Dependable Control Systems Design and Evaluation

Pekka Alho
Tampere University of Technology,
Department of Intelligent Hydraulics and Automation, Finland
pekka.alho@tut.fi

Jouni Mattila
Tampere University of Technology,
Department of Intelligent Hydraulics and Automation, Finland
jouni.mattila@tut.fi

Abstract

Remote handling (RH) is a key technology in the ITER fusion reactor. The controller systems used for performing mission-critical RH operations need to be dependable, as the fundamental requirement for the ITER RH system is a fail-safe and recoverable design. Additional design challenges include interoperability with systems and platform independence during ITER life cycle. Contributions are especially needed for development of cost-effective systems engineering (SE) practices and guidelines for fault-tolerant implementation. This paper addresses the issues by presenting a survey of industrial best practices and different fault prevention, tolerance, removal and forecasting methods. Based on the results, key findings to achieve dependable and cost efficient design include development a SE framework that supports reuse of components, models and analysis results; non-redundant fault tolerance; and use of commercial off-the-shelf hardware, operating systems and communication middleware.

1 Introduction

ITER is an experimental nuclear fusion reactor, currently under construction in Cadarache, France and planned to start operations in 2018. The ITER machine operation is based on remote handling (RH) maintenance systems that enable the operators to safely, reliably and repeatedly perform robotic manipulation of items without being in contact with those items. This paper focuses on systems engineering (SE) development process of dependable RH control systems that perform mission-critical operations in this demanding environment and presents objectives of the current research. The research is part of a PhD thesis topic in the Goal Oriented Training Program on Remote Handling (GOT-RH) managed by European Fusion Development Agreement (EFDA). GOT-RH aim is to train engineers for activities to support ITER project. The research in this paper combines SE and dependability approaches to fulfil the ITER RH control system requirements in a cost-efficient manner.

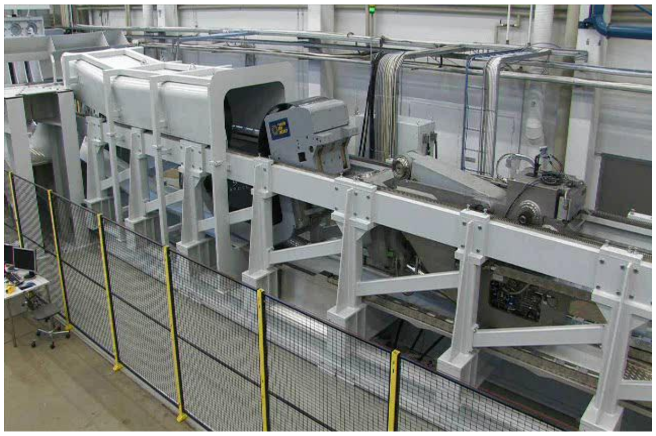

Fig. 1. Divertor Test Platform 2.

A major objective of the ITER project is to demonstrate that a fusion energy device can be maintained efficiently so that the plant availability is retained at sufficient level. During the ITER lifetime reactor components must be inspected and maintained, including replacement of the 9 tonne divertor components. Reactor operation produces high energy neutrons which are absorbed by components inside the reactor vessel, leaving them beta and gamma activated. Therefore RH has to be utilized to perform maintenance tasks instead of manual operations, as there is no human access into reactor. To test the proof-of-concept designs for the replacement of a divertor, a full scale prototype environment, designated ‘Divertor Test Platform 2’ (Fig. 1), is operational at Tampere, Finland. The facility is hosted by VTT and Tampere University of Technology, Department of Intelligent Hydraulics and Automation (TUT/IHA). TUT/IHA has worked with ITER RH since 1994 developing the ITER divertor maintenance, processes, tools and equipment.

Commonly used fault-tolerance techniques employ redundancy in order to improve reliability, but usually require significant amounts of resources. Use of these techniques is mandatory in safety-critical systems that need to keep operating regardless of failures, such as flight-control or fission reactor management (*fail-operate*), but in systems that can be guided to a safe state (*fail-safe*) the additional costs are more difficult to justify, therefore a balanced solution is needed. A key difference between ITER and fission reactors is that the energy density in ITER reactor cannot cause a catastrophic failure, but the economic losses in the case of an operation failure could be significant nevertheless. Thus the RH system is *safety-critical* and the design of a RH control system must be fail-safe or capable of operating in a *limp-home mode*, which is a form of fail-operational system.

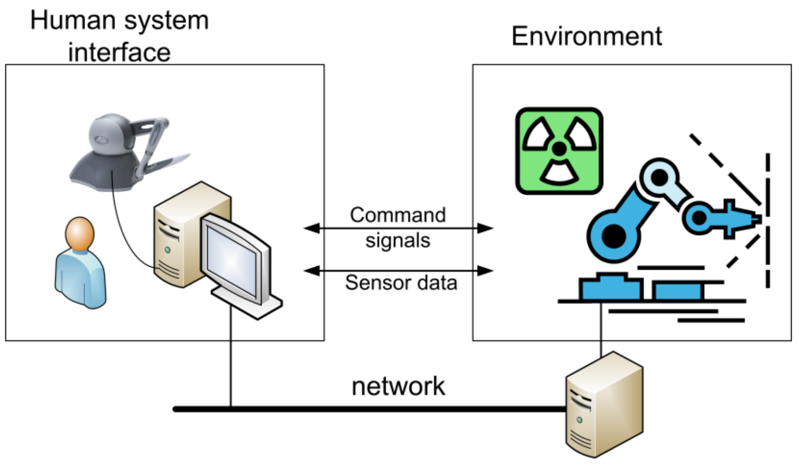

Fig. 2. Bilateral teleoperation system.

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 (see Fig. 2). The fundamentals of implementing such systems are well-known, with commercial manipulators and components available. Challenges with developing and using such systems in ITER are – in addition to aforementioned environment, dependability, etc. – need for interoperability and platform independency during ITER life cycle. As a whole, ITER RH aims to have one master system which is used to control several heterogeneous slave systems that perform various maintenance tasks, provided by different subcontractors. All these must be able to work harmoniously, regardless of changes to other systems and technology upgrades.

This paper includes a survey of industrial best practices developed by researchers in organizations like IEC, NASA, etc. and compares them against ITER requirements. Our overall target for EFDA GOT-RH is to propose a subset of a generic lean-minded SE framework to support reuse of artifacts (hardware, software, processes, models, etc.) suitable for ITER. Additionally we seek a system design that avoids extensively redundant and tightly coupled solutions. A proof-of-concept implementation of the architecture for Fig. 2 system is being currently developed.

In the next chapter we introduce research background, starting with dependability terminology and then covering related research and the research problem. In chapter 3 we examine how standards together with industry best practices and cost-efficiency affect the development process when compared to ITER requirements. Chapter 4 approaches the problem through systems development process, divided into specification, design and architecture, implementation and evaluation. Finally the conclusions are presented in chapter 5.

2 Background

In the following sections basic dependability concepts, state of the art in dependable systems and research problem are briefly reviewed.

2.1 Dependability

According to Avizienis et al. *dependability* is defined as the ability to deliver service that can justifiably be trusted. It is an umbrella term that consists of several attributes: availability, reliability, safety, integrity and maintainability. Researchers and the ITER requirements emphasize especially safety and reliability. However, all attributes need to be addressed in order to ensure delivery of the correct service – therefore the SE approach is necessary to manage all dependability-related design aspects. *Failures* are events where the delivered service deviates from the correct service. The deviations are called errors and the cause for the error is defined as a fault. It should be noted that not all errors lead to service failures – this depends from the structure and behaviour of the system. (Avizienis et al. 2004). As shown in Fig. 3, service failures can cause new faults for other systems.

$$\begin{tikzcd} \dots \arrow[r] & \text{fault} \arrow[r, "\text{activation}"] & \text{error} \arrow[r, "\text{propagation}"] & \text{failure} \arrow[r, "\text{causation}"] & \text{fault} \arrow[r] & \dots \end{tikzcd}$$

Fig. 3. Error propagation (Avizienis et al. 2004).

Applications that have dependability requirements can be categorized as fail-safe or fail-operate, depending from if the system can be brought into a safe state or whether it needs to continue operation in the presence of the faults. (Avizienis et al. 2004). Different means used to attain dependability can be categorized as *fault prevention*, *removal*, *tolerance* and *forecasting* techniques. Fault tolerance techniques, i.e. avoiding service failures in the presence of faults, can address one or more of the following stages of tolerating faults: error detection, damage assessment, and recovery and continued service. A review of different techniques can be found e.g. from NASA report Software Fault Tolerance: A Tutorial (Torres-Pomales 2000). With software (SW) systems duplication of modules replicates errors as well, so redundant components need to be diverse. Even though the development costs of N-variant software are less than N times non-fault-tolerant software (Laprie et al. 1990), it still presents a major cost increase for the development when compared to basic software or single version fault tolerance techniques.

2.2 Related work

Increase of complexity and amount of requirements for modern software systems present us the problem of how to attain and estimate the dependability of these complex systems. Another problem is related to interoperability of systems with long expected lifetimes. E.g. U.S. Navy intentionally sank the Aegis cruiser Valley Forge after 18 years of service – intended service life had been at least three decades but the integration costs of new software and weapon systems were too high (Schneider 2010). Clearly building stovepipe systems, i.e. complex single-purpose 'soon-to-be-legacy' systems that consist of inter-related and tightly bound elements, is not a viable solution. ITER will have several subcontractors providing software and has an expected life span of several decades, so integration of distributed real-time systems is a critical design factor.

In software systems service failures can create new faults via causation. To achieve no-single-points-of-failure goal in a software unit, we would need redundancy (Flammini 2010), (Hayama et al. 2010), which again increases development costs (Laprie et al. 1990). In fail-safe systems or systems using graceful degradation, structuring can be used to limit failures inside the architectural unit. If software safety, i.e. execution within system context without contributing to hazards, is considered more important than reliability, i.e. low mean time between failures, then fault tolerance techniques can concentrate on preventing catastrophic failures. Single version fault tolerance techniques are cost-efficient way to achieve safety with possible compromises to reliability when compared to multiversion techniques. Examples of single version fault tolerance include system structuring, atomic actions, error detection and exception handling, among others (Torres-Pomales 2000).

Dependable architecture designs and fault tolerant control methods tend to be too specific to be reusable, at least outside their application domain. There has been some earlier research on generic architectures for real-time dependable systems, e.g. architecture model developed by Powell et al. (Powell et al. 1999) uses fault containment to deal with faults and is based on the use of software components. The focus for our research is not in producing architectural design patterns or domain specific solutions to achieve dependability.

Some architectures have been developed to support use of commercial off-the-shelf (COTS) components (Powell et al. 1999), (Asterio et al. 2003), but reuse needs to be carefully considered to evaluate if possible cost benefits outweigh compromising effects on dependability and possible needs for additional fault tolerance. Based on experiences with the older experimental reactors, COTS components could be used to implement some parts of the control system in ITER; for example, in JET (Joint European Torus) results were positive with implementing the highly critical motion control, and integrating into a uniform control system framework. For fault tolerance JET employed a large number of error checks, which is one of the basic single-version fault tolerance techniques. For severe errors the system was put into safe-state by cutting all power, engaging brakes and opening emergency stop circuit. (Haist & Hamilton 2001).

2.3 Research problem

ITER organization promotes a standardized software-module based design approach and has an equipment controller (EC) architecture draft to improve cooperation of RH systems and higher level systems being developed by several different contractors. However, this architecture only outlines basic features. In addition to standard external safety features, like emergency stops, it does neither provide nor dictate solutions for fault tolerance.

In Fig. 4 area one presents *embedded* hard real-time systems, i.e. 'standard' solution to implementing controllers, and area two presents hard real-time systems implemented using *commercial PCs and RT operating systems* (OS) which is rarer alternative (Flammini 2010). Our purpose is to use industrial PCs to test their feasibility in implementing dependable RH systems

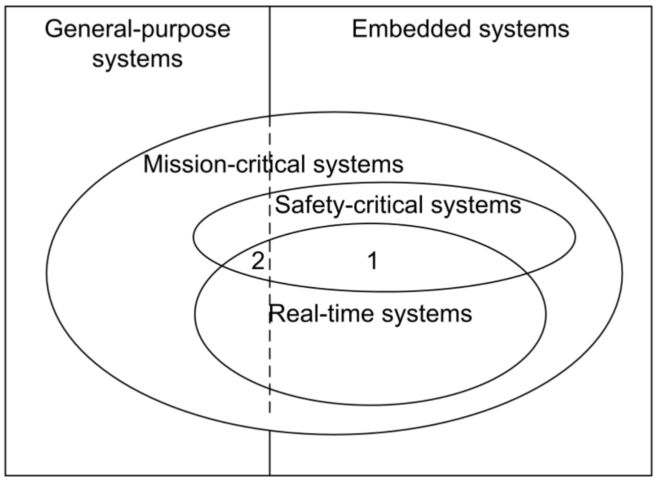

Fig. 2. A classification of critical computer systems, adapted from (Flammini 2010).

with strict performance requirements. If the use of general-purpose systems proves to be a viable alternative, it could be one way to reduce development costs for systems with dependability requirements.

Especially interesting from the perspective of using open source or commercial real-time operating systems (RTOS) in safety-critical applications is the report (Bishop et al. 2001) for Health and Safety Executive: Justifying the use of software of uncertain pedigree (SOUP) in safety-related applications. The report considers the safety assurance of SOUP in the context of IEC61508. Similar evaluation methods could be used with ITER RH systems.

In dependability research the fault-tolerant approach is often promoted over fault prevention. The reasoning behind this being that all faults cannot be prevented or removed, so it is better to concentrate on fault tolerance methods (Elder 2001); or fault prevention is shrugged off being part of general engineering (Avizienis et al. 2004). However, neglecting of fault prevention is short-sighted. First, no single mechanism can cope with all faults and anticipation of unexpected faults can increase costs. Second, the cost of finding and removing faults typically rises by development phase and finally, faulty specifications are major cause of software faults (Avizienis et al. 2004). Hence, additional research is needed on use of fault prevention to minimize the number of faults in the system as early as possible with optimal development methods. Furthermore, as scientific papers usually focus in one or two of the strategies used to achieve dependability, there is a need to bridge the gaps between the different models, methods, and tools that are used to improve the design and the operation of dependable systems, especially when being adapted to control systems (Bondavalli et al. 2001).

3 Standards and best practices

IEC 61508 is an international safety standard related to functional safety of electric/electrical/programmable systems with part three related to software requirements (IEC 2010). Most of European standards for *safety related* control systems refer to IEC 61508, if the implementation language is C or similar. ITER RH control system will include safety-related systems and some of the safety functions could be implemented by software – this would be safety-related software. Safety standards introduce concept of safety integrity level (SIL) (or performance level, PL), used to present risk reduction offered by safety functions. To achieve target SIL levels, standards include recommendations and requirements. However, especially software systems have the problem that SIL estimation is difficult because of systems complexity.

Even though this paper mostly refers to IEC61508 standard because of its suitable scope and internationality, there are a number of other well-known standards that can be used to contribute to system dependability development and evaluation. Software testing has its own standard, IEEE 829-1998 and some of the American internationally recognized standards include e.g. ISA 84 series and MIL-STD-882D.

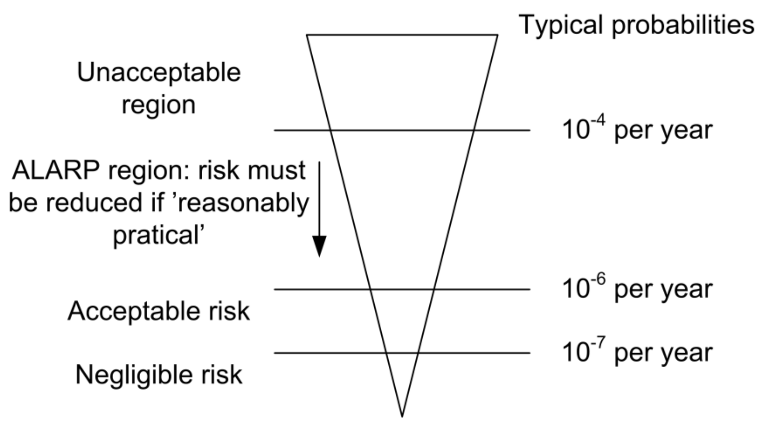

Fig. 3. As low as reasonably possible (ALARP) (Melchers 2001).

One approach to achieving a tolerable risk is 'as low as reasonably possible' (ALARP), mentioned e.g. in IEC61508 standard. If the evaluated risk is smaller than 'must be refused altogether', but larger than 'insignificant', ALARP principle together with a cost benefit assessment can be used to determine areas where risks need to be decreased, as shown in Fig. 5. Where the risks are less significant, the fewer resources are needed to be spent to reduce them and vice versa. ALARP is one of the principles used in ITER, and in nuclear project designs in general.

Component reuse and use of commercial off-the-shelf (COTS) components show some promise for achieving cost reductions in development. Especially use of commercial hardware components gives the benefit of utilizing performance and energy efficiency of cutting edge processor technology. Software component reuse has more problems related to it, as there is usually no guarantee that the components have sufficient quality for mission-critical applications and may require additional fault tolerance. Instead, use of commercial OSs has the same potential benefits as hardware, i.e. they include the latest developments in OS technology and have potentially better quality and less bugs than custom made software because of widespread use. For example, QNX Neutrino RTOS kernel has been certified to confirm to IEC61508 at SIL3 (Hobbs 2010) out of maximum level of 4. Even though implementing hard real-time systems using commercial PCs and real-time OSs is still fairly rare, this could be an interesting development path to cost-efficient and dependable systems.

Finally, design patterns are reusable general solutions that present best practice knowledge. They can and should be used to improve fault tolerance, as fault tolerance has patterns of its own – a classic example is the watch dog pattern (Hanmer 2007). However, possible design pattern use must be traceable to requirements and patterns should not be introduced without good reasons as they can add unnecessary complexity to system. A good general rule for architecture design is to keep it as simple as possible, especially for safety-critical components.

4 Development process for dependability

Development process should combine all possible methods – fault prevention, fault tolerance, fault removal and fault forecasting – to achieve sufficient level of dependability with optimal resource use by combining different methods, according to ALARP principles. As stated earlier, most studies focus on one or two methods and do their research within this limited scope, whereas efficient approach would be to combine all different approaches. The role of the development process is similar to quality assurance, i.e. reducing mistakes made by developers and ensuring product quality.

Typically V model, waterfall and other software life cycle models describe only development process (see Fig. 6). However, system life cycle also includes installation, operation, and maintenance. For safety-critical software it is important to also take these phases into consideration to ensure maintainability and interoperability, because of the higher development costs and consequently longer expected life time of the system.

The development process considerations presented in this chapter combine best practices and recommendations and discuss them in the ITER RH context. The process analysis is divided into system definition, design and architecture, implementation, and evaluation.

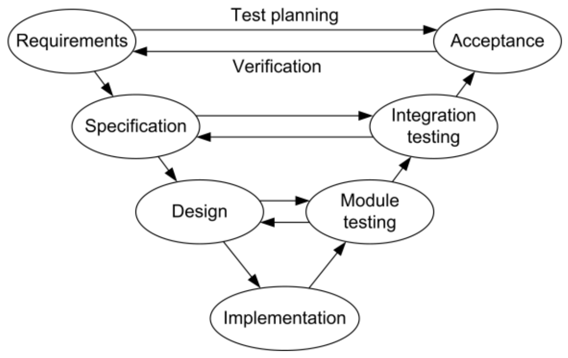

Fig. 4. Life cycle V model.

4.1 Specification

Specification, which in this case is considered to cover system analysis and definition, including hardware and software, has significant role in *fault prevention*. It is a well-known fact that errors made in the requirement specification phase of software cause more problems than coding errors (Pullum 2001). Requirements come from multiple sources and usually change as the project moves on, and the development process should offer support for this. According to Pikkarainen, use of agile methods and practices improved communication and management of requirements, features and project task dependencies (Pikkarainen 2008). However, agile methods are not necessarily suited for development of safety-critical software as such and may need additional emphasis on documentation, architectural design and traceability. IEC 61508 part 7 has a list of development methods IEC considers suitable for safety-related software.

Safety requirements are especially interesting from the dependability point of view, as they contain information about what the system is allowed and not allowed to do, as software should have indications and contraindications especially if reuse is planned for components. In most systems there are many opportunities to enhance safety, e.g. by simple value checks, but often they are not used. Safety requirements could be used to document possible values that can be used for safety checks, e.g. humidity, dust, vibration etc. (Herrmann 1999).

Hazard/risk identification and analysis should always be carried out for safety-critical systems, preceding the finalization of system requirements (Douglass 1999). Risk probability estimates can be made early, even before committing resources to hardware or software (Dunn 2002). Normal methods for risk analysis include fault modes, effects and criticality analysis (FMEA or FMECA), fault tree analysis (FTA) and risk analysis (RA). First two are qualitative and RA is quantitative (Dunn 2003). IEC61508 standard presents risk graph and hazardous event severity matrix as qualitative methods for determination of SILs.

Pre-design hazard identification and measuring reliability of existing system have some shared methods, e.g. FTA. Hazard identification requires significant resources and participation of different shareholders to gain accurate and useful results so reuse of methods and previous results should be considered for cost benefits. For ITER RH systems previously done risk analyses include e.g. manipulator FMECA and FTA for rescue of failed in-cask equipment.

4.2 Design and architecture

High-level design is the realization of quality requirements. It is also the first concrete form of the system that can be analyzed and tested. Architectural choices impact software attributes like availability, security etc., so the chosen architecture should support dependability requirements with appropriate fault tolerance techniques and patterns (Bass & Clements 2003). In addition to using these techniques, the actual system architecture should also be designed as fault-tolerant (e.g. with layering and error confinement areas). Safety-critical, safety-related and nonsafety-related software components should be isolated by partitioning the software (Herrmann 1999). Risk probability estimates for components can be made early and designs changed before actual commitments to HW and SW are made.

For ITER RH an initial version of the reference architecture has been developed, and after implementation it will be used as a testing platform for evaluating dependability of PC-based control systems and different fault tolerance methods. Possible fault tolerance components will combine patterns and COTS solutions, including QoS manager**,** network middleware, partitioning of architecture, use of heartbeat/watchdog and more. Project will make use of existing knowledge, hardware and software components etc. to maximum reuse of artifacts across different RH systems and projects.

4.3 Implementation

Implementation methods are generally dictated by development process (iterative, agile etc.), which defines the routines and support tools used in the project. Use of implementation-related methods to ensure dependability of the product is also part of fault prevention. Following good practices and programming methods can prevent generation of faults, e.g. NASA Software Safety Guidebook (NASA 2004) has comprehensively listed principles for implementation of real-time software.

4.4 Evaluation

Evaluation of software-based systems includes traditional software testing, V&V methods, and formal inspections etc., which are considered *fault removal* techniques from the dependability point of view. Another way to approach evaluation is from the *fault forecasting* standpoint by estimating or predicting reliability of the system. Component reliability is an important quality measure for system level analysis, but software reliability is hard to characterize. Post-verification reliability estimates remains a controversial issue (Torres-Pomales 2000).

In a sense, evaluation is risk control – evaluation of the software includes mishap risk assessment of the current implementation (‘is this safe enough?’) and finally acceptance. Risk assessment can be supported with data from testing, e.g. detected and corrected defects, and forecast results. Together these can be used to decide whether additional testing and fault-tolerance techniques are needed or if the product is ‘good enough’ to be finished.

Targets for evaluation include should be on all levels, including requirements, architecture, source code, software units and the complete system. In addition to evaluating the system under development, evaluation can also be done for organization before the project has been started, based on e.g. the Capability Maturity Model Integration (CMMI). Architecture evaluation methods like Architecture Tradeoff Analysis Method (ATAM) are used to determine if the architecture enables realization of key scenarios and identifying of potential risks (Grimán et al. 2007).

5 Conclusions

The research carried out in this paper has compared control system design against the industrial and scientific best practices developed for safety-critical applications. The paper presents the results phase-by-phase according to the SE process in Fig. 6. Considered viewpoints include balancing of requirements (especially dependability vs. cost) and design artefact reuse. Based on the analysis in this paper, the development process for dependable control system design and evaluation has to focus in the following:

1) Fault tolerance based on non-redundancy, i.e. single-version fault tolerance. Most of control systems are not required to be fail operate, so some compromises can be made in achieving dependability cost efficiently, like implementing fail-safe system with single version fault tolerance and reuse of components, as per ALARP principles. Standards for safety-related systems have recommendations about the use of diverse programming techniques (N-version redundancy), but e.g. in IEC61508-6 even on SIL3 they are still only 'recommended'.
2) Avoiding stovepipe systems, i.e. building large custom software, instead developing systems based on well-tested COTS or open source communication middleware, OSs and hardware to maximize interoperability, dependability and cost-efficiency. Only business-critical SW components should be custom built.
3) SE framework that covers requirements, architecture, design and evaluation to support reuse of software and hardware components, processes, models and analysis results (e.g. FMECA and FTA).

The next phase of this research consists of developing a proof-of-concept implementation of the dependable control system design for the bilateral master-slave teleoperation system presented in Fig. 2, utilizing the developed SE framework. After this the overall target is to V&V and propose a subset of generic lean SE framework, such as design models, processes, HW&SW modules, suitable for ITER RH systems.

6 References

Asterio, P., Guerra, C., Mary, C., & Rubira, F., "A fault-tolerant software architecture for COTS-based software systems." *Proceedings of the Joint European Software Engineering Conference (ESEC) and 11th SIGSOFT Symposium on the Foundations of Software Engineering (FSE-11)* ACM Press, pp. 375-382, 2003.

Avizienis, A., Laprie, J.-C., Randell, B., & Landwehr, C., "Basic concpets and taxonomy of dependable and secure computing." *Transactions on dependable and secure computing , 1* (1). 2004.

Bass, L., & Clements, P., *Software architecture in practice.* Addison Wesley, 2003.

Bondavalli, A., Fantechi, A., Latella, D., & Simoncini, L., "Design validation of embedded dependable systems." *Micro, IEEE , 21* (5). 2001.

Douglass, B., *Doing hard time: developing real-time systems using UML, objects, frameworks, and patterns.* Addison-Wesley, 1999.

Dunn, W., "Designing safety-critical computer systems." *Computer , 36* (11), pp 40-46, 2003.

Dunn, W., *Practical Design of Safety-Critical Computer Systems.* Reliability Press, 2002.

Elder, M., *Fault Tolerance in Critical Information Systems.* University of Virginia, 2001.

Flammini, F., *Dependability assurance of real-time embedded control systems.* New York: Nova Science Publishers, 2010.

Grimán, A., Pérez, M., Mendoza, L., & Méndez, E., "A method proposal for architectural reliability evaluation." *ICEIS 2007 - Proceedings of the Ninth International Conference on Enterprise Information Systems*, pp. 564-568, 2007.

Haist, B., & Hamilton, D., "A rational approach to remote handling equipment control system design." *9th ANS International Topical Meeting on Robotics and Remote Systems.* EFDA, Seattle, 2001.

Hanmer, R., *Patterns for fault tolerant software.* Wiley, 2007.

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

Herrmann, D., *Software Safety and Reliability.* IEEE, 1999.

Hobbs, C., *Using and IEC 61508-Certified RTOS Kernel for Safety-Critical Systems.* QNX, 2010.

IEC, *IEC 61508 Edition 2.0.* 2010.

Laprie, J.-C., Arlat, J., Beounes, C., & Kanoun, K., "Definition and analysis of hardware- and software-fault-tolerant architectures." *Computer , 23* (7), pp 39-51, 1990.

Melchers, R., On the ALARP approach to risk management. *Reliability Engineering & System Safety , 71* (2), 2001.

NASA, *NASA Software Safety Guidebook.* 2004.

Pikkarainen, M., *Towards a framework for improving software development process mediated with CMMI goals and agile practices.* VTT Publications, Espoo, 2008.

Powell, D., Arlat, J., Beus-Dukic, L., Bondavalli, A., Coppola, P., Fantechi, A., et al., "GUARDS: a generic upgradable architecture for real-time dependable systems." *Transactions on Parallel and Distributed Systems , 10* (6), 1999.

Pullum, L., *Software fault tolerance techniques and implementation.* Artech House, 2001.

Schneider, S., *What is Real-Time SOA.* RTI, 2010.

Torres-Pomales, W., *Software fault tolerance: a tutorial.* NASA, 2000.

7 Biography

M.Sc. **Pekka Alho** is a researcher and a doctoral candidate at TUT/IHA working with ITER RH control systems in EFDA's GOT-RH trainee program. He graduated with M.Sc. (Eng) in Oct 2009 from TUT. Earlier in 2007-2010 he worked first as research assistant and then as a researcher at Dept. of Automation Science and Engineering.

Professor, Dr. Tech. **Jouni Mattila** received M.Sc. (Eng.) in 1995 and Dr. Tech 2000 both from TUT. He is a TUT program manager in ITER-DTP2-projects on Remote Handling. He is a coordinator of EFDA GOT-RH trainee program with 10 trainees. His research interests include machine automation and preventive maintenance, and fault-tolerant control system development for advanced machines utilizing lean systems engineering framework.

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.