aim to have one master system which is used to control several heterogeneous slave systems[1] for various maintenance tasks, provided by different subcontractors. All these systems must be able to work harmoniously, regardless of changes in other systems and technology upgrades during the ITER life cycle, expected to be several decades, so software interoperability is definitely an important research aspect. Although basics of implementing teleoperation systems are fairly well known, a fusion plant in the scale of ITER is a novel environment which sets strict requirements for control system performance when compared to what has been done before. This makes it an interesting development environment, especially for researching solutions for making the development of fault tolerant RH control systems more affordable.

One goal for our research is to develop a reference architecture and methods for fault tolerant ITER RH equipment control system, i.e. a slave-level system. An outline of the target system is shown in Fig. 2. The system consists of human system interface computers placed in a control room, including PCs running virtualization software, graphical user interface (GUI) and input device controller. Equipment controller PC running the RH control system receives orders from the HMI PC and possible higher level systems, and sends control commands to a low level control system (LLCS), which is responsible for commanding robot actuators and reading sensor data.

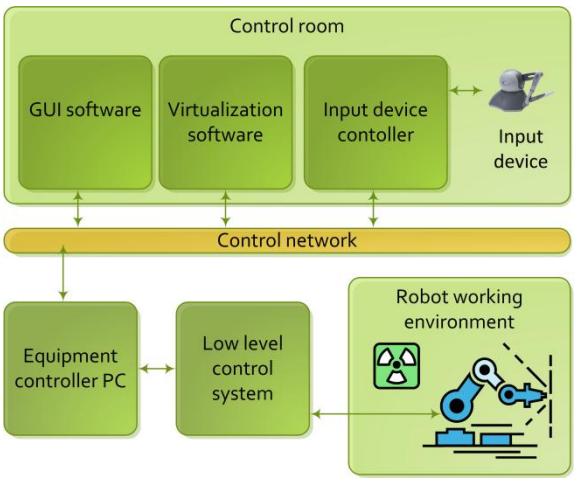

Fig. 2 – System outline.

Target robots for the RH control system in this research include a Water Hydraulic Manipulator (WHMAN) developed at Tampere University of Technology's Department of Intelligent Hydraulics and Automation (TUT/IHA) and a commercial robotic manipulator manufactured by Comau. WHMAN, shown in Fig. 3, is a multipurpose dexterous manipulator with 6 degrees of freedom.

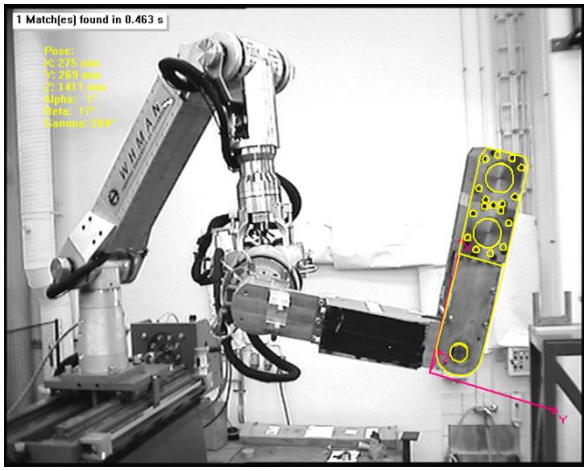

Fig. 3 – WHMAN wielding the Water Hydraulic Jack, shown in a video view with augmented reality graphics and data.

Major requirements for the system include fault tolerance, fail-safe functionality, implementation for the bilateral teleoperation and sufficient real-time performance. In order to support kinesthetic feedback, i.e. bilateral teleoperation, we need sampling frequency in the order of 1000 Hz which sets great demands for the communication between the equipment controller PC and

[1]Not to be confused with a master-slave teleoperation system, where *master* refers to the haptic device and *slave* to the teleoperated robot.

the control room PCs. Part of the research is investigating how affordable and widely-supported Ethernet-based networking supplemented with open RT protocols and communication middleware performs in a critical task like this.

To test the proof-of-concept designs for the replacement of an ITER divertor component, a full scale prototype environment, designated 'Divertor Test Platform 2' (Fig. 4), is operational at Tampere, Finland. The facility is hosted by VTT and TUT/IHA. TUT/IHA has worked with ITER RH since 1994 developing the ITER divertor maintenance, processes, tools and equipment.

Fig. 4 – Divertor Test Platform 2 at Tampere, Finland.

RESEARCH PROBLEM

The research needs to combine solutions that satisfy requirements for interoperability, dependability and control system performance in a cost-efficient manner. Modular architecture in the form of services allows addition of new components into the RH control system, and fast prototyping of new approaches. However, integrating multiple applications provides both benefits and challenges. One of the main benefits is easier sharing of information, since the data exchange has been designed to happen in a controlled way. Challenges include possible common mode failures and dependencies [13].

We have chosen an approach based on industrial PCs and real-time operating systems (RTOS) instead of embedded systems, even though embedded solutions are a more common approach for safety-critical real-time systems [14]. Embedded systems are built for purpose so code change must be done for each configuration which limits modularization and interoperability [2]. This can lead to stovepipe system antipattern, i.e. complex single-purpose 'soon-to-be-legacy' systems that consist of inter-related and tightly bound elements.

Another essential component for the system is a health monitor that is responsible for identifying faults and failures in the system software components. Health monitoring can help in isolating faults and preventing failures from propagating [13]. Since the monitor has an important role for the dependability of the system, it is vital that it be free from errors itself, similar to voting adjudicators [4] used in diversity-based fault tolerance techniques.

FRAMEWORK SOLUTIONS

Although the fundamentals of implementing teleoperation systems are well-known, the application remains demanding because of the environment, dependability and interoperability requirements. To achieve the goal cost-efficiently, the research aims to find and utilize the most useful methods from fault prevention, fault removal, fault tolerance and fault forecasting methodologies, all of which are necessary to ensure required level of dependability. These will be combined in a lean systems engineering framework that will include a reference architecture, hardware and software modules, processes and recommendations for development practices, as suggested in our earlier publication [6].

USE OF OPEN AND COMMERCIAL OFF-THE-SHELF COMPONENTS

Part of the framework consists of component reuse and use of commercial off-the-shelf (COTS) components. This approach shows some promise for achieving cost reductions in development. Especially commercial hardware components give the benefit of utilizing performance and energy efficiency of cutting edge processor technology. Use of software COTS components and component reuse have more problems related to them, as there is usually no guarantee that the components have sufficient quality for mission-critical applications and may require additional wrapping. Explosion of Ariane 5 launch system in 1996 is just one famous example of catastrophic failure caused by software reuse [15]. Instead, use of commercial or open operating systems and communication middleware has the same potential benefits as hardware, i.e. they include the latest developments in technology and have potentially better quality and less bugs than custom made software because of widespread use.

The communication bus/middleware needs to be fast and reliable. In order to fulfill timeliness requirements usually present in the machine control domain, the middleware should also provide Quality of Service (QoS) parameters. For networking Ethernet is a low cost and readily available option. However, typically it is not deterministic and proprietary Ethernet solutions have weak interoperability [16], but e.g. RTnet [17] can be used to provide hard real-time net-

working for systems using real-time Linux kernel extensions.

ARCHITECTURE

The architecture and its quality directly affect the system attributes like dependability and maintainability. Use of standard components like messaging buses can improve these attributes by providing well-tested and documented components.

A general control system architecture for machine automation was introduced, as an alternative to proprietary, specialized solutions. The platform emphasizes integrability, interoperability, maintainability and heterogeneity, satisfying many of these requirements by service-oriented design. The architecture itself is not committed to any single implementation technology.

Service orientation allows software components to be published and located, locally or over a network [18]. Services usually represent business tasks directly, making it more straightforward for the end-user. The implementation is usually hidden behind common interfaces, so it can be modified without users being affected (e.g. interoperability). The encapsulation, for its part, facilitates scalability and reuse [19]. For example, position data is read from the haptic device and then provided as a service to other interested parties.

A high level view of the architecture is shown in Fig. 5. Legacy devices are used through drivers, by both native applications and Intelligent Device converters. Services can be connected to Intelligent Devices and the Service Bus connecting these services, including Intelligent Devices, can be connected to Global Service Bus via Service Broker. On the other hand, the platform itself is recursively an Intelligent Device and thus, a service provider.

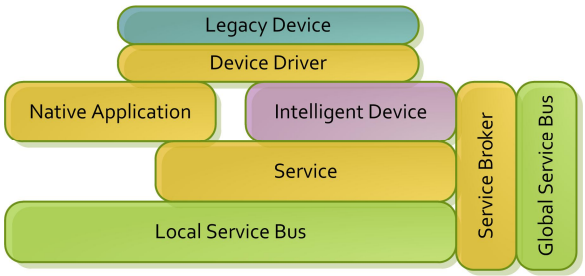

Fig. 5 – High level view of the architecture.

Special care was taken to provide a way for a control unit to be easily deployed on-top of the existing machine control. A peer-to-peer communication method between all the heterogeneous devices supporting the platform was developed, in order to avoid bottlenecks by decentralization. The networks of local services, residing inside the same physical machine, can utilize the services from other physical machines in a similar manner.

A Quality of Service (QoS) mechanism was designed, to meet the requirements for reliability. Each of the services registers through a Service Broker, acting as a gateway and keeping track of the system resources (CPU, memory and bandwidth). During the registration, a service identifies itself with the required system resources and the Broker then decides if the machine is capable of running the registrant. Spawning of the service can require spawning of other services also, and this is done in a recursive manner.

Another important functionality for the Service Broker is to provide health monitoring for the services. The Broker makes sure that the services keep operating normally. If a service crashes or starts consuming too much system resources, the Broker takes appropriate measures to preserve system stability and state consistency. For networking, the platform provides multiple QoS parameters such as real-time deadlines needed for machine control, among others.

The platform supports all the physical peripherals attached to it, such as sensors and actuators, as long as a service conversion layer is used between. The device data is encapsulated and converted to services by physical abstraction, so that the interoperability requirements are met.

In order to stop faults propagating, the safety-critical, safety-related and nonsafety-related software components should be isolated by partitioning the software [20]. The architecture provides a standard way to partition the software with the services.

SUPERVISORY SYSTEM

The maintenance work of the ITER in-vessel components must be carried out in a reliable and safe manner, with a strict schedule [21]. Supervisory systems are used to provide users with the necessary features to control a specific device, including control, indicating, and associated telemetry equipment at the master station, and all the complementary devices at the remote station [22].

This chapter describes the supervisory system and its software subsystems developed to carry out the complex RH maintenance operations at ITER and DTP2. The RH control system (Fig. 6) is divided into the supervisory system that provides the human-machine interface for the operators, RH controllers and hardware. The supervisory system is presented in the upper part of the Fig. 6 with software components listed next. The DTP2 control room is shown in Fig. 7.

Operation management system (OMS) is a software sub-system used to support operation

by planning, helping and instructing execution of RH procedures. Procedures are complete sequences of manual actions required to perform maintenance or testing operations. IHAPlanner is an OMS software developed at TUT/IHA.

Visualization system gives operators visual access to the virtual 3D model of the environment. Virtual reality software IHA3D can be used for task planning, practicing and simulation in addition to aiding the task execution with e.g. collision detection.

Viewing system (IHAView) can be used to provide live video feedback from the environment. The system is comprised of several cameras at the remote site and monitors at the control room. Augmented reality techniques can be used to support teleoperation by adding computer-generated graphics and data on top of the video, as shown in Fig. 3.

Command & control system provides a GUI that can be used to control RH equipment through its equipment controller.

Computer assisted teleoperation assists operators in teleoperation tasks that require manual manipulation of remote handling equipment. It can e.g. guide the manipulator away from pre-set force barriers to avoid collisions.

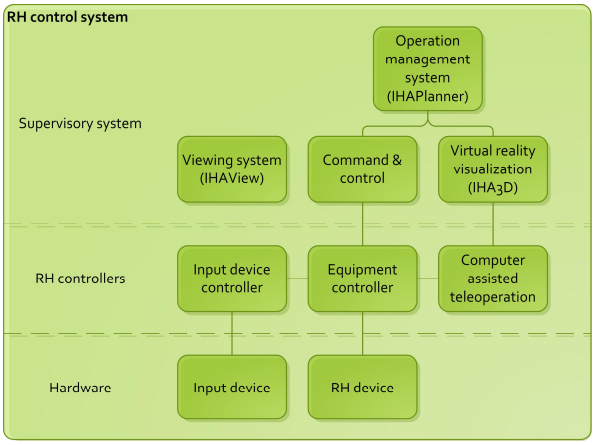

Fig. 6 – Architectural model for the RH control system.

CONCLUSIONS

This paper has presented the problem of developing dependable control system architecture for the ITER RH domain. In order to meet the interoperability requirements, an SOA for the machine control domain was developed. The architecture is based on the use of standard commercial and open components, and is aimed to provide real-time performance and QoS parameters needed in the machine control. Although this architecture gives support to key quality attributes, the development process still needs to utilize a combination of fault prevention, fault tolerance, fault removal and fault forecasting in order to ensure that the system is dependable. Fault forecasting, for example, can be implemented in the system as a service that performs fault diagnostics and reports possible problems.

Most control systems, like ITER RH, are not required to implement fail operate functionality, so some compromises can be made to achieve cost-efficiency in the development. E.g. single version fault tolerance techniques and use of off-the-shelf components are viable solutions, especially when combined with the use of fail-safe functionality or secondary high-assurance control system. Also systems based on well-tested COTS or open source communication middleware, OSs and hardware can be used to maximize interoperability, dependability and affordability.

Fig. 7 – Control room for the Divertor Test Platform 2.

ACKNOWLEDGEMENTS

This work supported by European Communities was carried within the framework of EFDA and financial support of TEKES, which are greatly acknowledged. The views and opinions expressed herein do not necessarily reflect those of European Commission or EFDA.

REFERENCES

[1] Dede, M.;& Tosunoglu, S. (2006). Fault-tolerant teleoperation systems design. Industrial Robot, 33(5), 365-372.

[2] Cucinotta, T.;Mancina, A.;Anastasi, G.;Lipari, G.;Mangeruca, L.;Checcozzo, R.;et al. (2009). A Real-Time Service-Oriented Architecture for Industrial Automation. Industrial Informatics, IEEE Transactions on, 5(3).

[3] Avizienis, A.;Laprie, J.-C.;Randell, B.;& Landwehr, C. (2004). Basic Concpets and Taxonomy of Dependable and Secure Computing. Transactions on Dependable and Secure Computing, 1(1).

[4] Pullum, L. (2001). Software Fault Tolerance Techniques and Implementation. Artech House.

[5] Randell, B. (1975). System structure for software fault tolerance. Proceedings of the international conference on Reliable software. New York: ACM.

[6] Alho, P.;& Mattila, J. (2011). Dependable Control Systems Design and Evaluation. Conference on Systems Engineering Research (CSER) 2011. Los Angeles.

[7] Torres-Pomales, W. (2000). Software Fault Tolerance: A Tutorial. NASA.

[8] Laprie, J.-C.;Arlat, J.;Beounes, C.;& Kanoun, K. (1990). Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures. Computer, 23(7), 39-51.

[9] Knight, J.;& Leveson, N. (1986). An Experimental Evaluation Of The Assumption Of Independence In Multi-Version Programming. Transactions on Software Engineering, 12, 96-109.

[10] Hayama, R.;Higashi, M.;Kawahara, S.;Nakano, S.;& Kumamoto, H. (2010). Fault-tolerant automobile steering based on diversity of steer-by-wire, braking and acceleration. Reliability Engineering and System Safety, 95, 10-17.

[11] Sha, L. (2001). Using Simplicity to Control Complexity. IEEE Software, 18(4), 20-28.

[12] Douglass, B. (2002). Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems. Addison-Wesley Professional.

[13] Krode, J.;& Romanski, G. (2007). Real-Time Operating Systems and Component Integration Considerations in Integrated Modular Avionics Systems Report. Federal Aviation Adminstration.

[14] Flammini, F. (2010). Dependability Assurance of Real-Time Embedded Control Systems. New York: Nova Science Publishers.

[15] Nuseibeh, B. (1997). Ariane 5: Who Dunnit? IEEE Software , 14(3).

[16] Moss, B. (2002). Real-time Control on Ethernet. Dedicated Systems Magazine(q2).

[17] RTnet Development Team. (2010). Referenced 12. May 2011 from RTnet: http://www.rtnet.org/index.html

[18] Ambroszkiewicz, S.;Bartyna, W.;Faderewski, M.;& Terlikowski, G. (2007). Interoperability in Open Hererogeneous Multi-robot Systems. AAAI Fall Symposium, FS-07-06, ss. 24-31. Arlington, Virginia.

[19] Jammes, F.;& Smit, H. (2005). Service-Oriented Paradigms in Industrial Automation. IEEE Transactions on Industrial Informatics, 1(1).

[20] Herrmann, D. (1999). Software Safety and Reliability. IEEE.

[21] Hahto, A.;Aha, L.;Nurminen, T.;Aaltonen, A.;Heikkilä, L.;Mattila, J.;et al. (2011). Supervisory System for DTP2 Remote Handling Operations. Fusion Engineering and Design.

[22] Ackerman, W.;& Block, W. (1992). Understanding supervisory systems. Computer Applications in Power, IEEE, 5(4), 37-40.