Integrating Requirements Authoring and Design Tools for Heterogeneous Multi-Core Embedded Systems

Using the iFEST Tool Integration Framework

*Thesis for the Global Software Engineering European Master (GSEEM).*

Ronald Wolvers
ronald.wolvers@se.abb.com

December 2012

Supervisors:
Juraj Feljan
juraj.feljan@mdh.se

Tiberiu Seceleanu
tiberiu.seceleanu@se.abb.com

Examiner MDH:
Ivica Crnkovic
ivica.crnkovic@mdh.se

Examiner VU:
Patricia Lago
patricia@cs.vu.nl
7.5. Integrating tools ................................................................. 68

8. Discussion .................................................................................. 79
   Tool access ...................................................................................... 79
   Version management ........................................................................ 79
   Web based or locally installed tool .................................................. 80
   GUI extensions ................................................................................ 80
   Identifiers ...................................................................................... 81
   Relationships ................................................................................... 82
   Metamodel modifications .............................................................. 82

9. Conclusions .................................................................................. 83

References ........................................................................................ 86

Appendices ....................................................................................... 88

Appendix A. Full Wind Turbine Requirement Specification................. 88
Abstract

In today’s practical reality there are many different tools being used in their respective phases of the system development lifecycle. Every tool employs its own underlying metamodel and these metamodels tend to vary greatly in size and complexity, making them difficult to integrate. One solution to overcome this problem is to build a tool integration framework that is based on a single, shared metamodel. The iFEST project aims to specify and develop such a tool integration framework for tools used in the development of heterogeneous and multi-core embedded systems. This framework is known as the iFEST Tool Integration Framework or iFEST IF.

The iFEST IF uses Web services based on the Open Services for Lifecycle Collaboration (OSLC) standards and specifications to make the tools within the tool chain communicate with each other. To validate the framework, an industrial case study called ‘Wind Turbine’, using several embedded systems tools, has been carried out. Tools used to design, implement and test a controller for a wind turbine have been integrated in a prototype tool chain. To expose tools’ internal data through Web services, a tool adaptor is needed. This work reports on the development of such a tool adaptor for the Requirements Management module of HP Application Lifecycle Management (ALM), one of the tools used in the Wind Turbine industrial case study. A generalization of the challenges faced while developing the tool adaptor is made. These challenges indicate that, despite having a tool integration framework, tool integration can still be a difficult task with many obstacles to overcome. Especially when tools are not developed with tool integration in mind from the start.
Abstract (Svenska)

Idag existerar det en mängd olika verktyg som kan appliceras i respektive fas i systemutvecklings livscykel. Varje verktyg använder sin egna underliggande metamodell. Dessa metamodeller kan variera avsevärt i både storlek och komplexitet, vilket gör dem svåra att integrera. En lösning på detta problem är att bygga ett ramverk för verktygsintegration som baseras på en enda, gemensam metamodell.

iFEST-projektets mål är att specificera och utveckla ett ramverk för verktygsintegration för verktyg som används i utvecklingen av heterogena och multi-core inbyggda system. Detta ramverk benämns iFEST Tool Integration Framework eller iFEST IF.

iFEST IF använder webbtjänster baserade på en standard som kallas OSCL, Open Services for Lifecycle Collaboration samt specifikationer som gör att verktygen i verktygskedjan kan kommunicera med varandra. För att validera ramverket har en fallstudie vid namn ”Wind Turbine” gjorts med flertal inbyggda systemverktyg. Verktyg som används för att designa, implementera och testa en styrenhet för vindturbiner har integrerats i prototyp av en verktygskedja. För att bearbeta och behandla intern data genom webbtjänster behövs en verktygsadapter. Detta arbete redogör utvecklingen av en verktygsadapter för kravhanteringsmodulen HP Application Lifecycle Management (ALM), ett av de verktyg som använts i fallstudien av vindturbinen. En generalisering av de utmaningar som uppstod under utvecklingen av verktygsadaptern har genomförts. Dessa utmaningar indikerar att, trots att det finns ett ramverk för verktygsintegration så är verktygsintegration fortfarande vara en svår uppgift att få bukt med. Detta gäller särskilt när verktyg inte är utvecklade med hänsyn till verktygsintegration från början.

Keywords

Tool integration, metamodeling, SOA, OSLC, Web services, multi-core embedded systems, hardware/software co-design, HP-ALM, MATLAB Simulink.
Acknowledgements

I would like to first and foremost thank my supervisor at ABB Corporate Research Sweden, Tiberiu Seceleanu, for his continued support and feedback. I would like to extend my gratitude to my supervisor at Mälardalen Högskola, Juraj Feljan, for his valuable feedback on the thesis. Furthermore, I would like to thank Patricia Lago for providing me the opportunity to study abroad as a student in GSEEM program and Ivica Crnkovic for recommending me to ABB. I would like to thank the people that I closely worked with in the iFEST project and the AN/ICE department in ABB. Special thanks to Frédéric Loiret from KTH for his invaluable support during the development of the tool adaptor. I would like to thank my friends, both my friends in the Netherlands and the many new friends that I made during my stay in Sweden. Lastly, I would like to thank my family and especially my parents for their everlasting patience and support.

Acronyms

iFEST Industrial Framework for Embedded Systems Tools
OSLC Open Services for Lifecycle Collaboration
SOA Service Oriented Architecture
HTTP Hyper Text Transfer Protocol
REST Representational State Transfer
CRUD Create Read Update Delete
XML eXtensible Markup Language
RDF Resource Description Framework
JSON Java Simple Object Notation
URI Uniform Resource Identifier
HP-ALM Hewlett-Packard Application Lifecycle Management
EA Enterprise Architect
SDK Software Development Kit
1. Background

The *industrial Framework for Embedded Systems Tools* (iFEST) project is a project aimed at integrating tools used in the development of multi-core, heterogeneous embedded systems. iFEST was launched by the ARTEMIS Joint Undertaking (ARTEMIS JU), which is a public-private partnership between the European Commission, 22 ARTEMIS member states and more than 200 organizations that are members of the ARTEMIS Industry Association.

As part of ABB Corporate Research’s (ABB CRC) participation in iFEST, an industrial case study that will integrate a number of embedded systems tools used in the development of a controller for a wind turbine has been initiated. This industrial case study is referred to as the Wind Turbine industrial case study. The controller for this wind turbine aims to achieve optimal parameters for generating power, based on a number of input variables. Examples include the measurement of variables such as wind direction and velocity and the control of parameters such as the pitch of the propeller blades and the direction of the nacelle.

In the Wind Turbine industrial case study, an array of different tools is employed. The most prominent being HP Application Lifecycle Management (HP-ALM) for Requirements Engineering & Analysis (RE&A) and Verification & Validation (V&V), MATLAB Simulink, developed by MathWorks, for Design & Implementation (D&I) and Enterprise Architect (EA) for Design & Implementation also. To integrate these tools, a so-called framework tool integration approach is taken. This means that all the tools are interconnected through a common framework, which specifies the protocols and formats the tools should use to communicate to one another. The framework that is used in the Wind Turbine industrial case study is known as the iFEST Tool Integration Framework (iFEST IF).

The iFEST IF is a prototype tool integration framework based on the Open Services for Lifecycle Collaboration (OSLC) specifications. The central idea of the OSLC specifications is to interconnect different tools through a Service Oriented Architecture (SOA) and Web services, using common protocols and data formats. iFEST aims to adopt OSLC specifications as much as possible, but may deviate whenever necessary.

In most cases, tools can not be connected to a framework out-of-the-box. Every tool uses its own data formats, supports its own range of protocols, provides its own authentication schemes and has its own mechanisms for storing data. These differences constitute the most fundamental problems that tool integration approaches aim to solve. To interconnect tools despite the differences, a software component known as a tool adaptor is written. Every tool is connected to the framework by its own tool adaptor, while all the tool adaptors use common data formats and communication protocols.
This work reports on the development effort for one such tool adaptor. This adaptor was written with the purpose of connecting the Requirements Management module of HP-ALM to other tools in the Wind Turbine industrial case study. The challenges faced and the solutions invented to overcome them are discussed in great detail. Furthermore, an attempt is made answer to question of whether tool integration was successful in the Wind Turbine industrial case study and how it compares against other tool integration efforts.
2. Introduction

In a typical system development project many different system development tools with highly diverging purposes are used. Examples of system development tools, or simply tools, are requirements authoring tools, but also system design and testing tools are very common. Ever since the heyday of computer-aided design (CAD) numerous new types of tools have been invented. Among the most prominent reasons to use these tools instead of developing a system ‘manually’ is to reduce a developer’s cognitive load and henceforth produce systems of a higher overall quality. The common conception is that a developer should not be burdened with tedious tasks that could be automated with a relatively low effort, nor should they be too concerned with the precise form of the inputs that a machine expects. Tool usage should be intuitive and stimulate rather than hinder the creative process. However, an essential aspect of any project is the successful transition between lifecycle phases. As we are employing different tools in different phases, so should we transition between the respective tools. Tool integration however is a non-trivial task. The subject has been receiving a lot of attention in computer science research for over two decades and is nowadays still very prominent.

In his work dating back to the early 90’s, Anthony I. Wasserman [1] already indicated that there is a need for standards on tool integration, where tool vendors agree on common protocols and data formats. This is a type of framework integration as opposed to point-to-point integration. In point-to-point integration the mechanisms are specific to pairs of tools and typically multiple mechanisms are used in tool chains consisting of more than two different tools.

Open Services for Lifecycle Collaboration (OSLC) is one such framework integration effort that aims to standardize the way that software lifecycle tools can share data [2]. It provides standards for the different types of data that tools can expose. Examples include data related to requirements, defects, test cases and so on. Although not being the only effort as such, OSLC is rapidly maturing and multiple efforts to implement and validate its specifications are made.

The iFEST (industrial Framework for Embedded Systems Tools) project [3] is one project that implements (part of) the OSLC specifications. Its main goal is to specify and develop a tool integration framework for the development of heterogeneous and multi-core embedded systems. The framework will use OSLC specifications as much as possible, but may also deviate in case specifications are deemed inadequate for a certain application. OSLC is mainly focused on software development tools, whereas the iFEST IF will also integrate hardware tools, or tools that deal with both hard- and software at the same time. Because of the additional complexity of the tools that deal with hardware, a deviation from OSLC is anticipated.

The highly specialized tools the iFEST IF is targeting are integrated by means of tool adaptors. These tool adaptors are relatively small pieces of software that integrate a specific tool into the framework through OSLC based Web services and, if necessary, interfaces based on other technologies. One such tool adaptor is the adaptor for the Requirements Management module of HP Application Lifecycle Management [4] or HP-ALM, which is essentially a collection of different tools – or tool suite - aimed at managing the entire software lifecycle.
As part of ABB’s participation in the iFEST project, an industrial case study called the Wind Turbine industrial case study was performed. The goal of the industrial case study is to validate and verify the iFEST IF and its tool adaptors. The tool adaptor for HP-ALM was developed by the author and has been used in different integration scenarios within the case.

This work reports on the challenges that were experienced while developing this adaptor. But to not restrict the document to a largely technical story, the HP-ALM/OSLC tool adaptor and the Wind Turbine industrial case study will be placed into their broader context. We take a critical look at whether or not the Wind Turbine industrial case study has achieved successful tool integration and how iFEST compares to other tool integration efforts.

Starting off with discussing why we desire tools to be integrated, Chapter 3 takes a critical look at tool integration and its implications. We explore the different definitions of tool integration that exist in the literature and ask ourselves why tool integration is such a challenging topic. We ask ourselves what the general issues are, after which we move on to discuss tool integration efforts other than the iFEST IF.

Chapter 4 explicates the iFEST project and the iFEST IF. Discussed are the general project goals, the key concepts and architecture of the iFEST IF and the difference between iFEST and other tool integration efforts.

Chapter 5 discusses OSLC in a more general fashion, shedding light on its general concepts, its goals and its constraints and benefits.

Chapter 6 discusses the HP-ALM/OSLC adaptor that was developed. Discussed are the adaptor architecture, general development challenges faced and deviations made from original specifications. Also provided is a walk through of how interactions are performed with the actual implementation of the adaptor in the context of the scenarios defined within the Wind Turbine industrial case study.

Chapter 7 is about the Wind Turbine industrial case study, in which the HP-ALM/OSLC and other tool adaptors were put to actual use in a complete tool chain. A walkthrough of the details of the industrial case study is taken, including but not limited to the requirements and design of the Wind Turbine system, the tool adaptor requirements and the different tools used in the tool chain. We discuss three distinct tool integrations scenarios and their outcomes.

In chapter 8 we make an attempt to generalize the challenges faced during the development of the tool adaptor. We show that, despite have a tool integration framework, tool integration is still a difficult task with many obstacles to overcome. Especially when tools are not developed with tool integration in mind from the start.

In chapter 9 we summarize the most important points made in this work.
3. Tool integration

In a typical system development process, different tools that address and support different phases of the system development process are being used. Together, these tools form what is referred to as a tool chain. The tools allow their users to manipulate data related to either the system that is being developed or the process orchestrating the development effort.

For instance, there typically will be a tool that is used to capture and manage requirements. The data that can be manipulated through the use of this tool either represents actual requirements or is strongly related to them. The user of such a tool is able to create, modify and delete requirements through a user interface (UI) and is being saved the burden of managing the requirements by hand.

There might be an architectural tool that allows a software designer to crystallize the requirements into an actual software system, by mapping the requirements onto components. Similar to the first tool, the user of this tool is able to manipulate data that represents either components or data that is strongly related to components.

There also might be a tool that assists in the implementation of the system; a tool that not only compiles but also analyzes the source code produced by the developers, and provides them with valuable information. Not only do tools make it easier for their users to manage artifacts related to the system or the development process, more advanced tools are able to perform validations and provide the user with improvement suggestions.

But the aforementioned tools are just the bare minimum. Based on the process that the developing organization defined, or the methodology they are using, there might be a tool to automatically generate test cases, and there might exist a process management tool that oversees the entire workflow. There might be a tool that assesses system performance. There might be a tool that keeps track of changes to the system. Many different flavors of tools varying degrees of complexity exist.

There are many different tools with different purposes used by different people in different organizations, using different processes. But for the system development process to be effective - to deliver the right system within a set time and budget - we need some guarantee on tool consistency. That is, every tool used should maintain data that represents the exact same system being developed using the exact same process. For instance, the system design should match the initial requirements, the implementation should match the design and the implementation should be validated using an appropriate set of test cases.
But typically, consistency among tools is maintained manually. When the requirements are deemed to be sufficiently captured, a requirements document is passed on to a software architect who is presumed to produce a matching system design. And even though the software architect may be an expert, it poses a risk to have a sometimes tedious and time-consuming step be performed by a human intermediary. Especially when this step could be - at least partially - automated. Without guarantees on tool consistency, the effectiveness of the developing organization suffers, because how can it be sure to be building the right system when the system produced potentially does not match the initial requirements? The need to have tools communicate independently seems apparent. But as is the nature of software, the communication protocols and mechanisms should be defined formally. Only when this prerequisite is met can tools communicate automatically, which is the very first step towards tool integration.

But what exactly are the goals of tool integration and how do we know we achieved them? What we need is a definition precise enough to provide us with a framework that allows us to assess to what degree we achieved the goals. Using appropriate metrics we can then establish how well integrated our tools are. Finally, this information can be used to more tightly integrate tools and in doing so improve the entire system development process.

In the upcoming sections, we take a look at the different definitions of tool integration that exist in the literature (3.1) and tool integration efforts other than iFEST for which material is publicly available (3.2). In the third and final section of this chapter we explore the relationship between tool integration and metamodeling, which constitutes an import subfield within contemporary tool integration research (3.3).
3.1. Definitions of tool integration

There is more than one way to think of tool integration. There are many different definitions, taxonomies and classifications that exist in the relevant literature.

One of the earliest work reporting on the subject matter is a piece by Anthony I. Wasserman [1]. Dating back to the early 90’s, in his paper Wasserman provides five types of fundamental issues related to tool integration that must be addressed. We discuss them briefly.

Platform integration being the most fundamental, it defines whether or not tools are able to simply communicate. Tools that do not share a common communication platform cannot be integrated. It is a minimum requirement for integration and is therefore not even considered by some authors.

Presentation integration is when tools share a common “look and feel” from the user’s perspective. A contemporary example is the Microsoft Office suite. Every tool in the suite has a similar user interface and exerts similar behavior, despite handling different types of artifacts.

Data integration means that tools are able to share data. This implies that the tools share a common metamodel, or are able to interpret each other’s metamodels and perform transformations between them.

Control integration then refers to tools being able to influence each other’s behavior. Examples include manipulating the tool’s user interface – if any- ranging to starting up and shutting down another tool.

Process integration, finally, is achieved when there is a process management tool present that supports different types of well-defined system development processes and their accompanying tools. The process management tool is used to monitor and manage the other tools in the tool chain. For example, a process management tool might display how many test cases were successfully run for a given day, or how many requirements have been implemented.

Wasserman moves on to define three dimensions of tool integration; data integration, control integration and presentation integration. Tools should agree on at least one of these dimensions to achieve minimal tool integration. Effective tool integration is achieved when tools agree on all three dimensions. These dimensions and examples for every dimension are depicted in Figure 1 below, taken from Wasserman’s work. The example tools T1 and T2 in this diagram cannot be integrated because they do not agree on any one of tool integration dimensions. That is, they do not store their data in the same way; T1 uses shared files whereas T2 uses an object base. They did not standardize the way they present data on the same level; T2 standardized the entire “look and feel”, whereas in T1 there is only a standard window system. Finally, the tools are not controlled in the same way; T1 uses explicit messaging, whereas T2 employs a message server. Surely, the values the dimensions can take are many more in reality, but the same challenge remains; tool should agree on at least one dimension in order to be integrated minimally.
Wasserman provides a workable, albeit somewhat simplistic taxonomy, but he does not provide a strict definition of tool integration. He states that users of tools desire those tools to work together to fully support the user’s design and development process. The goal of tool integration is thus to produce complete environments that support the entire system development life cycle.

The problems Wasserman points out in his work are still largely present in today’s practical reality of tool integration. He already indicated over 20 years ago that there is a need for standards for tool integration, meaning that tool vendors agree on mechanisms for data integration, presentation integration and control integration through open interfaces and data formats. This is a type of framework integration as opposed to point-to-point integration, where the mechanisms are specific to the different tool pairs and the tool chain is generally consisting of more than one of these mechanisms. The framework then consists of the aforementioned interfaces and data formats. Furthermore, he stipulates that tools should be developed with tool integration strategies in mind, rather than tool integration being an afterthought.

Thomas & Nejmeh [5] provide a stricter definition of tool integration. Instead of relating integration to the entire environment of tools, they define integration as a property of a tool’s relationships with other elements in the environment. Examples being other tools and the platform. Tool integration is about the extent to which tools agree with the ultimate goal being to make the entire software process more effective.
This agreement could be on data formats, presentation aspects, etc. A framework that focuses on defining integration and that is independent of the implementation and mechanisms used to achieve integration is proposed.

Thomas & Nejmeh use Wasserman’s integration issues to define types of tool relationships, except platform integration, as it is so fundamental that other types of integration would not be possible without it. The types of relationships and their properties are shown in Figure 2, taken from Thomas & Nejmeh’s work.

![Figure 2. Tool relationships and their properties](image)

In this entity-relationship diagram, the questions posed for every relationship property embody both the goals of integration and a way to determine whether that particular goal has been achieved. This follows Basili & Weiss’ [6] approach for metrics development, in which they advocate identifying goals, questions that refine the goals and quantifiable metrics that can provide the answer to the questions. Thomas & Nejmeh’s framework does not provide quantifiable integration metrics.

Defining such metrics however is not a straightforward task. Looking at presentation integration and its relevant questions (Figure 2) for instance, it is apparent that the answers will be subjective and therefore irrelevant to quantify. We can simply not assign any number to how similar two different tool metaphors are.
Also, the framework is merely about binary relationships. That is, relationships between pairs of tools. It does not consider integration of many tools nor the properties of such a relationship between three or more tools. The question then remains whether or not this type of integration would make the software process more effective per se.

Framework integration does not necessarily make the software process more effective. However, as Thomas & Nejmeh state, there are two points of view when looking at tool integration. One is that of the environment user and the other is that of the environment builder. In the eyes of the environment user, environment A consisting of the exact same tools as environment B integrated to the same degree, but A using point-to-point integration and B using framework integration are perceived as the exact same environments.

However, the environment builders will be more efficient and effective in integrating the tools. For every type of tool, they know what types of data they should expose and what types of data they need. What’s more, the limit to the number of interconnections between tools is limited to N, where N is the number of tools in the environment, as opposed to N(N-1)/2. This because all tools connect to the framework instead of each other. Therefore, the network of interconnections is a connected graph (Figure 4) rather than a complete graph (Figure 3).

Figure 3. Point-to-point tool integration
Thus, at its very minimum, framework integration will reduce the number of interconnections between tools and consequently the amount of development effort to achieve them. There is however more merit to framework integration than simply the amount of development effort it requires, as we will see later on in this work.
3.2. Related work

In this section we discuss other tool integration efforts as found in the relevant tool integration literature. We take a look at the problems that are faced and attempt to establish a generalization. Note however that we only consider a certain class of tool integration efforts. To be more precise, we only consider efforts in which tools developed by different tool vendors are integrated together. The most prominent reason being that for a given situation we would like to be able to select the most adequate tool rather than the one that integrates with the other tools merely because it was developed by the same vendor. Thus circumventing vendor lock-in, i.e. having freedom of choice independent from the vendor, which is one of iFEST’s major goals.

3.2.1. Merlin ToolChain

The first tool integration effort involves a ‘configurable’ tool integration solution referred to as the Merlin ToolChain, as discussed in a paper by Pesola et al [7]. The Merlin ToolChain is, like iFEST, an effort aimed at the integration of embedded systems tools through framework integration. The Merlin ToolChain divides tools up into four distinct categories. These are Requirements Management (RM), Test Management (TM), Configuration Management (CM) and Project Management (PM) tools.

![Image of Merlin ToolChain tool categories](image)

Figure 5. Merlin ToolChain tool categories [7]

The Merlin ToolChain is based on Eclipse plug-ins that use the Merlin ‘master’ plug-in to connect to each other. The tools are integrated together by providing an interface compliant with one of the four interface definitions, implemented as an Eclipse plug-in. Every artifact in every tool has a unique identifier that allows for distinction.
A traceability plug-in uses the unique identifiers to create traces and stores them in a traceability database. The central idea of the architecture is that tools should merely implement the plug-in interface and there is no need for the master plug-in to know how a tool retrieves the data it exposes.

The paper moves on to discuss a prototype implementation of the framework, but does not discuss the specifics of the tool interfaces nor does it explicate what the problems faced in implementing them were. The work acknowledges that tools should be categorized and metamodels defined, like the ones in Figure 5. However, the metamodels are extremely simple and what is more, in the face of more complex metamodels the implementation of writing tool adaptors can be daunting, as we shall see in our discussion of the Wind Turbine industrial case study.
3.2.2. Fujaba Tool Suite

The second tool integration effort, the Fujaba Tool Suite, takes a deeper approach [8]. The Fujaba Tool Suite and its Fujaba Approach that integrates tools into the tool suite aim to achieve data integration at the metamodel level. That is, the tools are not just communicating by for instance sharing a GUI or controlling each other’s functionalities, they communicate by speaking each other’s language or speaking a common language. We refer to this tool language as a metamodel. The metamodel is essentially the structure the models that a tool’s user is allowed to manipulate adheres to. Metamodels can for instance dictate the kind of attributes a requirement or component should have and what possible relationships to other artifacts it can have.

Ideally, we would like for the data we manipulate using tools to be consistent. That is, the data governed by the different tools in the tool chain retains the same picture of the system we are developing, or put more precisely; is semantically consistent. Not only does the process of assuring consistency for a given set of tools induce significant overhead, it makes it much more difficult to switch tools in the face of changing tool requirements. We discuss how the Fujaba Tool Suite and Approach dealt with these problems. For a general and more extensive discussion of tool integration and metamodeling, refer to section 3.3.

The attentive reader might be wondering why not all tool integration efforts deal with metamodel integration. After all, metamodel integration is the only type of integration that would potentially integrate tools so tightly they simply do not allow inconsistencies to exist in the first place. The reason for this is that for metamodel integration to be possible, tools need to have facilities in place that allow access to their metamodels. The Fujaba Approach not only requires tools to provide access to their metamodels, but requires those metamodels to be extendable. This however is a rather advanced feature that only very recent tools possess, as having a flexible metamodel by implication requires the tool itself to be flexible also.

The Fujaba Approach and Tool Suite are focused around the tools being represented by plug-ins. Being a common mechanism for adding third-party components to existing products, the central idea is that dependencies are unidirectional. That is, a plug-in is dependent on the product or environment it extends, but not vice versa. Hence, the product can be compiled without any plug-ins that might exist. All tools are integrated into the Tool Suite using the plug-ins mechanism. Two general plug-in based integration scenarios are considered in Fujaba:

![Figure 7. Tool B extending Tool A. [8]](image-url)
In Figure 7, Tool B extends Tool A, implying there is a relationship requiring Tool B to be consistent with Tool A. Generally however, consistency is a bidirectional relationship. That is to say, Tool A is required to be consistent with Tool B but the inverse is also true. Especially if one considers an iterative development methodology, this surely is the case.

![Figure 8. A third Tool 3 extending both Tool 1 and Tool 2](image)

In Figure 8, Tools 1 and 2 are integrated by means of a third tool, Tool 3. This additional tool maintains the links between the two disjoint metamodels in Tools 1 and 2. Tool 3 requires both tools to compile, but this dependency does not hold the other way around.

A fundamental problem that arises for frameworks like this one (and is discussed in great detail in the work about Fujaba [8]) is the possibility to have bidirectional associations. Two classes in different metamodels may point to each other to indicate some bidirectional relationship, but this introduces a mutual dependency between the metamodels. As a result the extendable framework approach breaks. There are different ways however of having bidirectional relationships without having classes directly point to each other. One is to use attributes instead of references, but retaining consistency quickly becomes a problem. Even if we would somehow allow references, retaining consistency is still a problem because there is always an inverse reference that should be updated accordingly. To solve this latter problem, the work about Fujaba proposes to use public access methods that handle this updating with sophisticated mechanisms. However, the paper also offers solutions that eliminate explicit references completely.

These solutions are captured the Meta-Model Extension design pattern for metamodel extensions (Figure 7) and the Meta-Model Integration design pattern for metamodel integration (Figure 8) respectively. Both patterns maintain flexibility, because they ensure compile-time independence. However, because of this flexibility, the extensions to the existing models are quite clumsy. There cannot be bidirectional relationships for instance in the model extensions, because they introduce mutual compile-time dependencies. The paper’s proposal is to encapsulate the logic of extending the existing metamodels completely, as to ensure that they are correctly updated. While a fair approach, the downside is that for complex metamodel extensions, the logic that needs to be encapsulated is also complex, hence a burden is put on the developer.
3.3. Tool integration & metamodeling

We said that the goal of tool data integration is tool consistency. An entire subfield of research within the field tool integration is that of metamodeling and metamodel transformations. At the heart of this research is the hypothesis that every system development tool is based on a certain metamodel and that if we can somehow manage to integrate these metamodels we achieve tool consistency and therefore successful (data) integration. This can either be done by finding a way to transform models conforming to one metamodel to models conforming to the other, or defining a common metamodel. Metamodeling and its relation to tool integration are frequent topics in contemporary research, especially after the Object Management Group (OMG) issued a Request For Proposal (RFP) concerning Queries/Views/Transformations (QVT) in 2002.

Also, model transformations are an essential part of the iFEST IF. In the first of integration scenarios discussed in section 7.5 for instance, a tool called MDWorkbench [9] performs model transformations from HP-ALM to MATLAB Simulink [10] and vice versa. This section provides the reader with a basic understanding of the field.

Tool consistency is when tools agree on their view of the system and/or the development process. Or put more precisely, when the data they manipulate represents the same system or development process. The data in two different tools may be mutually exclusive, but where it overlaps, it should not be conflicting. For instance, one tool strictly managing functional requirements and another strictly managing non-functional requirements can be said to be consistent because their data does not overlap and therefore cannot be conflicting. However, two tools managing functional requirements are only said to be consistent if requirements do not contradict each other. Even if there are some requirements exclusive to each tool.

Moreover, the tools that are employed in the subsequent phase of the system development process should be consistent with regard to the tools in the previous one. However, as is the nature of system development, here the requirements on tool consistency are more strict. All data managed in the previous phase should be able to be traced to data in the next one. As an example, all software components in the design phase together should implement all the requirements in the requirements phase. We cannot allow for requirements that are not mapped to a component nor for components that do not implement any particular requirement. This type of consistency we will refer to as vertical consistency, whereas horizontal consistency involves tools used in the same phase (Figure 9). That being, if for example a requirements authoring tool is replaced by another, it should retain the same requirements model.
Perfect tool consistency in practice however, is difficult to achieve. For now disregarding issues such as the availability of APIs and appropriate documentation for tools, there is one problem more inherent to data integration. And that is that different tools rely on different metamodels. The data the tools manipulate is structured according to metamodel(s) and in order for successful data integration to be established, what we need is to integrate metadata. Adapted from a paper by Laurence Tratt [11] about model transformations and tool integration, let us sketch an example scenario to show why data integration can be so challenging.
In Figure 10, we defined metamodels representing two different modeling languages for simple class diagrams. The building blocks of class diagrams are packages, classes and associations. All of these are diagram elements. Figure 10 (a) is a modeling language that allows package inheritance while the modeling language in Figure 10 (b) does not. Package inheritance makes it possible to create more elaborate diagrams with less clutter. A package inherits all the classes from its parent package. Figure 11 (a) represents an example model of a company created using modeling language ML1 while Figure 11 (b) is the company model in modeling language ML2.

Figure 10. Modeling languages with and without package inheritance [11]

Figure 11. Company model using (a) modeling language ML1 and (b) modeling language ML2 [11]
The company models in Figure 11 were created by hand. We intuitively merged the three packages Sales, Stock and Company into a single package. It would be much more convenient if we could somehow automate the transformation from one model to the other.

The first step is to make the models machine readable. One option is to use XML technology, describing the model textually. We could then try and build a program that parses the XML representation of the model in ML1 and produces a model in ML2. In the case of Figure 11, this would mean that the program looks for elements from inherited packages in ML1 models and brings them into the children directly. Under the assumption of imperative programming, there are two immediate problems however; if a package is specialized by two other packages then two copies of the classes in the parents will appear in the target model. Also, cycles in the model created by associations between two classes will cause the program to loop indefinitely. Surely, these problems are solvable, but only to a certain degree. To deal with cycles the program needs to remember which elements it has already seen, but for an arbitrarily complex model, the extra lines of code needed are arbitrarily complex.

It is apparent that in order to cope with the problems such as the ones just described, we need to keep adding an extensive amount of logic to the transformation program, for instance to keep track of which elements we have already transformed. This quickly leads to our program source code getting swamped and hard to comprehend. Tratt [11] goes as far as to argue that it is not sensible to write model transformations using a standard programming language, object-oriented (OO) or otherwise. Furthermore, so far we have only dealt with \textit{unidirectional stateless} transformations. We are only able to transform from models in ML1 to ML2 but not vice versa and the transformations is not aware of which element in ML2 models map to which elements in ML1 models. In the general case however, a transformation is followed by small changes to the target model that eventually are fed back to the source model. This cycle keeps revolving until both models are deemed sufficient. Propagating changes back from the target model into the source model, while at the same time keeping track of which element maps to which element in both models, is a \textit{stateful bidirectional} transformation. It is stateful because we keep track of elements mappings through \textit{tracing information}. Retaining tracing information separately might intuitively be deemed unnecessary. Because arguably, one might say that a specific and unique source model should transform into a single specific and unique target model. Therefore, the reverse transformation of the target model into the source model should result in the original source model. In practice however, this \textit{bijectiveness} of a model transformation usually does not hold.

What we need to look for, is some way to declare and perform transformations of models conforming to one metamodel to models conforming to another metamodel, but not using conventional programming paradigms. There are several different initiatives being undertaken, especially after the Object Management Group (OMG) issued a Request For Proposal (RFP) concerning Queries/Views/Transformations (QVT) in 2002. The goal of the RFP was to find a suitable standard for model transformations that fits within OMG’s already existing spectrum of standards related to Model-Driven Architecture (MDA) [12].
In the MDA approach to building software systems, the central idea is to keep models that relate to a system’s design separate from the models that represent the implementation. The former, Platform-Independent Models (PIM) are created using OMG standards such as the Unified Modeling Language (UML) and the Meta Object Facility (MOF) metamodeling language. These models can then be realized by any Platform-Specific Model (PSM) and implementation technology, such as CORBA, Java, XMI/XML, etc.

QVT is also one of OMG’s standards for Model Driven Architecture. QVT is a language that can define transformations upon models, in a declarative rather than an imperative way. The central idea is that in QVT one defines transformations from a PIM into various PSMs. We can argue that during development, only the PIM should ever be modified, essentially treating the transformation as a compilation. In practice however, it is necessary for developers to modify both source and target models. As a consequence, changes should be able to be propagated in both directions. Bidirectional transformations are a key requirement on QVT.

The QVT standard describes three languages for transformations: The Operational, Relations and Core languages. In QVT-Operational, it is possible to define unidirectional model transformations in an imperative way. Bidirectionality is achieved through manual consistency checking. In QVT-Relations we can define the relation between two models in a declarative way, allowing both unidirectional and bidirectional transformations. QVT-Core is a low level language that was created to be the target of QVT-Relations, but was never fully implemented. It has a simpler syntax than QVT-Relations though the transformations that are described using QVT-Core are more verbose. The objective however is to have QVT-Core be a language that can express any relationship between metamodels that is completely independent. That is, any model conforming to metamodel A can be transformed to a model conforming to metamodel B, without any inputs other than the two metamodels and the transformation expressed in QVT-Core. Also, the transformation back from B to A is entirely semantics preserving, without needing an additional input that contains the traces of the earlier A to B transformation. Which is not the case in QVT-Relations.

QVT-Core has never been fully implemented however, and to the knowledge of the author, neither has there ever been a full implementation of a semantics preserving transformation language that does not store traces explicitly. It almost goes without saying that a conceptual language like this would be of great value to any tool integration effort, including iFEST. After all, iFEST is using OSLC based metamodels that if integrated using a language such as QVT-Core could greatly reduce the development time of the tool adaptors.
4. The iFEST Tool Integration Framework (iFEST IF)

In a typical system development life cycle, different tools such as design tools, requirements authoring tools and testing tools are instantiated within their respective lifecycle phases, together forming a tool chain. These tools allow the creation and manipulation of relevant artifacts, based on underlying metamodels. These artifacts can be of various kinds. In the requirements engineering phase they can generally expected to be requirements. In the software design phase they can generally be expected to be software components. And so forth. However, these are merely examples and there are many more different flavors of artifacts.

As the life cycle progresses from one phase to the other, the ultimate goal of the entire life cycle is to retain consistency. Without going into a formal definition of consistency, it is not difficult to get an intuitive feel of what this consistency encompasses. A requirement defined in the requirements engineering phase should be captured by one or multiple software components in the software design phase. Respectively, all software components should be present as testing artifacts in the software testing phase.

If all artifacts in a certain phase are entirely traceable to artifacts in all the other phases, then from the standpoint of the engineer the system is flawless, as the artifacts across the different lifecycle phases are entirely consistent, meaning all requirements have been implemented, and all system shortcomings can be blamed on inadequate requirements.

But this kind of traceability is far from trivial to achieve. Not only do tools rely on different metamodels and do these metamodels in general differ greatly, there are many different tools being used in the industry today. Some tools provide APIs, some do not, some provide access to their local repositories and only a very small number of tools is open source.

The varying nature of tools makes tool integration a very challenging endeavor, with many different problems to solve. What’s more, there is an ever growing market demand for more powerful systems with a smaller and smaller size. The result is an ever increasing complexity of systems as requirements become more constrained. It is no longer an option to maintain subpar performance or have a device be larger than absolutely necessary. With increasing system complexity however comes an increasing complexity of the tools we use to design those very systems. Especially in the embedded systems range of tools available today, many are unable to cope with this increasing complexity, let alone provide adequate tool integration facilities. An increase in development costs, followed by increasing costs of poor quality is the result.

To develop a framework that will allow for the integration of embedded systems tools, in April 2010 the iFEST (industrial Framework for Embedded Systems Tools) project [3] was initiated. With 8 countries and 21 organizations participating directly and a duration of 3 years it is one of the largest tool integration research initiatives initiated to date.

In this chapter we discuss the iFEST project goals (4.1), iFEST key concepts (4.2), iFEST architecture (4.3) and make a comparison to other tool integration efforts (4.4).
4.1. Project goals

The iFEST project is a project aimed at specifying and developing a tool integration framework, the iFEST tool Integration Framework or iFEST IF, for hardware-software co-design of heterogeneous and multi-core embedded systems. The iFEST IF will allow tools to be readily placed within the tool chain. The framework will enable the tools to exchange data automatically, ideally eliminating the need for engineers to analogously make the transition from a tool in one life cycle phase to a tool in another. This is expected to significantly reduce the time-to-market of the system under development, reduce development costs and increase the quality of the overall system design. Also, it deals with the issues of tool lock-in and tool obsolescence, which are both less likely to occur as tools are more easily replaced. Switching costs can be lowered, allowing for a choice of the most adequate tool and its supplier rather than the one that is most attractive one when viewed from a commercial perspective.

The iFEST IF is validated by a number of industrial case studies carried out by the different iFEST partners. Two flavors of prototypical tool chains have been defined. The two application domains are data streaming and industrial control. Industrial case studies should be within one of these two application domains and demonstrate a potential reduction of development costs (1), time-to-market (2) and cost of poor quality (3).

The validation of the iFEST IF through industrial case studies focuses on tools that fit within the V-model development methodology. Having a rather broad definition, the V-model generally comes down to having two separate, converging “streams” of lifecycle activities or phases, together constituting the resemblance of a letter “V”. The activities in the left-hand arm of the V are the activities that are concerned with the actual inception of the artifacts that will together constitute the system being developed. The activities in the right-hand arm deal with the verification and validation of the activities on the left-hand side. Descending down the arms we move from activities dealing with artifacts on a higher meta level, down to tools dealing with more and more specific system details, to finally end up at implementation tools at the bottom of the V, where the streams come together. Although a lot of different editions of the V-model can be found in the literature, we categorize three different flavors of tools, depicted in Figure 12.
As we can see, the activities (depicted by the little squares) descend down the V, crossing lower and lower meta levels, to finally end up in a single final activity in which the system is fully developed. The dashed arrows across the V resemble the fact that activities in the left-hand arm always have a counterpart in the right-hand arm, that validates and verifies the activity outputs and feeds back.

The upper side left hand stream is defined as Requirements Engineering & Analysis, or RE&A for short. In RE&A we are concerned with defining requirements for the system we are building, both high and low level, in essence providing a blueprint on which the implementation should be based. The requirements are a reflection of what the client stakeholder wishes the system to do.

Design & Implementation, or D&I, is the set of phases in which we define and develop components that implement the requirements, in essence mapping those requirements onto implementation artifacts. These artifacts can be abstract, like software components, or physical, like an actual server on which software is deployed.

Verification & Validation finally is where we establish whether the requirements and system components match and are non-conflicting. Testing is an important aspect of V&V, ensuring components are of an adequate quality and behave as expected. Usually, within the V-model methodology there are additional maintenance activities after V&V, but within iFEST these are not considered.

The essence of the iFEST IF comes down to traceability between (artifacts in) the different phases. The horizontal arrows in between the two arms indicate a traceability relationship between the development artifacts and tests. That is to say, every requirement or system component is traceable to one or multiple tests in the V&V activities. In addition, the arrows leading back up the arms indicate a traceability relationship between the activities as they progress from the definition of requirements to design to
implementation. Every requirement should be traceable to one or more logical components, whereas every logical component should be traceable to one or more actual system components.

The iFEST IF is aiming to show benefits for tools in all of the three categories of lifecycle phases. Within Requirements Engineering & Analysis it will show an increased linkage of textual requirements to executable models and simulations. It will prevent loss of information and it will provide increased support for traceability to test cases and early validation of the requirements.

With respect to D&I the iFEST IF is expected to close the gap between the high level design of the system and its actual implementation components. Additionally, it will require less effort to replace proprietary tools and/or formats. This results in a more efficient design space exploration and more effective testing, simulation and evaluation opportunities.

As for V&V, requirements and model based verification will require less effort and become available as viable options earlier on the in the lifecycle. Furthermore, we are able to detect flaws and faults in a more timely fashion, leading to a reduced development time and cost.

Every tool used in the iFEST IF industrial case studies belongs to one of the three categories. Furthermore, every category has been given a definition in terms of process patterns, transformations and tool metamodels. HP-ALM\RM belongs to the category of Requirements Engineering & Analysis.
4.2. iFEST Key Concepts

In this section we take a look at the most important concepts that constitute the iFEST Tool Integration framework. Figure 13 provides a depiction of the metamodel that was defined at the start of the project.

Starting in the upper right corner, we find the concept of a tool chain. A tool chain is a set of tools and their corresponding tool adaptors that together form an integrated development environment. The tool chain may employ an integration platform that is typically provided by an external supplier. Integrated tools use tool adaptors to expose the artifacts in their tool instances in a common format. The adaptors comply to iFEST specifications, guidelines and principles. The tool adaptor specification consists of data and service specifications. The data specification provides a metamodel of the adaptor, whereas the service specification defines the tool adaptor in terms of provided and required services.

In its simplest terms, a tool adaptor provides a way for a systems tool, or as matter of fact any system component, to connect with other tools. A metaphor to an electrical adaptor is well suited to explain. Just like an electrical adaptor transforms one current to another because the machine it is connected to only supports a certain current, so does a tool adaptor provide the correct current or data formats for a given tool. Providing the right current however, is a far from trivial matter. Practicalities such as the availability of APIs aside, the transformation from one data format to another can be arbitrarily complex or even impossible to fully implement (i.e. retaining semantics completely). The tool adaptor is the software component that for a given tool implements this transformation from its native metamodel and data format and makes it available to other tools in another metamodel and data format.
Lastly, the iFEST tool Integration Framework provides a technological space. That is, the technology on which the implementation will rely. This includes but is not limited to data formats, communication protocols and standards. The chosen technology in the case of iFEST is OSLC, but conceptually any other suitable technology could serve.

In Figure 14 we see an example tool chain as it could look like within the iFEST IF. Tools, whether they are specific to a lifecycle phase – engineering tools – or cross-domain are connected to the framework and its technological space by means of a tool adaptor. The adaptor is defined in terms of required and provided services. The iFEST IF does not dictate in this respect. In addition to the tool adaptors for specific engineering tools, in the figure we also find traceability tools and process tools along with their respective adaptors. Together with a traceability and/or a process tool, a tool can form what is known as an integration platform. The traceability tools provides the facilities that allow for the creation of traces between artifacts in two different tools, whereas the process tool retrieves information it needs from the other tools to manage the development process, if any.
4.3. iFEST Architecture

In the framework, we find the tool instances, the tool adaptors, the repository (or Configuration Management System – CMS) and the service orchestrator. Tool instances represent the running instances of the tools used within the tool chains. Examples of tools include HP-ALM, Simulink, Enterprise Architect (EA) and IRQA. The tool adaptors are the means through which the tool instances communicate with the other participants in the framework. Paramount is that the adaptors expose tool data through common format. The IF repository is the entity in the framework that for every relevant tool stores its so-called IF versions. This is to say that for every tool in the tool chain, there exists a separate folder on the IF repository that contains the published versions of its model(s), either in OSLC, ReqIF [14] – Requirements Interchange Format, a recent standard for denoting requirements in XML -, some other formats or a combination of the three. The orchestrator is the final entity in the framework and provides general services to the other entities. The tool adaptors can for instance register themselves with the orchestrator, in order to receive notifications about newly published IF versions of models from tools they are interested in.

Figure 15. iFEST Tool Integration Framework architecture
The IF repository is a central storage facility that will maintain the models for every tool separately. Every tool is assigned its own unique folder, whereas an “IF Data” folder contains data relevant to the entire tool chain, such as traces between artifacts.

As we said, the repository will contain the so-called IF versions of the data contained in the tools. As a tool user is using his or her respective tool, the data that is being manipulated progresses through different versions. However, not all these versions include major changes. Therefore, only when the tool user publishes is a new IF version created. This is depicted in Figure 16.

![Figure 16. IF versions](image)

As we can see, a tool might have its own version management system that is entirely decoupled from the version management system of the entire tool chain. Not every minor change is relevant to enough to publish a new IF version. For this reason, a tool adaptor has direct access to its own folder on the IF repository, but the other tool adaptors access this folder through the framework.
4.4. How is iFEST different from other tool integration efforts?

iFEST is a framework meant to integrate tools that are used in the development of multi-core and heterogeneous embedded systems. iFEST takes a framework integration approach by building tool adaptors for the lifecycle tools it means to integrate and then have them communicate through a common data format. The central idea is that the lifecycle tools in the tool chain manipulate shared concepts and as such, the lifecycle tools should act in a coherent way. Two engineers working in the same domain with different lifecycle tools should be manipulating the same artifacts to avoid inconsistencies. Figure 17 visualizes the iFEST approach.
Ideally, the transition from one activity in the lifecycle to another is seamless, shown on the left-hand side of Figure 17. The tools employed share a common metamodel and communication between them is completely transparent. But this is far from today’s practical reality. In a lot of cases the transition between activities in tools is manual or semi-automatic at best. An engineer looks at the output of the activities up the tool chain and produces an intuitive mapping of the models to the engineer’s own domain. A designer creates components that implement the requirements specified by the requirements engineer, but whether or not all requirements have been addressed sufficiently is established mostly by hand.

In the iFEST Tool Integration framework, this transition is intended to be automated. The realization that tools in the tool chain have shared concepts makes it possible for them to speak a common language and as a result, (semi-) automatically transform models and validate their consistency. To integrate lifecycle tools in a tool chain using the iFEST framework, some development effort is required. The effort consists mainly of specifying and implementing the tool adaptors. The tool adaptor is a generally small piece of software that acts as a bridge or interface between the lifecycle tool and the framework. For every lifecycle tool a tool adaptor needs to be developed, but as we will see in chapter 6, a fairly large chunk of functionality is shared among adaptors and the amount of unique development effort is limited if we exploit this property through the use of SDKs and the like. Moreover, the iFEST framework will provide definitions of process flows describing the set of activities (and their respective tools) to be performed. These scenarios describe:

- How tools should be chained together.
- How tool data is exchanged and transformed.
- How tools are to be invoked.

iFEST aims to use the specifications of the Open Services for Lifecycle Collaboration (OSLC) [2] initiative. Being a fairly young initiative, these specifications are meant to standardize the way lifecycle tools share data, but the implementation efforts performed thus far at the time of writing are not many, nor are the specifications mature. Therefore, iFEST will try to adhere to OSLC as much as possible, but deviate where necessary. For instance, as we will see later on, OSLC’s capabilities to represent relevant models in the Wind Turbine industrial case study were deemed too verbose for transformations between HP-ALM and MATLAB Simulink, so a different solution was found.
The tools iFEST is targeting are focused around the design and implementation of heterogeneous, multi-core embedded systems. Because of the specific nature of these tools, their integration also requires special treatment.

Embedded systems are systems with a very specific application within a larger system, in a lot of cases intended to monitor or control some physical process under real-time constraints. The codebase of embedded systems is generally kept relatively small, as memory and processing power available may be limited. Exception handling and general fault tolerance is pruned. Also, as embedded systems often operate under real-time constraints, they are developed to be as efficient as possible.

However, the efficiency of these embedded systems comes at a price. Parallelization increases the system’s complexity, while fault tolerance is already being sacrificed. In other words, the risk of faulty behavior is bigger as is the risk of that faulty behavior causing a complete system failure.

Because of these increased risks, the modeling efforts performed in an embedded systems development project are of a more formal nature. Models of Computation (MoC) are employed to analyze algorithmic behavior. This includes analysis of execution time and memory use, but also behavior of individual variables and the operations performed on these variables. Examples of MoCs include Communicating Sequential Processes (CSP) by Hoare [15] and Petri nets by Petri [16]. They allow mathematical reasoning about the system and therein constitute a way to formally prove the system’s validity.

What’s more, The tools used in the iFEST IF manipulate models of a high complexity, typically containing thousands of artifacts. For requirements authoring tools there are, in addition to having functional and non-functional requirements, structural requirements that specify logical or actual components to be used in the system design tools. It is paramount that these requirements be transformed and traced accurately. It, in our case, encompassed additions to both the tool and tool adaptor, as will be discussed in chapter 6. The structural requirements are primarily logical blocks that carry out a specific function within the system. The blocks are wired together through the use of ports, which in turn can be grouped by interfaces.

The reason it was decided to manage these structural requirements within requirements authoring tools also, is because it allows us to establish traceability relationships from functional requirements to structural requirements within the same tool. Generally, the requirements engineer alongside defining the functional requirements also conceives a general system architecture, that is later exchanged to an actual D&I tool. The designer refines the architecture, after which the it is reimported into the requirements tool.

In summary, iFEST Tool Integration framework is subject to very specific requirements pertaining to the tool chain and process, as well as the tools themselves and the models they manipulate. In chapter 6 we will see how this specificity influenced the development of the HP-ALM/OSLC adaptor.
5. Open Services for Lifecycle Collaboration (OSLC)

As the iFEST project aims to use OSLC specifications and standards as part of the iFEST Tool Integration framework, in this chapter we introduce the reader to the general principles and concepts of OSLC (5.1). We also discuss how OSLC achieves tool integration (5.2) and how tool adaptors are implemented using OSLC (5.3).

5.1. General concepts

Open Services for Lifecycle Collaboration (OSLC) is a community of software developers, researchers and software professionals that was founded to create a set of specifications that aim to standardize the way software lifecycle tools share data. The specifications produced by the OSLC initiative are rapidly developing and its members include some of the world’s leading software companies and research institutes.

All OSLC specifications are based on or around the concept of Web services. That is, computer programs communicating over a network in a RESTful way, in most cases using HTTP as a protocol and RDF/XML as a way to represent the data that is being sent and received. Their communication is bound by what is known as a service contract, which in essence is a description of the functions a certain service implements. OSLC Web services are based on W3C’s concept of Linked Data [17]. Including, but not limited to a set of 4 simple rules:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL).
- Include links to other URIs, so that they can discover more things. (sic)

The OSLC Web services enable integration of tools, products and services that support Product Lifecycle Management (PLM) and Application Lifecycle Management (ALM). Each type of tool, product or service has its own group specification, alongside the existence of a core specification, on which all the others are based. Examples of these groups include Requirements Management (RM), Change Management (CM) and Quality Management (QM).

The core specification describes the primary techniques and patterns for integration.

Central to the OSLC specifications are Services, Resources and related concepts. All artifacts in the lifecycle - like requirements, tests, defects, etc. - are resources that can be accessed via HTTP using the Linked Data principles described above. I.e. every resource has a URI and links to other resources using URI’s. Each resource is required to provide an RDF representation, but an arbitrary number of other representations is allowed alongside RDF. Examples include HTML and JSON, an XML format that allows for serialization of Java objects.
The group specifications describe additional resource types that extend the concepts described in the core specification. The Requirements Management specification [18] describes the Requirement resource, Quality Management the Defect resource, etcetera. The concepts are depicted in the figure below.

![Figure 18. OSLC Core Concepts](image)

Tools exposed through Services are accessible via a Service Provider that describes the Services it offers in the form of a service catalog. The service catalog functions as a phonebook for services. In the catalog all service contracts are contained, allowing for the lookup of a certain service provider based on the descriptions of the services it provides.

Each Service can provide Creation Factories for Resource creation, Query Capabilities for resource querying and Delegated UI Dialogs to enable clients to create and select resources via a (graphical) UI. Query Capabilities and Creation Factories may offer Resource Shapes that describe the properties of resources managed by the Service. Resource Shapes are essentially the metamodels for Resources. However, Resource Shapes are not mandatory.
5.2. Integrating tools

The OSLC specification describes two strategies for integrating tools. The choice between the two strategies depends on the presence of an HTML UI that is attached to the tool that one wishes to integrate with. The first strategy “Linking data via HTTP” is the only viable approach when such a UI is absent. “Linking data via HTML User Interface” allows the integration to exploit already existing UI infrastructure.

5.2.1. Linking Data via HTTP

In order to create, retrieve, update and delete (CRUD) resources, HTTP and RDF in accordance with the Linked Data principles are used. To link and essentially integrate data of one tool with the other, HTTP URIs of resources are embedded in the representation of other resources. This approach requires the development of a tool adaptor that is capable of transforming HTTP requests to API calls to the targeted tool in that tool’s native data format. Without the presence of an API to the tool, this approach is not possible.

5.2.2. Linking Data via HTML User Interface

If the tool that is the target of integration provides an HTML based user interface that allows a user to manipulate its data, the “Linking Data via HTML User Interface” approach can be more efficient than the “Linking Data via HTTP” approach to develop. It basically means that instead of one tool integrating with the other by linking resources to each other, we embed the URL of a tool’s UI and exploit its capabilities.

OSLC describes what is referred to as an open model of resources and their states. The core specification describes properties for resources, as do the domain specifications for the domain resources, but they all assume that any given resource may have in practice many more properties than the ones that have been defined. In order for an OSLC client to still know what kind of properties to expect, the OSLC server may provide a Resource Shape containing all the properties of the resources it exposes. Besides knowing what properties to expect, clients should still assume that there may be more. This is possible in multiple ways. The most prominent being that not all resources of the same type have all the same properties. Another being that the Resource Shape is fixed at one point in time, but older and newer resources have different properties. Furthermore, not all references or links in OSLC resources point to other OSLC resources. A link can point to any kind of HTTP resource, not just other HTML or OSLC for that matter. These principles keep the OSLC models flexible, but on the other hand require additional programming as one should code defensively, not knowing the exact kind of data to be found at the end of an HTTP URI.
5.3. Implementing OSLC Web Services

To further clarify what the impact is of using OSLC to integrate tools, let us sketch an example that involves the implementation of an OSLC Web Service that will enable our tool to exchange data with others.

Suppose that we want to expose our tool Requirements Tool (RT) as OSLC compliant Web service(s). The first step is to determine which of the OSLC domains fits our lifecycle tool. RT is a tool concerned with authoring and managing requirements so the OSLC Requirements Management specification provides us with guidelines and principles to integrate our tool with both tools in the same domain as well as other domains like Configuration Management or Quality Management. To put it more specifically; We would like to build a Service Provider that exposes Requirement Resources and RequirementsCollection Resources that can be consumed by Service Consumers.

The next thing to do is to establish which kind of properties the Requirement Resources and the RequirementsCollection Resources will possess. The specification provides us with only a single mandatory property title and a number of optional attributes such as description and identifier. Besides having the possibility of specifying our own properties, we should use these predefined properties as much as we can as to limit complexity. If we however do need to invent new properties, we can provide the Service Consumers with a ResourceShape that will inform clients what kind of properties to expect.

The gist of exposing RT through OSLC compliant Web service(s) is the development of a tool adaptor. A tool adaptor is a program that communicates with RT - in most cases through APIs, but more simplistic approaches like file exchanges are not impossible - to retrieve the data it needs, transform it to “OSLC” and expose it through Web service(s). The difficulty is that every tool employs its own data formats. The developer of the tool adaptor needs to learn the data formats RT expects and transform it. To make matters worse, not every tool provides a metamodel, enforcing the developer to step-by-step compose it through trial-and-error. The developer has to deal with the idiosyncrasies of the tool and in essence mold them into the commonalities of OSLC.

The effort of developing a tool adaptor can now be split into two parts. The tasks of retrieving RT’s native data and transforming it are closely intertwined and have to be repeated for every unique tool. This is the internal part of tool adaptors. However, exposing the “OSLC” data is a task that is essentially the same for every tool adaptor targeting the same domain. Exposing the data is the external part. As the external part is susceptible to standardization, multiple development efforts to create System Development Kits (SDKs) that allow programmers to easily develop this part are underway, one of the most prominent being Eclipse Lyo [19]. Lyo was not used in the development of the HP-ALM/OSLC adaptor however, as at the moment development started it was not yet released.

After developing a tool adaptor for RT and deploying it on a Web server, what we require is some way to provide potential clients with the location of our Service Provider. In line with Service Oriented Architecture (SOA) principles, the OSLC Core specification suggests implementing a Service Provider Catalog, which does exactly this. However, the specification provides merely the concept and does not specify any guidelines or principles on how to implement it. Figure 19 visualizes all the aforementioned concepts.
In essence, what makes a service provider into an OSLC compliant service provider, is merely its behavior as seen externally. That is to say, the programming language, technology, etc. in which the service provider and services are implemented has no bearing at all. The internal part could be implemented in technology A, the connection to the lifecycle tool in technology B and the external part in technology C; as long as service consumer talking to the service provider perceives the expected OSLC service provider behavior, the service provider is OSLC compliant. Thus, the only true “OSLC part” in Figure 19 is the service provider and its services.
6. HP-ALM/OSLC adaptor

The goal of the iFEST project is to develop a framework that allows for the integration of tools that collaborate in the development of multi-core, heterogeneous embedded systems. As the data captured in the tools needs to be compatible with one another, the adaptors create the necessary bridge between the tools and the framework. One such a bridge is the HP-ALM/OSLC adaptor for the HP Application Lifecycle Management tool [4]. The HP-ALM/OSLC adaptor exposes requirements captured in HP-ALM as OSLC compliant resources. HP-ALM is one of the major tools in the Wind Turbine industrial case study’s prototype tool chain, which will be discussed in chapter 7.

This chapter reports on experiences implementing the adaptor, challenges faced and solutions found. We start off by giving an overview of HP-ALM and its Requirements Management module (6.1). Then follows a discussion of the general adaptor architecture and implementation (6.2).

6.1. HP-ALM Requirements Management

Application lifecycle management or ALM refers to the continuous process of managing the lifecycle of an application through governance, development and operations [20]. In an attempt to make the system development process more transparent from a business management perspective, the term was first coined in the late 2000’s. It is often used as an umbrella term for both the process of making the information flow between different software lifecycle phases more transparent, as well as the tools that accommodate this. Including, but not limited to HP’s ALM software suite, HP-ALM, ALM tools generally bundle different tools with different purposes – such as requirements management, testing, etc. – together, accompanied by so-called process management tools that gather information about the other tools and make it accessible to business managers or other relevant stakeholders. This information can contain metrics ranging from very simple ones, such as how many tests were carried out in a given week, up to more complex ones, such as the average productivity of developers, calculated using several different metrics combined.

HP-ALM bundles several different tools into a single suite. These tools are aimed at the activities of requirements management, test planning, functional and performance testing, developer management and defect management. The tool aimed at requirement management, i.e. the activity of putting requirements authoring into its broader context, is referred to as the HP-ALM Requirements Management module. Not only does the tool allow requirements to be captured and managed, HP-ALM provides advanced tracking and analysis capabilities that allow managers to attain a quick insight into the overall status of the requirements specification and its relationship to other important project artifacts and deliverables.
Because HP-ALM is a collection of different tools that should be able to be used simultaneously and exchange relevant data, it is deployed as a centralized Web based solution. That is, rather than installing it locally, HP-ALM is deployed on a Web server. Hence, the entire tool suite is accessible by navigating a Web browser to a common URL. To accommodate safe concurrent access, HP-ALM implements locking and version control mechanisms.

Within the Wind Turbine industrial case study, the HP-ALM Requirements Management module (HP-ALM/RM) is the tool of choice for requirements authoring and management. Next follows a description of how HP-ALM handles requirements, how well its approach suits the needs of the Wind Turbine and which modifications or workarounds had to be performed in order to make the tool adequately represent the requirements that were specified within the industrial case study.

In HP-ALM/RM, requirements are structured in a file system-like hierarchy. In this hierarchy, every requirement is basically a file and folder at the same time. That is, every requirement has some properties - such as Name, Direct Cover Status, Author - but can also contain other requirements.
Creating and managing requirements in HP-ALM is straightforward. In the dialogue screen showed in Figure 21, the user can input values for the different properties of a requirement, such as Name, Priority and Requirement Type.

Through HP-ALM’s administrator panel, it is possible to extend the properties of a requirement, or create entirely new properties. It is also possible to create entirely new requirement types or extend existing attribute enumeration values. In the Wind Turbine industrial case study several new attributes, requirement types or attribute enumeration values had to be created.

Apart from interacting with the requirements stored in HP-ALM through the GUI, HP has also provided an HTTP based RESTful API that allows for communication with virtually any other program. The HP-ALM/OSLC Adaptor makes use of this API to insert and retrieve data from HP-ALM. A more thorough explanation of the HP-ALM RESTful API will follow in section 6.2.
6.2. HP-ALM/OSLC Adaptor Architecture

As was mentioned earlier, the data manipulated by different tools within the framework is generally not compatible with one another. That is, tools may use different names for what is semantically the same attribute, but the structure or metamodel of the artifacts managed by the tool may differ completely. Tool integration and metamodeling is however an entire subfield within the tool integration field of research. It is discussed in detail in section 3.3.

Because of the incompatibility of tools, we need a software component called a tool adaptor that bridges the gap and exposes the data of one tool in a format that is known to other tools, or a format that can be transformed into another tool’s native format.

In the case of iFEST, these data formats are the data formats specified by the Open Services for Lifecycle Collaboration (OSLC) initiative. The OSLC initiative aims to “standardize the way lifecycle tools can share data”. OSLC was discussed in great detail in chapter 5.

As such, within OSLC a specification for requirements management tools was created [18]. This specification mandates in which format a requirements management tool adaptor should expose its data. It demands certain attributes a requirement artifact should possess when it is being exposed, but also makes recommendations about the behavior of the adaptor. The OSLC specifications are highly extendable however, allowing for the definition of additional attributes for instance. In the Wind Turbine industrial case study the OSLC Requirements metamodel was extended with a number of additional attributes.

However, one can only make a finite amount of extensions to a certain standard or specification before it tends to overstretch. That is, we are employing the standard or specification towards a certain purpose that lies far from what the original intention of the standard or specification was.

Among the key scenarios of the Wind Turbine industrial case study we find scenarios that involve transformations of highly complex embedded systems models. Model transformations however are not what the OSLC specifications were designed for. For this reason, besides the OSLC Web service endpoint, a whole other Java Swing based graphical endpoint was implemented, aimed at accommodating model transformations using a slightly modified version of HP-ALM’s native XML format and publishing the models into local or remote repositories.

In this section we discuss the challenges faced while developing the HP-ALM/OSLC adaptor, starting with the internal endpoint of the adaptor, then moving on the OSLC Web service endpoint. Then follows a discussion of the Java Swing based GUI endpoint.
Figure 22 provides an overview of the HP-ALM/OSLC architecture. Both the OSLC Web Service endpoint and the GUI endpoint use the internal endpoint to connect to HP-ALM. The internal endpoint exploits the functionalities of HP-ALM’s HTTP REST API. The GUI and OSLC WS endpoints are connected in the picture, but can be deployed completely separately. In the upcoming subsections we discuss all the components in the figure.
6.2.1. The internal endpoint

The internal endpoint is the part of the adaptor that communicates with the tool in question. HP-ALM provides an API that is based on HTTP RESTful communication that allows retrieval and insertion of resources in XML format. That is, it allows external applications to use the common HTTP methods GET, POST, PUT and DELETE to manipulate its data. This is exactly what the internal part of the adaptor does, implemented purely in Java, using the Apache HttpComponents libraries [21].

The schema that this XML is based on, is relatively simple. Central in HP-ALM is the concept of entities. Everything in HP-ALM, from a requirement to defect, is an entity. All Entity elements have a single Fields element. This Fields element contains at least one Field element, whereas a Field element contains a completely arbitrary number of Value elements.

![Figure 23. HP-ALM Entity schema](image)

In Figure 23 the HP-ALM Entity schema is depicted. When the user retrieves an Entity through HP-ALM’s API, the XML that is returned in the HTTP response looks as follows:
As we can see in the above figure, there are no XML elements called Requirement or Defect, rather they are simply Entity elements with a predefined number of Field elements. This keeps the schema flexible, as the introduction of a new attribute does not require the alteration of the schema. The downside however is that the schema itself does not provide enough information to determine what kind of attributes a certain entity generally possesses. Because of this, when one would like to perform model transformations for instance and maintain the attribute values, one would have to define an additional, more restrictive schema that specifies what Field elements an Entity element of a certain type would possess.
6.2.1.1. **Internal endpoint challenges**

After receiving the XML information from HP-ALM, it should be processed further. This processing is not always straightforward however and some difficult challenges were encountered during development. The most prominent being the fact that, although the reference and the schema claim otherwise, the API does not seem to support multi-value attributes. An essential part of the requirements specification in the Wind Turbine industrial case study are the ports. These ports connect logical blocks to each other and without them the models would be incomplete. The original intention was to use the ‘trace-to’ and ‘trace-from’ Field elements that all requirement Entity’s possess, but since they are multi-value their value would not be returned via the API. A workaround had to be found and eventually it was settled that the introduction of a new attribute was an acceptable one. This attribute is a mere single field that specifies for a given port whichever port connects to it. Fortunately, this can never be more than one port, so the replacement of a multi-value attribute with a single value attribute in this case resulted in no loss of information.

Another issue that was encountered developing the internal endpoint of the HP-ALM/OSLC adaptor was the way HP-ALM handles versions. HP-ALM possesses quite sophisticated version control features, but quite surprisingly, there is no notion of an overall model version. That is to say, when an individual requirement is updated to a new version, it is only that requirement’s own version identifier that is increased. As a result, there is no way of telling that the model has changed apart from inspecting individual requirements. This makes distinguishing between different model versions more difficult. To overcome this problem, the HP-ALM/OSLC adaptors maintains its own notion of model versions in a local file. This way, every time a new version of the requirements model is published, it updates the model version identifier and records whichever individual requirements and their versions belong to that model version. Let us clarify this approach by sketching an example.
At the start there are two requirements in our requirements model, and every entity is at version 1:

Table 1. Requirements model

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity</th>
<th>Status</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirements Model (RM)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RM/Some Requirement</td>
<td>On-going</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>RM/Another Requirement</td>
<td>On-going</td>
<td>1</td>
</tr>
</tbody>
</table>

Now the requirements engineer using HP-ALM makes a modification to the requirement with ID 2:

Table 2. Requirements model after changes

<table>
<thead>
<tr>
<th>ID</th>
<th>Entity</th>
<th>Status</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Requirements Model (RM)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>RM/Some Requirement</td>
<td>Finalized</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>RM/Another Requirement</td>
<td>On-going</td>
<td>1</td>
</tr>
</tbody>
</table>

As we can see in the table above, the Requirements Model (RM) did not automatically have its version identifier increased also. Even though the requirements model was changed, there is no way of telling it apart from its previous state, as it’s still at the same version. On the other hand, we would not want for every small change that is made to individual requirements to increase the version identifier of the requirements model. This would lead to an unnecessarily large amount of different versions.

These two problems are solved by the adaptor maintaining its own mapping of requirements model versions to individual requirements and their versions. Moreover, the requirements model version is only updated to a new version when the requirements engineer wants it so. The requirements engineer indicates through the GUI endpoint that he wishes to publish a new version and as a result, the adaptor updates the mapping file. The mapping file is a simple ASCII file that merely states requirements model versions and the ID’s and versions of the requirements it contains:

Table 3. Version mapping

<table>
<thead>
<tr>
<th>Version 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:Some Requirement/Version 1</td>
</tr>
<tr>
<td>3:Another Requirement/Version 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:Some Requirement/Version 2</td>
</tr>
<tr>
<td>3:Another Requirement/Version 1</td>
</tr>
</tbody>
</table>
The requirements model versions are completely transparent to HP-ALM and are only visible through the use of the adaptor. HP-ALM itself holds no notion of the requirements model version, but only of individual versions. Thus, the approach does not work in case requirements are deleted in HP-ALM. When a user deletes a requirement, its record is removed from the system altogether. Therefore, in the Wind Turbine industrial case study, no requirement is ever actually deleted from HP-ALM, but rather has its status field put to ‘Deleted’.

6.2.2. The OSLC Web Service endpoint

Central to the manner in which the OSLC specifications aim to achieve tool integration is the concept of Web services. According to the OSLC philosophy, every relevant resource or artifact in the software lifecycle such as a requirement or defect should be accessible via a unique and consistent URI. Therefore, the HP-ALM/OSLC adaptor too is implemented as a Web service.

The HP-ALM OSLC Web Service (WS) endpoint was developed using FraSCAti [22], an open-source implementation of the Service Component Architecture (SCA) [23], which is a programming framework used for applications that follow Service Oriented Architecture (SOA) principles. Among the key advantages of SCA are its high degree of decoupling between a service interface and its implementation and the high level of configurability of SCA components and composites.

Using SCA, the Web service can be kept decoupled from its implementation, giving the tool adaptor developer the choice of technology in which it is most convenient to make the connection to the tool. In case of the HP-ALM adaptor we chose Java, as we had prior experience with the technology. Theoretically speaking however, it could have been any other technology for which SCA provides bindings. This means that for other tools, which may provide APIs or accessibility mechanisms supporting only a specific mechanism, SCA is extremely convenient to use. The internal structure of the OSLC WS endpoint is depicted in Figure 24.
At the bottom of the picture we find the Tool Instance. This Tool Instance represents the deployed and running tool instance of HP-ALM. Within any tool we generally find the concept of elements and their interrelations. We generally refer to these as the artifacts. In our case, these are the requirements as discussed in Appendix A. Full Wind Turbine Requirement Specification. Before the requirements can be exposed however, they need to be transformed to the formats as described in the OSLC Requirements Management specification. Or put more precisely, the artifacts need to be transformed to OSLC Requirement Resources.

This transformation, for any tool adaptor, be it an OSLC tool adaptor or not, is the unique part of its implementation. Every tool has its own data formats and idiosyncrasies so the transformation needs to be rewritten for every tool. After the data is transformed to the OSLC schema and formats, it is exposed through Web services, depicted by the white and grey squares at the top of the picture. The OSLC endpoint offers a number of provided services, but may also require other services to supply data. The provided services are the services that other parties in the framework can use to access the OSLC Resources. An example of a required service on the other hand might be a notification service that notifies the tool adaptor about issues that were filed pertaining to the latest published version of the requirements specification.

As the task of exposing the OSLC compliant data is generic across all requirements management tools, we can develop and use SDKs to provide this functionality. The HP-ALM/OSLC adaptor uses a prototype of the KTH SDK, an SDK doing exactly this, developed by the Kungliga Tekniska Högskolan in Stockholm. The KTH SDK provides convenient interfaces and classes that smoothen the deployment of
OSLC Resources as actual Web resources. Another example of an OSLC SDK is the OSLC4J (OSLC for Java) SDK developed within the Eclipse Lyo project [19].

When the tool adaptor for a given tool is ready, it can be connected to other tools and their tool adaptors in the iFEST tool integration framework. This is depicted in Figure 25.

![Figure 25. Tool adaptors communicating with one another](image)

In the figure, we find HP-ALM at the bottom. It is connected to the HP-ALM/OSLC Adaptor by means of the HTTP REST API it provides. The Java based adaptor makes HTTP calls to HP-ALM through this API and uses the KTH SDK to process the information it retrieves into OSLC artifacts. It is then ready to communicate with the other tool adaptors in the iFEST framework.

The services the OSLC endpoint implements are based on the CRUD model (Create, Read, Update, Delete) and its corresponding HTTP counterparts POST, GET, PUT and DELETE. The Requirements Management OSLC specification mandates certain services, whereas some custom services were implemented also.
In Table 4 all services under the “OSLC services” header are related to the extraction and manipulation of OSLC Requirement and RequirementCollection resources. A RequirementCollection is merely an aggregate of Requirement resources and the OSLC RM specification makes no recommendations about how exactly this aggregation should be performed. Beneath the “iFEST specific” header we find two services, issueNotification and updateAvailable, that are not defined by the OSLC specifications, but were conceived in the specific context of the case.
Besides provided services, there are also three services that the OSLC endpoint requires. Listed in Table 5.

Table 5. Required services

<table>
<thead>
<tr>
<th>Required services</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>Register to the service catalog</td>
</tr>
<tr>
<td>subscribe</td>
<td>Subscribe to updates from a service listed in the service catalog</td>
</tr>
<tr>
<td>sendNotification</td>
<td>Send a notification to another service provider.</td>
</tr>
</tbody>
</table>

All these services are required from the service orchestrator, which was discussed in 4.3 about the iFEST architecture. “Register” is used when the adaptor comes online and wants to make itself known to the other services in the tool chain. “Subscribe” is used to subscribe to changes in another tool. That is, when that particular tool publishes a new IF version, we are sure to receive a notification of the event. “SendNotification” finally is used to send notifications to the other tool adaptors. We will clarify how all the services are used in their proper context in section 7.5.

The concepts of a Requirement and RequirementCollection are defined in the OSLC Requirements Management specification [18]. Thus, a HTTP GET called for a Requirement with, let’s say identifier 4, would look like this:

```
http://localhost:8787/Requirements/requirement/4
```

This request would yield an XML response like the one below.

```
  <oslc_requirements_data:Requirement rdf:about="http://localhost:8787/Requirements/requirement/4">
    <oslc_requirements_data:requirementType>Folder</oslc_requirements_data:requirementType>
    <rdfs:label>Authentication</rdfs:label>
  </oslc_requirements_data:Requirement>
</rdf:RDF>
```
In this response, we find a lot of the different attributes that belong to a Requirement as specified in the OSLC specification. The challenge is mapping the attributes as they are returned from the HP-ALM RESTful API to the ones specified by OSLC. Not all mappings are trivial and sometimes there does not exist an equivalent at all for a given attribute. On the left hand side of Table 6, we see the HP-ALM attributes, whereas on the right hand side we see the OSLC Requirement attributes to which they were mapped.
Table 6. Mapping attributes

<table>
<thead>
<tr>
<th>HP-ALM attribute</th>
<th>Description</th>
<th>OSLC Requirement attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>Original creator of the requirement</td>
<td>dcterms: creator</td>
</tr>
<tr>
<td>creation-time</td>
<td>Time at which the requirement was created</td>
<td>dcterms: created</td>
</tr>
<tr>
<td>req-comment</td>
<td>Comment</td>
<td>dcterms: description</td>
</tr>
<tr>
<td>last-modified</td>
<td>Time at which the requirement was last modified</td>
<td>dcterms: modified</td>
</tr>
<tr>
<td>name</td>
<td>Name of the requirement</td>
<td>dcterms: title</td>
</tr>
<tr>
<td>id</td>
<td>ID of the requirement</td>
<td>dcterms: name</td>
</tr>
<tr>
<td>type-id</td>
<td>ID of the requirement type</td>
<td>requirementType</td>
</tr>
<tr>
<td>vc-version-number</td>
<td>Version identifier</td>
<td>version</td>
</tr>
<tr>
<td>status</td>
<td>Status of the requirement</td>
<td>directCoverStatus</td>
</tr>
<tr>
<td>req-priority</td>
<td>Priority of the requirement</td>
<td>priority</td>
</tr>
<tr>
<td>parent-id</td>
<td>ID of the parent of the requirement</td>
<td>oslc_rm: elaborates</td>
</tr>
</tbody>
</table>

We extended the OSLC Requirement Resource attributes with four new ones. These are:

- `version`
- `directCoverStatus`
- `priority`
- `requirementType`

For these attributes we could not find a similar one already present in the OSLC predefined attributes. Version, direct cover status, and requirement type directly map to their namesake attributes found in the native HP-ALM requirements. Priority maps to 'req-priority’ in HP-ALM.

Version is self-explanatory, as is priority. Requirement type corresponds to the types as discussed in section 6.1 (Figure 32). That is, requirements can be either functional, non-functional, blocks, ports or interfaces. The values of direct cover status are the requirement statuses as discussed in section 6.1 (Figure 31).

Another extension that was made to the original OSLC Requirements Management specification, was the inclusion of a version as parameter to the GET methods for Requirement and RequirementCollection OSLC Resources. This means that instead of retrieving the latest version of a certain requirement, as we just did for a requirement with ID ‘4’, we can also retrieve a specific version, by directing our client to this URL:

`localhost:8787/Requirements/requirement/4/versions/2`
Enabling this feature for RequirementsCollection OSLC Resources was not straightforward. As we explained in section 6.2.1, HP-ALM holds no notion of model versions. Our solution to this was to maintain a separate versions file.

Thanks to the versions file, whenever a new IF version is published using the GUI endpoint, explained in the next section, that version is also visible from the OSLC WS endpoint. Thus, we can direct our client to the URL:

\[
\text{localhost:8787/Requirements/requirement\_collection/0/versions/1}
\]

This will retrieve the entire requirements model with IF version ‘1’ and display all the requirements that belong to it.

6.2.3. The GUI endpoint

In addition to exposing the requirements captured in HP-ALM/RM as OSLC compliant artifacts, the adaptor provides a whole other range of functionalities related to model transformations directly from a slightly modified version of HP-ALM’s native XML format that it outputs through the RESTful API to other formats.

The original intention in the Wind Turbine industrial case study was to perform model transformations from the OSLC data exposed by the HP-ALM/OSLC to Simulink models but OSLC was later deemed too verbose to implement transformations for. One of the factors is that OSLC uses the RDF [24] (Resource Description Framework) format, in which resources generally link to each other by specifying URIs as opposed to having an inline specification. Consequently, a model transformation would have to follow the links to be able to construct a complete model of the requirements. The party in the Wind Turbine industrial case study responsible for implementation of model transformations indicated that this was too complex a task. As a result, the decision was taken to perform model transformations that use HP-ALM’s native XML as an input.

The model transformation using HP-ALM’s native XML in addition requires that the requirements model is a .xml file accessible somewhere on the file system. HP-ALM however does not provide the functionality to export to .xml files natively, nor does it allow extensions to its graphical user interface. Therefore, the decision was taken to implement an additional GUI endpoint using Java Swing that would provide file export and any other features that a requirements engineer using HP-ALM would need and that could not be sensibly implemented as part of the OSLC Web service endpoint. The Java Swing based GUI endpoint provides three primary functionalities:

1. Publish requirements
2. Import requirements
3. Update requirements

Publishing requirements refers to the act of producing an actual .xml file representing the entire requirements model and assigning a version to it. Essentially this .xml file is a snapshot of the requirements model at a certain point in time. It is stored either locally or on a remote storage facility.
When requirements are imported, the adaptor assumes that its local repository is empty and therefore simply overwrites it with the incoming data. Hence for a usual project this step is taken either only once or never, depending on whether the initial requirements model is created in HP-ALM or elsewhere.

Updating requirements finally is when the requirements in the local repository are updated according to the changes indicated in the incoming data, regardless of its source. That is, the adaptor compares the local model with the incoming model and makes changes into the local model accordingly. It lets the user select whichever changes he accepts or rejects.

The GUI endpoint is able to connect to an OSLC Software Configuration Management Service Provider, compliant to the OSLC Software Configuration Management specification [25]. This specification is deprecated at the time of writing however, and being superseded by the OSLC Configuration Management specification [26], which is still a work in progress. Nonetheless, let us clarify what this connection encompasses.

The OSLC SCM specification introduces two important OSLC Resources: OSLC DirectoryVersions and OSLC FileVersions. These resources represent actual directories and files in the configuration management system respectively. DirectoryVersion OSLC Resources contain only a single additional attribute compared to standard OSLC Resources indicating whichever OSLC Resources are contained inside the directory. As for FileVersions, there are two. One is the actual file content, whereas the other is the MIME type of the file.

The .xml file representing the entire requirements model can now be contained inside a FileVersion OSLC Resource and be inserted to and retrieved from the CMS. This translates to the GUI endpoint requiring at least two services from the CMS:

<table>
<thead>
<tr>
<th>OSLC</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET FileVersion</td>
<td>Retrieve a FileVersion.</td>
</tr>
<tr>
<td>POST FileVersion</td>
<td>Create a FileVersion.</td>
</tr>
</tbody>
</table>

For every new version of the requirements model a new FileVersion is posted to the CMS, contained in the designated directory for our tool. More on this follows in the next section.
The attentive reader might now be wondering how exactly the HP-ALM adaptor and its endpoints tie in to the iFEST IF architecture. In previous sections we exhibited the adaptor as being a single computational entity, but this in reality is not how the adaptor is deployed. Figure 26 explains.

The OSLC Service endpoint and the GUI endpoint are in fact two entirely separate software components, deployed on separate machines. Although they both fulfill an essential role, their functionalities can be decoupled completely. The OSLC Service endpoint is deployed on a Web server, as fundamental to Web services is that they be available to the network and to multiple users at the same time and with a high availability. As we said, the OSLC Service endpoint is to be used by actors other than the HP-ALM engineer.

The GUI endpoint on the other hand is used by the HP-ALM engineer for more powerful purposes, such as importing an entirely new HP-ALM requirements model from a file or publishing a new version to the Configuration Management System (CMS). The GUI endpoint is a locally run component deployed through an executable file that can be started and stopped on demand.
7. The Wind turbine industrial case study

To validate the iFEST Tool Integration framework, two flavors of prototypical tool chains have been defined. Industrial case studies should be within one of these two application domains and demonstrate a potential reduction of development costs (1), time-to-market (2) and cost of poor quality (3). The two application domains are data streaming and industrial control.

The Wind Turbine is an industrial case study in the latter domain. Its goal is to attempt to integrate a number of embedded systems tools used in the development of a controller for a wind turbine. The controller being developed is an embedded systems solution that should make use of multi-core CPUs and take a holistic approach to hardware/software co-design including the usage of FPGAs. The controller for this wind turbine aims to achieve optimal parameters for generating power, based on a number of input variables. Examples include the measurement of variables such as wind direction and velocity and the control of parameters such as the pitch of the propeller blades and the direction of the nacelle.

The iFEST IF comes into play by improving tool integration between requirements authoring and system design tools through the use of tool adaptors. This allows the assessment of the iFEST integration technologies and refinement of the framework. The adaptors are specifically developed for use in the industrial case study and are subject to continuous improvement as the integration framework matures.

The purpose of the industrial case study is to validate the iFEST IF and the tool chain prototype and demonstrate that the three aforementioned goals are being met.

In this chapter, we give a brief introduction to the wind turbine controller (7.1) and we discuss the Wind Turbine development process (7.2). We move on to give an overview of the tools in the Wind Turbine tool chain and how they fit together (7.3). Then follows a discussion of the Wind Turbine requirements structure and how it influenced the adaptor (7.4). Finally, we discuss three distinct integration scenarios that involved the HP-ALM/OSLC adaptor and their results (7.5).
7.1. The Wind Turbine controller

The system being developed is a controller for a wind powered turbine system that contains an electric generator connected to rotating propeller blades. The power thus generated is compared against a model representing a general user demand for electric power. There are sensors that measure parameters such as wind velocity and direction, speed of the propeller unit generated power. Actuators control variables like the pitch of the propeller blades and the direction of the nacelle. This way, the optimum conditions for generating power can be achieved. However, the power generation of the wind turbine should be within the margins of user demand, as not to generate too little or too much power. Figure 27 depicts typical components of a wind turbine. The component of interest in our case is the controller, the purple cube attached to the generator. More on wind turbines and their design principles can be found in Pao et al [27].

Figure 27. Typical components of a wind turbine
7.2. The Wind Turbine development process

In the development process of the Wind Turbine controller three major process steps have been defined. They are visualized in Figure 28 [28]. As the very first step in the process, the system owner specifies the requirements that the system should implement, divided into functional and non-functional requirements. After the requirements are specified, the system designer specifies a system design that maps the functional requirements onto implementation agnostic components. These components are then transformed to actual deployment artifacts in the implementation design. The final step is the implementation step, in which both the necessary hardware and software components are developed or acquired and put together to constitute the actual system. The process steps and their respective deliverables are continuously verified to see if they retain a consistent picture of the system under development. In addition, the outputs of the steps are validated with their relevant stakeholders, as to ensure that the final product meets expectations.

Figure 28. Wind Turbine development process [28]
7.3. Wind turbine tool chain

The tools used in the Wind Turbine tool chain can roughly be divided into three categories:

- Requirements Engineering & Analysis (RE&A)
- Design & Implementation (D&I)
- Verification & Validation (V&V)

In RE&A we find the Requirements Management module of HP-ALM and IRQA [29], a requirements authoring tool developed by Visure Solutions. HP-ALM and IRQA are used in the Wind Turbine industrial case study to represent the same requirements model, and part of the factors that could make the industrial case study to be considered a success are related to the successful on-the-fly exchange of HP-ALM and IRQA without any loss of information or extra manual effort to maintain model consistency.

Simulink [10], a tool used to create block diagrams that represent dynamic systems, is the main tool used in the D&I category of tools. Simulink allows for advanced analysis and simulation of models, automatic code generation and offers tight integration with Matlab[30]. This integration includes the import of Matlab algorithms into Simulink and test cases vice versa. Enterprise Architect [31] is another - UML compliant - D&I tool, but is at the time of writing merely prospected to be used in the iFEST industrial case studies. It is therefore outside the scope of this work.

![Tool used in the Wind Turbine industrial case study](image-url)
7.4. Wind Turbine requirements

The requirement specification for the wind turbine controller is fairly large and consists of requirements of many different types. The requirements are structured in a hierarchy of multiple levels, where the uppermost node in the hierarchy branches into High Level Requirements and System Requirements. Both categories are then further subdivided into Non-Functional and Functional Requirements. In the Functional Requirements we find the Plant and Control System categories, the latter containing requirements related to the actual controller while the first contains requirements related to the rest of the wind turbine system components. Non-Functional Requirements can be of four different categories; Performance, Safety, Standards and Design. The structure of the Wind Turbine requirements is depicted in Figure 30 below. For an overview of all the requirements in the industrial case study, refer to Appendix A. Full Wind Turbine Requirement Specification.

In HP-ALM, every requirement is required to be of a certain Requirement Type. HP-ALM provides a number of Requirement Types, by default, but it also allows administrators to define new Requirement Types. For the requirements in the Wind Turbine industrial case study, two of the existing Requirement Types – Functional requirements and Folder requirements – were used, while several different new types had to be defined. The new Requirement Types that were specified are Block, InPort, OutPort, Interface, and Non-Functional requirements. A Block refers to a block in a block diagram, which is an isolated part of the system with a specific function. InPort requirements represent input ports into Blocks, whereas OutPort requirements represent output ports of Block, i.e. ports that output the result of the function that the Block implements. Interface requirements aggregate related ports, for instance ports of the same block or ports with similar functionalities.
By default, HP-ALM provides a large number of predefined attributes for requirements. However, in the Wind Turbine industrial case study only a few attributes are important. These are: ID, name, description, priority, and status. All of these attributes have adequate counterparts in HP-ALM’s predefined requirements, except status. The status of a requirement in the industrial case study is represented by an enumeration that can take the following values: Ongoing, Finalized, Under Review, Approved, Rejected and Deleted. Every requirement starts out as Ongoing and is eventually Finalized and put Under Review, but the subsequent statuses thereafter depend on the interaction between the requirements author and the reviewer. The state flow diagram in Figure 31 shows how requirements progress through the different statuses.

![Figure 31. Requirement statuses](image)

After a requirement is reviewed, the reviewer either approves the requirement, meaning it is accepted as valid requirement and is not to be altered in the future, or he rejects the requirement. Based on the review, the requirements author decides to either delete the requirement altogether, or alter it. In the latter case, the status is changed back to Ongoing, progresses through Finalized and is finally reviewed again. This process repeats until requirements are either Deleted or Approved. All of these statuses had to be introduced into HP-ALM as additional enumeration values of the HP-ALM’s native Direct Cover Status attribute.
The final addition that was made to HP-ALM was the introduction of the ‘outport’ attribute to requirements of types InPort and OutPort, or ports for short. This attribute indicates for a given port whichever other port connects to it. Because every port is connected to by either no or a single other port and because HP-ALM behaves poorly with regard to multi-value attributes, it was decided not to specify explicitly whichever other ports a port connects from. Rather, this information should be derived from the ‘outport’ attribute. Additional explanation as to why this decision was taken can be found in section 6.2. Figure 32 summarizes the structure of requirements in the Wind Turbine industrial case study.

Figure 32. Requirement structure
7.5. Integrating tools

In this section we discuss three scenarios that demonstrate the most important features of the HP-ALM/OSLC adaptor and how they are used in an actual tool chain instance. Scenario 1 demonstrates the features of the GUI endpoint whereas scenarios 2 and 3 demonstrate features of both the OSLC endpoint and the GUI endpoint.

Scenario 1: Transformation to design artifacts

Key to the validation and verification of the HP-ALM/OSLC adaptor in the context of the Wind Turbine industrial case study is a prototype integration with Simulink [10]. The HP-ALM model is transformed to a corresponding Simulink model and vice versa using the MDWorkbench [9] model transformation environment. Figure 33 visualizes the essential steps that compose this integration scenario.

![Figure 33. Transformations to and from MATLAB Simulink](image)

At the start of the project, the requirements engineer using HP-ALM starts working on the requirements and produces a first version of the HP-ALM requirements model (step 1). At some point the HP-ALM model is uploaded to the Configuration Management System (2). Since the CMS is OSLC compliant we can use a Web-based delegated UI to browse the artifacts it contains (3). We use the HP-ALM model as an input to the transformation (4) and receive a Simulink model in return (5). The Simulink model is fed back to the actual tool instance (6), after which the Simulink engineer makes modifications to it (7). After the work is done, the Simulink model is uploaded to the CMS again (8) and transformed back to an HP-ALM model (9 and 10). Finally, the HP-ALM engineer picks up the latest model from the CMS (11) and continues the work.
Steps 1-3

After using the HP-ALM user interface as discussed in section 6.1, at some point in time the engineer considers the requirements model to be ready for publication to the CMS. At this point he launches the adaptor, after which the interface of Figure 34 is shown.

![Figure 34. Adaptor interface](image)

After pressing the button in the upper right corner, ‘Write requirements to a file’ an XML file containing the entire HP-ALM model is created in a folder in the local file system. This XML file is in a slightly modified format as compared to HP-ALM’s native format. The main modification is the exclusion of attributes HP-ALM attributes that are not used.
Now when the engineer clicks ‘Publish requirements to CMS’ he is prompted to select a file from the local file system and a URL of a CMS into which to publish the file. After publishing the file to the CMS, we can use the CMS delegated UI to see the file. We browse to the appropriate OSLC DirectoryVersion and FileVersion [26].

Steps 4 & 5

We use MDWorkbench, a model transformation environment based on Eclipse, to perform a transformation from the HP-ALM model to a MATLAB Simulink model. The transformation considers all structural requirements in the HP-ALM model. That is, requirements that are of type Block, InPort, OutPort or Interface, as discussed in section 7.4.

As we can see in Figure 36, we use the newly produced XML file containing a representation of the HP-ALM model as an input to the transformation, whereas the output of the transformation is a Simulink model file (.mdl) that will be placed in its appropriate position in the local file system.
Steps 6 – 8

After importing the Simulink model using the Simulink adaptor, the engineer makes changes to the model.

In Figure 38 the Simulink model that was the result of the transformation from HP-ALM to Simulink is shown. Main Controller and Pitch Control are logical blocks (the Block class of requirements as we discussed in section 7.4). They carry out some specific function that takes the values from the ports as an input. In the case of Main Controller the inputs being MC_Rotor Speed and MC_Wind Speed. The outputs are fed to ports leading out of the block and into another block.
An example of such an outgoing port is Pitch Brake. Ingoing ports correspond to the InPort class in section 7.4, whereas outgoing ports correspond to the OutPort class. After the work is done, the engineer uses the Simulink adaptor to publish a new version of the Simulink model to the CMS in the same way as the HP-ALM engineer did in steps 1 through 3.

Steps 9 – 11

Using MDWorkbench, we now run the reverse transformation. That is, from the modified Simulink model back to an HP-ALM model.

![Figure 39. Specifying the Simulink-to-HP transformation inputs and outputs](image)

This time, our input is a Simulink model file, whereas the output is an XML representation of an HP-ALM model. The output is placed into its proper location inside the local file system again.

Launching the HP-ALM adaptor once more and using the ‘Import requirements from file’ functionality, we can see we can see that an additional Block named ‘Back-up Controller’ was created by the Simulink engineer. Using the checkboxes on the left hand side of the table we can choose whether or not to accept this new entity.
Mapping HP-ALM entities to Simulink entities

As we said, we only transform requirements of types Block, InPort, OutPort and Interface to a counterpart in MATLAB Simulink. Accommodating this transformation however was not a straightforward task. Even though the XML files processed by the adaptor in this scenario are very similar to ‘native’ HP-ALM XML, they are slightly modified by the adaptor before being exported and imported. Let us explain what these modifications encompass.

First of all, HP-ALM does not support multi-value attributes. At least, not where the HTTP REST API is concerned. Hence, our original intuition of using the native ‘trace-to’ and ‘trace-from’ fields to represent traces between ports would not work, as ‘trace-to’ and ‘trace-from’ are both multi-value fields. We explained earlier that we therefore introduced a new field ‘outport’ for entities of type InPort. The trouble is however, that on the side of the API, user-defined attributes are assigned a name ‘user-<some identifier>’’. This identifier differs for every HP-ALM project, hence, we would have to modify the adaptor every time the project changes. Fortunately, a separate API call allows us to retrieve the actual name for a user-defined attribute. The adaptor reads this value and then replaces the ‘user-<some identifier>’ attribute name.
A similar problem was the one we faced with regard to different entity types. We introduced several new entity types to accommodate the integration with MATLAB Simulink, but just like with user defined attributes, the HTTP REST API is not as flexible. Rather than appending the actual type name like ‘Functional’ or ‘Block’, the API puts a ‘type-ID’. The mapping of these type-ID’s again is retrieved through a separate API call. What’s more, type-ID’s are also assigned arbitrarily. Hence, the adaptor reads the type-ID, maps it, and writes the actual type name instead. This eliminates project dependency and lowers the burden on the transformation and possibly the MATLAB Simulink engineer. That is to say, depending on how the transformation is implemented, the engineer might have to insert some arbitrary type-ID field value for every entity he creates.

To maintain a consistent mapping between HP-ALM and MATLAB Simulink entities, we decided to use entity names. As both HP-ALM and MATLAB Simulink have their own entity identification schemes, keeping a mapping would in our opinion induce too much overhead. Using entity names however, obviously forces us to introduce the constraint of entity names remaining the same throughout their lifetime. An interesting issue that is now arose is the use of anonymous MATLAB Simulink entities. More precisely, in MATLAB Simulink, an entity can be created without having a name. This wouldn’t introduce issues necessarily, given the transformation is able to handle nameless entities, but it requires some thought should be given to how a consistent mapping can still be guaranteed. Our solution was this: Every time an anonymous entity is created in MATLAB Simulink, instead of using the name of the entity, the transformation instead takes the identifier that was automatically assigned to that entity and generates a name itself. This name is of the format ‘anonymous_<identifier>’. Needless to say, the mapping would have to treat entities with this prefix specially.

Scenario 2: Sending and receiving notifications

Notifications are sent to the OSLC WS endpoint of the adaptor using the issueNotification or updateAvailable function calls listed in Table 4. Notifications through issueNotification are sent by other tool adaptors in the framework. They can contain any kind of message, but are generally expected to contain some comment on the model in its current state. When the user of another tool wants to send a notification to the HP-ALM engineer, he points the tool adaptor to the URL where the HP-ALM/OSLC tool adaptor expects to receive a notification:

http://localhost:8787/Requirements/issueNotification

The notification contains at least the following information:

- Name of the sender
- Role of the sender
- Related IF version
- Description of the issue
- Suggested changes
A typical notification that could be sent through this mechanism is one saying that there is a problem with the model in its latest IF version and how it should be improved.

Notifications sent through the `updateAvailable` function call are of a somewhat different nature. Notifications like this can only be sent through the framework after, chronologically:

1. An OSLC tool adaptor comes online.
2. The adaptor registers itself with the service catalog by calling the `register` function.
3. The users of other tools browse through the service catalog and `subscribe` to the service.
4. A new IF version is published to the CMS.
5. The CMS informs the service catalog.
6. The service catalog calls `updateAvailable` in all the tool adaptors that were subscribed.

The steps are visualized in the sequence diagram of Figure 41.

![Figure 41. Publishing of new IF versions](image)

The essential information that a notification through the `updateAvailable` contains is:

- The tool adaptor from which the update originated.
- The identifier of the IF version that was published.
For both types of notifications, a HTTP POST containing the message will put the message in the notification inbox. Then when the HP-ALM engineer starts up the GUI endpoint and browses to the notifications, the message will be displayed there.

Scenario 3: Browsing using OSLC

Aside from providing an interface to issue notifications to the HP-ALM engineer, the OSLC WS endpoint provides the facilities to browse all existing requirements in HP-ALM using both the OSLC interaction patterns and data formats. But by merely directing our web browser to the URL where the web service is listening, we are being returned the data in a rather user unfriendly manner. The OSLC RDF/XML is just printed flat on the web page. What would be convenient to have is a delegated UI that processes the RDF/XML and present it in a more accessible way.

In addition to the adaptor, we have developed a delegated UI based on Java Servlet technology, that allows the user to browse OSLC RequirementCollections and Requirements in an intuitive way. The delegated UI uses Apache Jena technology to process the RDF/XML returned by the tool adaptor. After processing it passes it on to a Java Server Page that displays the relevant artifacts. The delegated UI was developed specifically for the HP-ALM adaptor, but should connect to other OSLC tool adaptors with only minor modifications. These modifications encompass the handling of, for example, non-OSLC compliant attributes that are nonetheless required in the specific tool chain.
In Figure 42, which is a screenshot taken of the delegated UI, we can see that there are two separate boxes to select RequirementCollections and Requirements respectively. As there is no such thing as a RequirementCollection in HP-ALM, we decided to map the identifier of the RequirementCollections to versions as stored in the aforementioned versions file. Thus, when a call to the delegated UI is made, it retrieves all versions of the HP-ALM model and the Requirements contained within them.

After browsing to the desired RequirementCollection or Requirement and pressing the ‘Submit’ button, the URI of the selected artifact is returned. When merely using a browser, this would not have any result, but if we were to embed the delegated UI into, for instance, a traceability tool, this tool could use the URI returned to create traces between the selected artifact and another artifact.
Integrating without the adaptor

To clarify what is the usefulness of having the HP-ALM/OSLC adaptor, let us look at how the 3 scenarios would have been realized, if even possible, without the HP-ALM/OSLC adaptor.

First off, the adaptor provides a version control mechanism completely independent of HP-ALM’s own versioning scheme. Without this version control mechanism, other tools in the tool framework would not be able to distinguish between different versions of the overall HP-ALM model, as HP-ALM only manages individual requirement versions.

Furthermore, the adaptor translates the HP-ALM model into a common format that is interpretable by the other tools in the framework. Also, it makes the HP-ALM models centrally available by publishing to the OSLC compliant Configuration Management System (CMS). HP-ALM does not allow direct access to its database, so without the adaptor the other tools would not be able to access the model.

The adaptor allows for communication between not only the different tool adaptors and tools, but perhaps even more importantly it allows for communication between the different tool users. By browsing models contained in other tools through the OSLC based delegated UI and being able to send notifications, the engineers are able to synchronize the development effort.
8. Discussion

Based on our experience building the HP-ALM/OSLC adaptor and using it to integrate HP-ALM with other tools, we make an attempt to generalize the challenges that we faced and the solutions we devised to overcome those challenges. The challenges discussed are by no means exhaustive, there undoubtedly exist many more issues for other tool adaptors used in other tool integration frameworks, but the ones listed are merely the most prominent that arose during the development of the HP-ALM/OSLC adaptor.

Tool access

Some tools use their own repository to store the models they manipulate, while others at best allow a user to save the model to a file in the local file system. These repositories can be of varying levels of sophistication. Concurrent access, security management and version control are typical features that a repository could or could not possess, and they should be taken into consideration when developing a tool adaptor.

In the iFEST it is essential that the models are centrally available. The tool adaptors are capable to create models in the local file system in a common format (OSLC, ReqIF [14], etc.), after which these models are uploaded to the central iFEST repository. This way, the internal repository of the tool is merely used by the engineer working directly with that tool, whereas the other tool adaptors are safely shielded from the idiosyncrasies of the tool’s repository. This assumes however the presence of an API that allows the tool adaptor to retrieve the relevant data and that is expected to remain stable. That is, the tool adaptor still functions properly after the tool is updated to a later edition. In the absence of such an API, we have no choice but to resort to, if possible, direct file access (i.e. accessing natively produced files) or direct database access. Both are brittle approaches in the sense that one slight change to the file structure or database layout can break the tool adaptor.

In the case of the HP-ALM/OSLC adaptor fortunately an API was available, that is expected to remain stable. HP-ALM also is capable of producing and importing Excel files, but these have to be very specifically formatted. Also, with administrator rights one could directly access HP-ALM’s SQL based database. Neither of these easier to implement, but brittle approaches was taken for the reasons explained above.

Version management

While strongly related to the first challenge, proper version control and management is greatly emphasized in the iFEST IF. Moreover, the central iFEST repository is an OSLC compliant configuration management system. Although a tool may have its own version control features, it might not be adequate to our needs.

In the case of HP-ALM for instance, versioning is individual to requirements. This means that when a single requirement is updated, the only way of telling that the entire model was therefore also updated, is to inspect that same requirement. This is highly impractical.
Our solution is to keep the internal version management and the version management in the entire framework decoupled. The engineer working with the tool may choose to use the internal version management system or not, but the other tool adaptors are only able to view a version mapping. That is, the moment the engineer publishes a new version to the CMS, a version file maintained on the local file system is updated. This version file contains all the versions and the necessary information it would need to fully reproduce those versions in the common data formats. In the context of HP-ALM, this meant a version identifier accompanied by a list of requirement identifiers and their individual versions. This was discussed in great detail in section 6.2.1.

### Web based or locally installed tool

The deployment of the tool, be it as a Web based tool or a local installation, influences the tool adaptor deployment alternatives. In the first case, especially when (part of) our tool adaptor is implemented as a Web service, as is the case with the HP-ALM/OSLC adaptor, it is convenient to deploy the tool adaptor on a Web server also. Just like the tool itself, the adaptor can be accessed over the network.

In the case of a locally installed tool however, the problem is that a tool adaptor deployed on a Web server would have to reconnect to the tool every time it is deployed on a different machine, and the availability depends on the uptime of that machine. Alternatively, it would have to maintain its own copy of the data that is available on the Web server even when the tool itself is offline.

HP-ALM is a Web based tool, so the (OSLC part of) the adaptor could be very conveniently deployed on a Web server also. Nonetheless, even if HP-ALM was a locally installed tool, we could overcome the practical difficulties thanks to the GUI adaptor.

The GUI adaptor namely is responsible for publishing new versions. That is, the engineer uses the locally run adaptor to publish, so as long as the new version is placed in a central repository somewhere, which the OSLC based Web Service can access, we have circumvented the problem. This is the main reason the adaptor was essentially split into two.

### GUI extensions

In addition to having API’s available that allow for the retrieval and manipulation of their artifacts, some tools allow extensions to be made to their graphical user interface. As to maintain uniformity, we believe that either the full tool adaptor user interface should be implemented as an extension to the tool’s existing functionalities or be kept entirely within its own independent application, as is the case with the GUI endpoint of the HP-ALM/OSLC adaptor. The criteria that apply to the tool extensions are:

- Is the entire range of API functionalities available?
- Is it possible to access the local file system, so that files can be written?
In the case of HP-ALM, GUI extensions were not possible at all. The original intention was to create new functionalities within HP-ALM’s GUI to allow the engineer to, for instance, publish new IF versions or receive and send notifications, but this was not accommodated. This was another important reason to split the tool adaptor into two; one Web Service endpoint intended for users of other tools that are interested in HP-ALM but not initiated into the tool specifics, another GUI endpoint that is locally run and intended to be an extension – albeit not literally – to HP ALM’s functionalities.

Identifiers

Every tool uses its own scheme to distinguish between artifacts and handle their modification, creation and deletion appropriately. Some tools allow their users to assign identifiers themselves, some assign them automatically. Paramount is that it should be possible to maintain traces between artifacts within different tools.

In the case of transformations between HP-ALM to MATLAB Simulink and vice versa, this proved somewhat difficult. The problem was that HP-ALM does not allow the assignment of identifiers, but assigns identifiers to artifacts itself at the moment of their creation. This meant that when an artifacts was newly created in Simulink and subsequently transformed to its HP-ALM counterpart, HP-ALM would assign its own identifier different from the one in Simulink. There are two alternatives that were considered:

- Introducing a new identifier field in both tools.
- Requiring the artifact name never to change.

In the first case it would have to be possible to create our own artifacts fields in both tools. However, even if it were, if one of the two tools would ever be replaced, the replacing tool might not possess this feature. Thus, the only option that remains is requiring artifact names never to change. Although it may seem constraining, this is the only alternative that guarantees consistency in the face of changing tools.

There is a third alternative, that in our case was not yet available, and that is a traceability tool. That is, a third party in between the two tools under consideration keeps track of which artifact in tool A matches which artifact in tool B. This traceability tool would have be rather sophisticated however to keep track of changing artifacts, and might be faced with the same aforementioned problems.
Relationships

In the Wind Turbine industrial case study it was essential that structural requirements are somehow traceable to each other. Traces in this context simply indicate a relationship from one structural requirement to the other. However, some tools do not natively support the creation of relationships between artifacts. In HP-ALM for instance, multi-value attributes are not supported nor can we make association classes. The solution to this is to either create a new field that mimics a multi-value attribute or let a third party handle relationships. In our case, the multiplicity of the traces was one-to-many. This meant that we could use a single-value attribute that would for every artifact on the ‘many’ side of the relationship indicate which artifact traces to it by specifying a single identifier. This however would not work in case of many-to-many relationships; a true multi-value attribute should be used.

If a traceability tool would have been available, it would not have been necessary to define a new custom attribute. As this traceability tool is able to create relationships between artifacts in different tools, it should also be capable of creating relationships between artifacts in the same tool. If a traceability tool is available, this should be the preferred approach as it shields us from the intricacies of modifying a tool’s metamodel to suit our needs.

Metamodel modifications

Very rarely are the artifacts that tools manipulate completely independent. Requirements authoring tools manipulate more kinds of artifacts than merely requirements, for instance. The requirements are related to design components, tests and other requirements. A proper tool therefore allows its users to create custom artifact types to which the core artifacts can be related. In the case of HP-ALM and the Wind Turbine industrial case study, this meant the creation of several different new artifacts types, among which logical blocks and ports.

The trouble however was that, despite HP-ALM providing at least the feature to create custom artifact types, the API only would return a seemingly random ‘type-id’ that represented the type of an artifact, rather than the actual name. This resulted in a separate API call that mapped the ID to its respective type name.

This is but an example of one of the many difficulties that we may face when trying to modify a tool’s metamodel. If they are at all possible, these modifications may not have the effect we would like them to have. In HP-ALM for instance, the metamodel is extremely simple and allows for additions, but is so simple that it becomes inflexible. This, for the developer of the tool, keeps the logic needed to handle the metamodel manageable. Or put differently, a simple metamodel leads to simple models. On the other hand, it potentially increases the effort to bend the tool to our needs and develop a tool adaptor for it.

What’s more, tools that provide facilities to modify their metamodel(s) are a rarity, because of the increasing logic that is needed to make the tool able to handle the increasingly complex models.
9. Conclusions

In this work we have discussed in detail the challenges faced and solutions found during the implementation of a tool adaptor for the Requirements Management module of HP Application Lifecycle Management within the context of the ARTEMIS iFEST project. The goal of the ARTEMIS iFEST project, or simply iFEST project, is to specify a tool integration framework for heterogeneous, multi-core embedded systems tools. This framework is referred to as the iFEST IF.

The reason we desire tool integration is for the development process to be more effective - to deliver the right system within a set time and budget. To this end, we need some guarantee on tool consistency. That is, every tool used should maintain data that represents the exact same system being developed using the exact same process. For instance, the system design should match the initial requirements, the implementation should match the design and the implementation should be validated using an appropriate set of test cases.

The iFEST IF aims to adopt Open Services for Lifecycle Collaboration (OSLC), a set of specifications aimed at standardizing the way software lifecycle tools share data. Central to OSLC is the concept of tool adaptors implemented as Web services. That is, the component that connects the tool to the framework (the tool adaptor) is an OSLC Service Provider that exposes its data using a common format. iFEST may however deviate from OSLC when necessary.

An industrial case study called the Wind Turbine IC was carried out in order to test the validity of the iFEST IF. One of the tools used in the Wind Turbine IC is HP-ALM. HP-ALM was integrated to other tools, primarily MATLAB Simulink, through the use of the HP-ALM/OSLC adaptor. The industrial case study included several different integration scenarios, among which a scenario that included a transform to MATLAB Simulink models. Because of the complexity of these transformations and other reasons, it was decided to split the HP-ALM adaptor into two parts.

The first part of the adaptor, known as the OSLC Web Service endpoint, takes requirements from HP-ALM and exposes them as OSLC resources through an OSLC Web service. To accommodate the desired range of attributes however, the OSLC Requirements model was extended to include new attributes.

The second part of the adaptor, known as the GUI endpoint, allows for the extraction of requirements from HP-ALM as XML files and their insertion into an OSLC compliant configuration management system. It is a locally run application based on the Java Swing graphical library. The reason for the inception of this endpoint was that OSLC was deemed too verbose to sensibly perform model transformations to MATLAB Simulink. For instance, OSLC uses URI’s as links between resources. This would mean that a transformation, in order to traverse the entire requirements model, would have to follow these links. With XML files that represent the entire requirements model, this no longer necessary.

In the development of the adaptor, we have faced several challenges. The challenges that are faced in developing tool adaptors for the iFEST IF can be anticipated by inspecting certain tool characteristics.
The challenges are by no means exhaustive and there undoubtedly exist many more. We merely summarize the challenges we faced in our particular context and attempt to generalize them. We identified seven challenges that took significant effort to solve.

- Tool access
- Version management
- Web based or locally installed tool
- GUI extensions
- Identifiers
- Relationships
- Metamodel modifications

We accessed HP-ALM through an API it provided, shielding us from the specifics of local files or databases. The API is expected to be more stable and should be the preferred entry point if a choice exists.

We kept the version management system of the iFEST IF completely decoupled from the internal version management system of HP. This would allow us to have the versions that the engineer working with HP sees and the versions that are visible for the entire framework separate. This way, the engineer can make changes to the models without being forced to publish them.

The issue of Web based and locally installed tools was solved by splitting the adaptor into two. One part that is locally run by the HP-ALM engineer, another part deployed as Web Service and accessed by the other parties in the framework. Another reason to split was because we desired to extend the functionalities provided by HP and locally run application was the most convenient way to do this.

Names were used as unique identifiers for artifacts, but basically any attribute with a stable value can be used. Ideally, a third party traceability tool should be used to minimize restrictions on the tool.

Finally, as long as tools use their own idiosyncratic metamodels, arbitrary development effort is needed to build a tool adaptor. Tools should allow extensions or even modifications to their metamodels, but this also increases the complexity of the tool itself.

In the development of the HP-ALM/OSLC adaptor we feel that we overcame the challenges and managed to successfully integrate HP-ALM with the other tools in the iFEST IF. We believe that any tool can be integrated into the iFEST IF, as long as it provides appropriate, stable access mechanisms and we can find an appropriate mapping to the common (OSLC) formats.

Regarding an adoption of OSLC, we believe that OSLC provides a great deal of flexibility for the loose integration of tools, but is too crude to suit the needs of tool chains that require more tight integration. That is to say, the specific tools of the Wind Turbine tool chain for instance, required the tool adaptors to be involved in critical model transformations, whereas to fully represent the models would overstretch the OSLC metamodels. Despite those metamodels being highly extendable.

What’s more, it is also not always pragmatic to set up a fully functioning Web service for every tool, especially tools that are run locally.
Despite efforts such as OSLC, there is still a lot of effort involved in integrating tools. A proprietary, rather closed tool such as HP-ALM requires a considerable amount of development effort. What’s more, without any standardization it is hard to estimate beforehand how difficult it is to develop a tool adaptor. As Wasserman [1] indicated already, tools should be built from the ground up with tool integration in mind, rather than tool integration being an afterthought.
References


Appendices

Appendix A. Full Wind Turbine Requirement Specification

Note: The requirements specification below was exported to Microsoft Excel using IRQA.

<table>
<thead>
<tr>
<th>Code</th>
<th>Name</th>
<th>Priority</th>
<th>Source</th>
<th>Stability</th>
<th>ReqType</th>
<th>Blocks</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR-T-001</td>
<td>Sensors for measuring wind velocity</td>
<td>1</td>
<td></td>
<td>Not finalized</td>
<td>OtherReq</td>
<td></td>
<td>There shall be sensors to measure wind velocity (speed and direction)</td>
</tr>
<tr>
<td></td>
<td>PR-T-002 Output power DC</td>
<td>1</td>
<td></td>
<td>Not finalized</td>
<td>OtherReq</td>
<td></td>
<td>The generator must be able to produce DC power of max 80 W (nom 25) at 12V DC.</td>
</tr>
<tr>
<td></td>
<td>PR-T-003 Output Power AC</td>
<td>2</td>
<td></td>
<td>Not finalized</td>
<td>OtherReq</td>
<td></td>
<td>The generator shall be able to generate 12 V AC power</td>
</tr>
</tbody>
</table>
### Interface

| PR-F-003 | Propeller axle | 1 | iFES Project Coordinator | Stable | OtherReq | Functional High Level Requirements Plant Requirements | It shall be possible to stop and lock the propeller axle. |

### Control System

### Pitch Control

### AC-Interface

| PR-F-001 | Pitch Control | 1 | iFES Project Coordinator | Ready for review | OtherReq | Control System Functional High Level Requirements Pitch Control Requirements | It shall be possible to control pitch of the blades during operation. |

### Protection system

| PR-F-012 | External Internal Failure | 1 | iFES Project Coordinator | Not finalized | OtherReq | Control System Functional High Level Requirements Protection system Requirements | The protection functions shall be activated as a result an internal or external failure or dangerous event. |

| PR-F-013 | Control Function Failure | 1 | iFES Project Coordinator | Ready for review | OtherReq | Control System Functional High Level Requirements Protection system Requirements | The protection functions shall be activated when control function fails. |

| PR-F-016 | Braking System Functionality | 1 | iFES Project Coordinator | Not finalized | OtherReq | Control System Functional High Level Requirements Protection system Requirements | The braking system shall be able to bring the rotor to idling mode or complete stop from any operation condition. |

<p>| PR-F-018 | Power Supply Failure | 1 | iFES Project Coordinator | Not finalized | OtherReq | Control System Functional High Level Requirements Protection | Brakes shall be designed to function even if their external power supply fails. |</p>
<table>
<thead>
<tr>
<th>Process Number</th>
<th>Description</th>
<th>Object</th>
<th>Other Req</th>
<th>System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR-F-019</td>
<td>Braking System Functionality on Rotor</td>
<td>1</td>
<td>Not finalized</td>
<td>Control System Functional High Level Requirements Protection system Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A brake shall be able to keep the rotor in the full stop position for the defined wind</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>conditions for at least one hour after the brake is applied.</td>
</tr>
<tr>
<td>PR-F-020</td>
<td>Monitoring of Braking system</td>
<td>1</td>
<td>Not finalized</td>
<td>Control System Functional High Level Requirements Protection system Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The remaining life of any component subject to wear in the brake system shall be</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>monitored automatically and subject to regular inspection.</td>
</tr>
<tr>
<td>PR-F-022</td>
<td>Disconnection from grid</td>
<td>2</td>
<td>Not finalized</td>
<td>Control System Functional High Level Requirements Protection system Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>It shall be possible to disconnect a wind turbine electrical system from all</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>electrical sources of energy as required for maintenance or testing.</td>
</tr>
<tr>
<td>PR-T-012</td>
<td>Emergency Button</td>
<td>1</td>
<td>Not finalized</td>
<td>Control System Functional High Level Requirements Protection system Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The emergency stop button shall have higher priority than other control functions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>in accessing the braking systems and equipment for network disconnection when</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>triggered..</td>
</tr>
<tr>
<td>PR-T-019</td>
<td>Priority of Protection functions</td>
<td>1</td>
<td>Not finalized</td>
<td>Control System Functional High Level Requirements Protection system Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The protection functions shall have higher priority in accessing the braking</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>systems and equipment for network disconnection when triggered.</td>
</tr>
<tr>
<td>Main Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC_INTERFACE</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR-F-004</td>
<td>Reaction to failure</td>
<td>1</td>
<td>Not finalized</td>
<td>Functional High Level Requirements Requirements</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The system must react to imminent failures to control or actuation systems within</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>the turbine to prevent the</td>
</tr>
<tr>
<td>Requirement</td>
<td>Type</td>
<td>Description</td>
<td>Status</td>
<td>Other Requirements</td>
</tr>
<tr>
<td>-------------</td>
<td>------</td>
<td>-------------</td>
<td>--------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Non-Functional Performance</td>
<td>PR-T-005</td>
<td>Wind speed sustainability</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>Non-Functional Performance</td>
<td>PR-T-008</td>
<td>Minimum propeller blade pitch</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>Non-Functional Performance</td>
<td>PR-T-009</td>
<td>Minimum propeller speed</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>Safety</td>
<td>PR-T-010</td>
<td>Self contained safety functions</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>Safety</td>
<td>PR-T-011</td>
<td>Withstand wind conditions</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>Standards</td>
<td>PR-E-001</td>
<td>Noise</td>
<td>1</td>
<td>iFEST project coordinator</td>
</tr>
<tr>
<td>PR-T-007</td>
<td>IEC61400 1</td>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>--------------------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>G.Toper</td>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>iFES project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
<td>Design High Level Requirements Non-Functional Requirements</td>
</tr>
<tr>
<td>PR-F-028</td>
<td>Sensors measuring angle of blade</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------</td>
<td>---</td>
<td>--------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Blade</td>
<td>Blade twist</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
</tr>
<tr>
<td>Generator_1</td>
<td>Generator_Input_1</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
</tr>
<tr>
<td>Generator_2</td>
<td>Generator_Input_2</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
</tr>
<tr>
<td>Generator_Output_1</td>
<td>Generator_Output_1</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
</tr>
</tbody>
</table>

Functional Requirements shall be disconnected in the event of loss of network power.
<p>| PR-F-077 | Generator_Output_2 | 1 | ifES T project coor dinator | Not finalized | OutPort | Functional Generator Generator_ Interface Plant Requirements System Requirements | The output2 of generator is the current (i) supplied to the wind turbine. |
| PR-F-078 | Generator_Output_3 | 1 | ifES T project coor dinator | Not finalized | OutPort | Functional Generator Generator_ Interface Plant Requirements System Requirements | The output3 of generator is Voltage (u). |
| PR-F-079 | Generator_Output_4 | 1 | ifES T project coor dinator | Not finalized | OutPort | Functional Generator Generator_ Interface Plant Requirements System Requirements | The output4 of generator is the angular speed of the rotor (Omega_g). |
| Servo | Servo_Interface | | | | | | |
| PR-F-084 | Servo_Input_1 | 1 | ifES T project coor dinator | Not finalized | InPort | Functional Plant Requirements Servo Servo_Interface System Requirements | The input1 to servo is the setpoint. |
| PR-F-085 | Servo_Output_1 | 1 | ifES T project coor dinator | Not finalized | InPort | Functional Plant Requirements Servo Servo_Interface System Requirements | The output1 of servo is the pitch (theta). |</p>
<table>
<thead>
<tr>
<th>PR-F-080</th>
<th>Rotor Input_1</th>
<th>1</th>
<th>iFES Project Coordinator</th>
<th>Not finalized</th>
<th>InPort</th>
<th>Functional Plant Requirements Rotor Rotor_Interface System Requirements</th>
<th>The input1 to rotor is wind speed or V.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PR-F-081</td>
<td>Rotor Input_2</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>Functional Plant Requirements Rotor Rotor_Interface System Requirements</td>
<td>The input2 to rotor is the pitch (theta).</td>
</tr>
<tr>
<td>PR-F-082</td>
<td>Rotor Input_1</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>Functional Plant Requirements Rotor Rotor_Interface System Requirements</td>
<td>The input3 to rotor is the rotor speed (omega).</td>
</tr>
<tr>
<td>PR-F-083</td>
<td>Rotor Output_1</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>Functional Plant Requirements Rotor Rotor_Interface System Requirements</td>
<td>The output1 of rotor is the torque of the rotor (T_rotor).</td>
</tr>
<tr>
<td>Control System</td>
<td>Pitch Control</td>
<td>AC-Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR-T-050</td>
<td>AC Input_1</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>AC-Interface Control System Functional Pitch Control Requirements System Requirements</td>
<td>The output of actuation controller shall be pitch command.</td>
</tr>
<tr>
<td>PR-T-053</td>
<td>AC Input_4</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>AC-Interface Control System Functional Pitch Control Requirements System Requirements</td>
<td>The input to the actuation controller shall be wind speed.</td>
</tr>
<tr>
<td>PR-T-054</td>
<td>AC_Input_5</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>AC-Interface Control System Functional Pitch Control Requirements System Requirements</td>
<td>The input to the actuation controller shall be rotor speed.</td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td>---</td>
<td>--------------------------</td>
<td>-------------</td>
<td>--------</td>
<td>-------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>PR-T-055</td>
<td>AC_Input_6</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>AC-Interface Control System Functional Pitch Control Requirements System Requirements</td>
<td>The input to the actuation controller shall be pitch for braking.</td>
</tr>
<tr>
<td>PR-T-056</td>
<td>AC_Input_7</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>AC-Interface Control System Functional Pitch Control Requirements System Requirements</td>
<td>The input to the actuation controller shall be the state of the MC (park).</td>
</tr>
<tr>
<td>PR-F-024</td>
<td>Adjust pitch angel</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Pitch Control Requirements System Requirements</td>
<td>The pitch control should adjust pitch angle to regulate rotational speed.</td>
</tr>
<tr>
<td>PR-F-029</td>
<td>Determine the pitch command</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Pitch Control Requirements System Requirements</td>
<td>The pitch control system should determine the pitch command or the angle to which the blade should be turned.</td>
</tr>
<tr>
<td>PR-F-030</td>
<td>Reaction to system failure</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Pitch Control Requirements System Requirements</td>
<td>The controller must ensure that in an event of system failure the blades must rotate to a safe position.</td>
</tr>
<tr>
<td>PR-F-031</td>
<td>Angle comparison</td>
<td>1</td>
<td>iFES Project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Pitch Control Requirements System Requirements</td>
<td>The pitch control system must subtract the desired angle of attack from the inflow angle or angle at which the wind strikes the blade.</td>
</tr>
<tr>
<td>PR-F-059</td>
<td>Pitch command for parking</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Pitch Control Requirements System Requirements</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>--------</td>
<td>--------</td>
<td>--------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Protection system</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the wind turbine is in braking state and the inflow angle is more than desired angle of attack then the wind turbine must send pitch for braking. If the wind turbine is in park state then the controller sends a pitch command for park.</td>
<td></td>
</tr>
<tr>
<td>PR-F-049</td>
<td>Working pressure</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Protection system Requirements System Requirements</td>
<td></td>
</tr>
<tr>
<td><strong>Main Control MC_Interface</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The brakes must be able to apply working pressure between 140-150 bar.</td>
<td></td>
</tr>
<tr>
<td>PR-T-047</td>
<td>Input 1</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The input to the MC is wind speed.</td>
<td></td>
</tr>
<tr>
<td>PR-T-048</td>
<td>Output 1</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The output of the MC system is turbine state.</td>
<td></td>
</tr>
<tr>
<td>PR-T-057</td>
<td>Input 2</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>InPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>The input to the MC is turbine speed.</td>
<td></td>
</tr>
<tr>
<td>PR-T-058</td>
<td>Output 2</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>The output of the MC system is generator trip.</td>
</tr>
<tr>
<td>PR-T-059</td>
<td>Output 3</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>The output of the MC system is parking brake.</td>
</tr>
<tr>
<td>PR-T-060</td>
<td>Output 4</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OutPort</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>The output of the MC system is pitch brake.</td>
</tr>
<tr>
<td>PR-F-032</td>
<td>Functionality of SCS</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>The main control system must analyse operating conditions of turbine.</td>
</tr>
<tr>
<td>PR-F-033</td>
<td>Switch states of WT</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>The main control system must switches the wind turbine between park, startup, generation and brake stages depending on the wind turbine conditions.</td>
</tr>
<tr>
<td>PR-F-034</td>
<td>Park state operation</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>In park stage the park brake must be turned on, the pitch brake must be turned on and the turbine state must be 0.</td>
</tr>
<tr>
<td>PR-F-035</td>
<td>Switching to startup stage</td>
<td>1</td>
<td>iFES Project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control MC_Interface Requirements System Requirements</td>
<td>If the wind speed is greater than cut in speed and less than cut out speed then the control system should switch the wind turbine to the startup stage.</td>
</tr>
<tr>
<td>PR-F-036</td>
<td>Start up stage operation</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>In startup stage the park brake must be turned off, pitch brake must be turned off and the turbine state must be 1.</td>
</tr>
<tr>
<td>PR-F-038</td>
<td>Generating stage operation</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>In generating stage the park brake is turned off, the pitch brake is turned off and the turbine state must be 2.</td>
</tr>
<tr>
<td>PR-F-039</td>
<td>Switching to brake stage</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>If the wind speed is less than cut in lower wind speed or wind speed is greater than cut out wind speed or turbine speed is less than turbine speed cut in or turbine speed is greater than turbine speed cut out then the controller switches the wind turbine to brake stage from generating stage.</td>
</tr>
<tr>
<td>PR-F-040</td>
<td>Brake stage operation</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>In the brake stage the park brake should be turned off, the pitch brake should be turned on and the turbine state must be 3.</td>
</tr>
<tr>
<td>PR-F-041</td>
<td>Switch to park stage</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>If the turbine speed is less than or equal to the park speed speed then the controller should switch the wind turbine park stage.</td>
</tr>
<tr>
<td>PR-F-047</td>
<td>Annual average wind speed</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support annual average wind speed of 8.5 m/s.</td>
</tr>
<tr>
<td>PR-F-048</td>
<td>Wind shear</td>
<td>1</td>
<td>ifES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support wind shear of 0.2.</td>
</tr>
<tr>
<td>PR-F-050</td>
<td>Extreme wind conditions</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must be able to support extreme wind condition of 42.5 m/s.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PR-F-051</td>
<td>Survival wind speed</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must have a survival wind speed 59.5 m/s.</td>
</tr>
<tr>
<td>PR-F-052</td>
<td>Automatic stop limit</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support automatic stop limit of 20 m/s.</td>
</tr>
<tr>
<td>PR-F-053</td>
<td>Re-cut</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support Re-cut in 18 m/s.</td>
</tr>
<tr>
<td>PR-F-054</td>
<td>Characteristics turbulence</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support a characteristic turbulence intensity of 16%.</td>
</tr>
<tr>
<td>PR-F-055</td>
<td>Maximum inflow angle</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must tolerate Maximum inflow angle of 8 degrees.</td>
</tr>
<tr>
<td>PR-F-68</td>
<td>Wind shear</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>The controller must support a wind shear of 0.2.</td>
</tr>
<tr>
<td>PR-F-69</td>
<td>Switching to generation stage</td>
<td>1</td>
<td>iFEST project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>If the turbine speed is greater than cut in speed of the turbine then the controller must switch the turbine to generating stage from start up stage.</td>
</tr>
<tr>
<td>PR-F-77</td>
<td>Switch to startup stage</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>If the wind speed is greater than the cut in lower wind speed and the wind speed is less than cut out wind speed then the controller must switch the wind turbine from park stage to start up stage.</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------</td>
<td>---</td>
<td>---------------------------</td>
<td>--------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PR-F-037</td>
<td>Switch between startup and brake stage</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Control System Functional Main Control Requirements System Requirements</td>
<td>If the wind speed is less than cut in upper wind speed and the wind speed is less than cut in lower wind speed then the wind turbine switches between startup and brake stage.</td>
</tr>
<tr>
<td>Non-Functional Performance</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PR-T-023</td>
<td>Step input response</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Non-Functional Performance Requirements System Requirements</td>
<td>With the step input the pitch control system response should be within the range of 0 to 2.5 sec.</td>
</tr>
<tr>
<td>PR-T-024</td>
<td>Force by the actuator</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Non-Functional Performance Requirements System Requirements</td>
<td>The amount force the pitch actuator must provide is 2.5e7 Pa.</td>
</tr>
<tr>
<td>PR-T-025</td>
<td>Nacelle speed</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Non-Functional Performance Requirements System Requirements</td>
<td>The wind turbine must maintain a nacelle speed limit of 10 deg/s.</td>
</tr>
<tr>
<td>PR-T-061</td>
<td>Nacelle rotation</td>
<td>1</td>
<td>iFEST project Coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Non-Functional Performance Requirements System Requirements</td>
<td>The nacelle shall be able to turn at least 360 degrees.</td>
</tr>
<tr>
<td>Safety Standards</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>PR-T-020</td>
<td>Mechanical linkage</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PR-T-026</td>
<td>Blade length</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The blade length shall be 40 meters.</td>
</tr>
<tr>
<td>PR-T-027</td>
<td>Blade material</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The blade will be made of wood.</td>
</tr>
<tr>
<td>PR-T-028</td>
<td>Blade color</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The color of the blades shall be RAL 7035 and the gloss will be of class 2.</td>
</tr>
<tr>
<td>PR-T-029</td>
<td>Type of rotor air brake</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>Full blade must be used for the wind turbine.</td>
</tr>
<tr>
<td>PR-T-030</td>
<td>Largest chord</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The wind turbine must have a largest chord of 3.08 meters.</td>
</tr>
<tr>
<td>PR-T-031</td>
<td>Blade profiles</td>
<td>1</td>
<td>ifEST proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The blade profile of the wind turbine must be FFA - W3, NACA 63.4.</td>
</tr>
<tr>
<td>PR-T-032</td>
<td>Type description</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The brake must be an active brake.</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PR-T-033</td>
<td>Brake disk</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The brake disk must be made of steel and should be mounted on high speed shaft</td>
</tr>
<tr>
<td>PR-T-034</td>
<td>Number of calipers</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The brake must have two calipers.</td>
</tr>
<tr>
<td>PR-T-035</td>
<td>Voltage</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The brake must operate on a voltage of 3 x 480 V.</td>
</tr>
<tr>
<td>PR-T-036</td>
<td>Oil capacity of the brakes</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The oil capacity of the brakes must be 11 l.</td>
</tr>
<tr>
<td>PR-T-037</td>
<td>Accumulator Capacity</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The accumulator capacity of the pitch actuation system must be 0.1L.</td>
</tr>
<tr>
<td>PR-T-038</td>
<td>Yaw actuation type</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The yaw actuation system must be of planetary gear motor type.</td>
</tr>
<tr>
<td>PR-T-041</td>
<td>Gear ratio of yaw gear unit</td>
<td>1</td>
<td>iFES T proje ct coordinat or</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The gear ratio of yaw gear unit shall be approximately 1:1687.</td>
</tr>
<tr>
<td>PR-T-042</td>
<td>Voltage of yaw actuation</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The voltage of yaw actuation must be 3 x 480 V.</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
<td>---</td>
<td>-----------------------------</td>
<td>--------------</td>
<td>----------</td>
<td>-------------------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>PR-T-043</td>
<td>Number of yaw gears</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The yaw actuation shall have four pieces of yaw gears.</td>
</tr>
<tr>
<td>PR-T-044</td>
<td>Number of Yaw Friction Units</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The yaw brake must have six pieces of yaw friction unit.</td>
</tr>
<tr>
<td>PR-T-046</td>
<td>Oil capacity of yaw brake</td>
<td>1</td>
<td>iFES T project coordinator</td>
<td>Not finalized</td>
<td>OtherReq</td>
<td>Design Non-Functional Requirements System Requirements</td>
<td>The oil capacity of the yaw gear must be approximately 10 l.</td>
</tr>
</tbody>
</table>