Safety Certification of Software-Intensive Systems
with Reusable Components

<table>
<thead>
<tr>
<th>Report type</th>
<th>Deliverable D611.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Report name</td>
<td>SafeCer Tool Framework and Other Domains tools integration (including guidelines)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dissemination level</th>
<th>PU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Version number</td>
<td>1.0</td>
</tr>
<tr>
<td>Date of preparation</td>
<td>2013-10-31</td>
</tr>
<tr>
<td>Name</td>
<td>E-mail</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>Intecs, Stefano Puri (Editor)</td>
<td><a href="mailto:stefano.puri@intecs.it">stefano.puri@intecs.it</a></td>
</tr>
<tr>
<td>Intecs, Silvia Mazzini</td>
<td><a href="mailto:silvia.mazzini@intecs.it">silvia.mazzini@intecs.it</a></td>
</tr>
<tr>
<td>AIT, Rupert Schlick</td>
<td><a href="mailto:Rupert.Schlick@ait.ac.at">Rupert.Schlick@ait.ac.at</a></td>
</tr>
<tr>
<td>AIT, Egbert Althammer</td>
<td><a href="mailto:Egbert.Althammer@ait.ac.at">Egbert.Althammer@ait.ac.at</a></td>
</tr>
<tr>
<td>AIT, Thomas Gruber</td>
<td><a href="mailto:Thomas.Gruber@ait.ac.at">Thomas.Gruber@ait.ac.at</a></td>
</tr>
<tr>
<td>Reviewer 1 – ULMA,</td>
<td>E-mail: <a href="mailto:oberreteaga@ulmaembedded.com">oberreteaga@ulmaembedded.com</a></td>
</tr>
<tr>
<td>Oskar Berreteaga</td>
<td></td>
</tr>
<tr>
<td>Reviewer 2 – UOM,</td>
<td>E-mail: <a href="mailto:millarramendi@mondragon.edu">millarramendi@mondragon.edu</a></td>
</tr>
<tr>
<td>Miren Illarramendi</td>
<td></td>
</tr>
</tbody>
</table>
## Revision chart and history log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>2013-07-08</td>
<td>Template of the document</td>
</tr>
<tr>
<td>0.2</td>
<td>2013-09-06</td>
<td>New section covering guidelines has been added</td>
</tr>
<tr>
<td>0.3</td>
<td>2013-10-18</td>
<td>New section about guidelines and additional tools have been added</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added contribution from AIT, in particular regarding sections 3.5.1 and 5.</td>
</tr>
<tr>
<td>1.0</td>
<td>2013-10-31</td>
<td>Updated according to the internal review comments.</td>
</tr>
</tbody>
</table>
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD</td>
<td>Block Definition Diagram</td>
</tr>
<tr>
<td>CAR</td>
<td>Certification Artifact Repository</td>
</tr>
<tr>
<td>CHESS</td>
<td>Composition with Guarantees for High-Integrity Embedded Software Component Assembly</td>
</tr>
<tr>
<td>CTF</td>
<td>Certification Tool Framework</td>
</tr>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>FoReVer</td>
<td>Functional Requirements and Verification Techniques for the Software Reference Architecture</td>
</tr>
<tr>
<td>GPM</td>
<td>Generic Process Model</td>
</tr>
<tr>
<td>GSN</td>
<td>Goal Structuring Notation</td>
</tr>
<tr>
<td>IBD</td>
<td>Internal Block Diagram</td>
</tr>
<tr>
<td>MARTE</td>
<td>Modeling and Analysis of Real-time Embedded Systems</td>
</tr>
<tr>
<td>MAST</td>
<td>Modeling and Analysis Suite for Real-Time Applications</td>
</tr>
<tr>
<td>MDA</td>
<td>Model Driven Architecture</td>
</tr>
<tr>
<td>OCL</td>
<td>Object Constraint Language</td>
</tr>
<tr>
<td>OCRA</td>
<td>Othello Contract Refinement Analysis</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OSLC</td>
<td>Open Services for Lifecycle Collaboration</td>
</tr>
<tr>
<td>SACM</td>
<td>Structured Assurance Case Metamodel</td>
</tr>
<tr>
<td>SCTF</td>
<td>SafeCer Tool Framework</td>
</tr>
<tr>
<td>SFTA</td>
<td>Software Fault Tree Analysis</td>
</tr>
<tr>
<td>SysML</td>
<td>Systems Modeling Language</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>WCET</td>
<td>Worst Case Execution Time</td>
</tr>
</tbody>
</table>
Table of contents

Authors .............................................................................................................................. 2
Revision chart and history log ........................................................................................ 3
List of Abbreviations ...................................................................................................... 4
Table of contents ........................................................................................................... 5
Table of figures and tables ............................................................................................. 6
1 Introduction ................................................................................................................ 7
2 Input from previous pSafeCer and nSafeCer work ...................................................... 8
3 Using UML to support the SafeCer process model activities ....................................... 9
  3.1 About model based design with UML ................................................................... 9
    3.1.1 UML, a semi-formal language ..................................................................... 10
  3.2 Requirement definition ......................................................................................... 11
  3.3 Contracts definition .............................................................................................. 12
  3.4 Design .................................................................................................................. 15
    3.4.1 Functional Architecture .............................................................................. 16
    3.4.2 Logical Architecture ................................................................................. 17
    3.4.3 Physical Architecture .................................................................................. 17
    3.4.4 Tracing between the different design levels ............................................... 18
    3.4.5 From software model to code ...................................................................... 18
  3.5 Verification ............................................................................................................ 19
    3.5.1 MoMuT ........................................................................................................ 19
  3.6 Certificate Preparation and Argumentation ........................................................... 20
4 Complementary Specific Modeling Techniques and Tools ........................................... 24
  4.1 SCADE .................................................................................................................. 24
    4.1.1 SCADE Overview ...................................................................................... 24
    4.1.2 SCADE gateways ...................................................................................... 25
    4.1.3 SCADE and UML/SysML ....................................................................... 26
  4.2 Matlab/Simulink ................................................................................................... 26
    4.2.1 Matlab/Simulink Overview ....................................................................... 26
    4.2.2 Simulink and SysML/UML ...................................................................... 29
5 Tool’s Integration Approaches .................................................................................... 34
6 Contribution to overall SafeCer objectives ................................................................ 35
7 References ................................................................................................................ 36
Table of figures and tables

Figure 1- SafeCer activity pattern ................................................................. 9
Figure 2: UML ports delegation ................................................................. 11
Figure 3: Contract modelling ................................................................. 13
Figure 4: Instantiating a contract in a component .................................. 14
Figure 5: Reusing a contract ................................................................. 14
Figure 6: Tracing FormalProperty to Requirement .................................. 15
Figure 7: Using the SysML Rationale (from SysML specification) .......... 17
Figure 8: tracing between different levels ............................................. 18
Figure 9: MoMuT::UML ........................................................................... 20
Figure 10: SACM metamodel excerpt ................................................. 22
Figure 11: SafeCer Component Model related to Contracts, properties and argumentation .... 23
Figure 12: SCADE support to the SafeCer activity pattern .................. 26
Figure 13: Simulink SystemRequirement Block .................................. 28
Figure 14: Simulink and DOORS relations ........................................... 29
Figure 15: SysML/UML vs Matlab/Simulink ....................................... 30
Figure 16: Common language from SysML and Simulink ................... 31
Figure 17: From SysML to Simulink ...................................................... 31
Figure 18: Activity modeling numerical solution .................................. 32
Figure 19: Matlab/Simulink support to the SafeCer activity pattern ........ 33
1 Introduction

One of the main goals of the SafeCer project is to provide support for the SafeCer component-based certification process application. The technological part of the proposed solution is represented by the SafeCer Tool Framework (SCTF) which includes the Certification Tools Framework (CTF) [5] and the Certification Artifact Repository (CAR) [6]; the proposed framework makes it possible to integrate and manually/automatically orchestrate already existing tools to properly support the different process model activities [4] and to realize the so-called continuous certification.

The integration of an initial set of tools into the SCTF has already been addressed in the SafeCer project [8]. The goal of this deliverable is to investigate the integration of new cross-domain languages and tools which can be useful in SafeCer.

In particular the need of integrating the UML language [11] and editors in the SCTF has been highlighted as useful for the healthcare domain\(^1\) and corresponding demonstrator. Taking into account the aforementioned requirement this deliverable provides guidelines for the proper instantiation of the UML language and tools with regards to not only the SCTF but also to the SafeCer generic process model activities [4]; standard extensions of the UML (e.g. SysML [14]) will be also referred.

Other modeling languages and tools commonly used across the different domains have been also considered and their relationship\(^1\) integration with respect to the UML and the SafeCer tool platform has been taken into account.

In summary, the content of this deliverable is the following:

Section 2 provides a list of previous results/deliverables of the SafeCer project which are used as an input to this deliverable.

Section 3 provides guidelines for the application of UML and UML tools to support the SafeCer process model activities.

Section 4 introduces other common languages and tools which can be considered as complementary to the UML one and provides some consideration about their integration with the UML.

Section 5 provides some consideration about tool integration in the current definition of the SCTF.

Section 6 concludes the deliverable.

\(^1\) In particular the IBM Rational Rhapsody tool has been indicated as the UML tool which will be used in SafeCer in the healthcare demonstrator. However the guideline provided in this document can be applied by using any editor compliant with the UML language specification.
2 Input from previous pSafecer and nSafeCer work

The study addressed in this deliverable has been performed mainly on the basis of the current versions of the following SafeCer deliverables:

- D2.1.1 [4] describing the Generic Process Model (GPM);
- D2.2.4 [18] describing the SafeCer component model;
- D3.1.4 [5] describing the current version of the Certification Tool Framework (CTF);
3 Using UML to support the SafeCer process model activities

This section provides guidelines for the application of model based engineering practices as possible support for the instantiation of the activities envisaged by the SafeCer generic process model. Figure 1 below (taken from [4]) shows the activity pattern, in particular the green activities denote steps that are included in any development process while the blue activities denote steps that are specifically added to describe SafeCer specific activities.

Figure 1- SafeCer activity pattern

Here the UML standard [11] is taken as the reference modelling language given its recognized suitability in the different domains; moreover, regarding SafeCer, the UML has been chosen as the modelling language in the context of the health-care demonstrator.

It is worth noting that the activity pattern showed in Figure 1 covers generic activities. More detailed activities have to be derived once specific system development is addressed according to company specific processes, standards to be adopted etc. At the time of writing the definition of refined activity patterns for (some of) the demonstrators is an on-going task in SafeCer; for this reason the guidelines addressed in this deliverable are proposed in a quite general level, also referencing current state-of-the-art practices, and more specialized version should be derived according to specific activity pattern definition.

The UML and related profiles² to support the different activities defined in the SafeCer generic process are discussed in the following subsections.

3.1 About model based design with UML

UML model based design allows exploring potential system\software design solutions while increasing the formalism of the software model, so making the analysis process easier. Furthermore, it is the heart of the Model-Driven Architecture (MDA) initiative³ by OMG⁴; in the quest for increased quality and productivity, MDA promotes:

1) the use of models at various levels of abstraction as a vehicle for system specification, in the place of source code artefacts and informal diagrams that do not qualify as models;

² A profile is an extension mechanism defined by OMG which can be used to adapt the UML to a particular domain. The process of adaptation relies on the definition of stereotypes and constraints. A stereotype allows to extend the semantic of a particular entity defined in the UML whether constraints can be specified to put additional restriction on the meta-model entities.

Just like a Class, a Stereotype may have Properties, which have traditionally been referred to as Tag Definitions. When a Stereotype is applied to a model element, the values of the Properties have traditionally been referred to as tagged values. Stereotype specializes Class and its Properties have the same meaning in Stereotypes as they do in Class.

When a Stereotype is applied to a model element, the name of the Stereotype is shown within a pair of guillemets above or before the name of the model element.

³ http://www.omg.org/mda/

⁴ The Object Management Group (OMG®) is an international, open membership, not-for-profit computer industry standards consortium. OMG's modeling standards include the Unified Modeling Language (UML)
2) the use of automated transformations to progressively turn the user model into a software product ready for final compilation, binding and deployment.

The goal of MDA is to separate the design from the particular run-time architectures and by allowing the binding between them by using dedicated model transformations, e.g. in order to enable the reuse of systems/components across different domains and architectures.

UML is a domain independent language, and it comes with a set of standard profiles fulfilling domain specific needs, like SysML[14] and MARTE [13].

SysML is a standardized language for systems engineering. It is defined as a subset of UML entities and diagrams, the ones more related to the system design, moreover it provides specialized new constructs and diagrams for requirements engineering, traceability, and precise modelling of diverse physical phenomena.

SysML is used to deal with system development while UML is used to deal with software development; when used together on the same model they can help the system and software engineering to communicate and collaborate more easily.

MARTE is a dedicated profile used to enable the expression of real time properties for components and real time platforms.

Combining SysML and MARTE is another alternative to bring together SysML’s systems engineering constructs and MARTE’s ability in specifying non-functional aspects, e.g. to enable quantitative analysis starting from a UML model.

### 3.1.1 UML, a semi-formal language

The UML specification [11] comes with a set of so called semantic variation points, i.e. missing information about the semantic of certain constructs and their usage. The variation points have not been fixed by OMG in the UML specification in order to avoid the inapplicability of the UML itself in certain application scenarios and domains.

One significant example of variation point which can be ‘easily’ encountered while addressing the decomposition of a given component with UML concerns the way component ports can be connected in a parent child relationships; in particular, if one port of the parent is delegated to two (or more) ports of the inner parts (Figure 2) then it is not defined by the UML whether a request coming through the parent’s port will be forwarded on all internal links, or on only one of those links.
While modelling safety-critical systems all semantic ambiguities that can be found in the UML definition have to be fixed in order to be able to effectively use the model itself. This is especially true in a model-driven approach, where automatic model transformation need to be applied for analysis and code generation, and so artefacts production, and the semantic of the model to be transformed must be unambiguous.

Possible solutions to fix UML missing semantic rely on the definition of profiles; in fact profiles can be defined to enrich the modelling element available in UML but also to provide additional semantic for the already available UML language constructs.

OMG itself has defined a subset of UML, called Foundational UML (fUML) [9], so giving a precise semantics to an executable subset of UML (e.g. component ports are not available in fUML), without being tailored to any executable modelling methodology.

3.2 Requirement definition

By adopting a model driven design approach, requirements have to be captured/represented in the model and traced along the modeling process to the proper satisfying entities (e.g. component, contract, interface and argumentation) to be able to provide justification about the design choices.

By using UML a preliminary activity that helps the requirements elicitation phase is the definition of the use cases. Use cases can be used to identify scenarios which provide documentation about how the system/component is intended to be used; they are particularly useful to start defining requirements upon the interfaces and the protocols that will be used between the system/components and the given environment.

Use case based approaches for software requirement analysis are used extensively in software development industry to capture mainly functional and behavioural requirements. But use-case based techniques for requirement analysis has not been found to be very effective nor supportive enough for capturing non-functional requirements such as safety requirements. In [2] the authors discuss the principles and problems of hazard analysis and propose an approach to conducting hazard analysis on use case requirements representations. In [3] the authors propose a systematic approach for eliciting additional and/or missing safety requirements from textual description of use cases by the manual application of a well-known software safety analysis technique named Software Fault Tree Analysis (SFTA).
Use cases are generally written as plain text descriptions, so they are particularly suited to be part of the documentation associated to the requirement definition phase. Once the use cases have been identified, then requirements have to be derived from the use case description and linked to them for traceability. Of course, there can be are a large number of requirements bound to each of the use cases.

SysML allows managing requirements in the model. However the management of the requirements themselves is typically performed by using advanced and specialized requirements management tools, such as IBM Rational DOORS [15], while at model level they are typically introduced to manage traceability to the design entities, like components, contracts, test activity. E.g. the majority of the UML commercial tools (e.g. IBM Rational Rhapsody, Enterprise Architect, Artisan Real Time) allow to import the DOORS requirement into the UML/SysML model and manage the synchronization between the two environments.

The requirement entity available in SysML is also quite generic; in particular it comes with very basic predefined properties like identifier and description. Other more specific fields have to be provided through domain\company/project specific SysML requirement stereotype; for instance, to better support the SafeCer process, for a requirement it has to be possible to specify (and later query) if a given requirement is actually related to safety, i.e. a requirement which defines conditions that the system or the component must meet in the presence of failure conditions, or a safety-related requirement, i.e. a requirement (typically derived from a refinement of a safety requirement) which could have an impact on safety. The <<safety requirement>> stereotype should also have a property to specify the corresponding requested level of integrity for the expressed guarantees.

### 3.3 Contracts definition

One goal of the contract definition activity is to improve the communication between safety engineering and software/system engineering by using modelling constructs that allows software/system engineers to model safety related properties associated to components.

Some safety properties relevant to SafeCer [16] can be expressed through UML and its standard profiles, e.g. by using the Object Constraint Language (OCL) [12] for pre-post condition on the functional specification, or MARTE stereotypes for real-time properties such as WCET, memory size etc. On the other hand the concept of contract is not currently supported by the UML language and its standard profile and the right support has to be provided via a dedicated profile for contract definition.

To properly support the SafeCer process the profile for contracts has to be compliant with the contract modelling support specified in the SafeCer component meta-model. At the time of writing the SafeCer component meta-model is not in a final definition, a first release of the deliverable D132.1 "generic component meta-model" is going to be released, so is not possible to propose a final version of the UML profile for contracts here. Just as example we propose here a UML\SysML profile for contracts definition with the following stereotypes:

- **<<Formal Property>>**: a stereotype which extends UML Constraint. The specification of the <<FormalProperty>> is a string which holds the formal property expressed using a given grammar (e.g. Othello [7] or OCL).
- **<<Contract>>**: a stereotype which extends SysML ConstraintBlock. It has two attributes “Assume” and “Guarantee” both of type FormalProperty.
<<ContractProperty>>: a stereotype of SysML ConstraintProperty. It allows to instantiate Contracts in system blocks\textsuperscript{5} or in software components. It has a “RefinedBy” attribute in order to define the set of ContractProperties (i.e. the contract instantiated in the children blocks) that refine it. This is particular useful in order to be able to support the contract-based refinement approach proposed in SafeCer by FBK and to formally verify the refinement with the OCRA formal verification tool \cite{7}.

Next figure shows how the contract and its properties can be modeled in UML with the proposed profile.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.png}
\caption{Contract modelling}
\end{figure}

Here we refer to SafeCer D2.2.4 \cite{17} and D2.3.2 \cite{18} deliverables for what concern:
\begin{itemize}
\item how contracts should be defined in order to be useful for safety argumentation,
\item which languages for expressing contracts assumptions and guarantees can be used.
\end{itemize}

Note that the proposed Contract and ContractProperty stereotypes are particular extensions of the ConstraintBlock and ConstraintProperty SysML ones, where the latter support the definition of constraints upon the properties of a given Block.

FormalProperty is defined as a kind of Constraint which imposes some restriction upon the possible value of the properties of the component or block (e.g. its ports) on which the Contract that owns the FormalProperty itself is instantiated.

With the proposed profile the association between a Contract and a component is obtained by instantiating a ContractProperty in the component and then typing the ContractProperty with the given Contract; next figure shows Component1 having a <<ContractProperty>> typed with Contract1, the latter defined in Figure 3, which formalize the association between Component1 and Contract1.

---

\textsuperscript{5} In SysML blocks are modular units of system description. Blocks provide a general-purpose capability to model systems as trees of modular components.
The usage of the Contract and ContractProperty pair has some advantages; in particular they allow reusing the same contract for different components, where reuse of contracts can be thought as the analogous of reuse of requirements. In fact ContractProperty allows differentiating the refinement of the contract taking into account the particular component where the ContractProperty itself has been defined. So it can be that multiple ContractProperty, having the same Contract as type, have different refinement definition based on the different internal structure of the components where the ContractProperty are instantiated. For instance Figure 5 shows blocks Block1 and Block2 where the same contract Contract1 has been instantiated. Block1 and Block2 come with different decompositions, so they have different sub-blocks. By using the ContractProperty, i.e. its “RefinedBy” attribute, it is possible to specify that contractProp1 of Block1 and contractProp1 of Block2 are refined in a different way according to the different ContractProperty defined in the owned blocks.

Moreover by having Assumption and Guarantee modelled as distinct elements makes it possible to trace requirements to them. In fact it can be the case that an assumption and/or guarantee is introduced in the model according to the current derived safety requirements; in this case a SysML trace dependency has to be created from the requirement to the corresponding FormalProperty (Figure 6).
Figure 6: Tracing FormalProperty to Requirement

3.4 Design

By using the UML and SysML languages the entire design steps of the system down to the software can be addressed at different abstraction levels. In fact SysML/UML offers a good support to specify models and their refinement, by switching between specifications and prototypically implementing components or aspects for explorative purposes, and thus gaining a much clearer picture of characteristics of the remaining design space than with other development approaches.

The Requirements and Contract Definition activities discussed in the previous sections are usually performed with some interaction with the Design activity; this is due to the relationships and dependencies between the interested entities, like requirements vs contracts vs components. E.g. the contract assumes and guarantees properties are typically expressed as predicates over the component ports.

The guidelines presented in this section concerning the design are derived from the Functional Requirements and Verification Techniques for the Software Reference Architecture (FoReVer) project [10], an ESA study. FoReVer focuses on defining an integrated methodology and supporting toolset relying on a model-based approach and on introducing the formal verification of system properties from the early stages of the development process, allowing checking for the correctness of model refinements.

Basically FoReVer proposes the adoption of different system and software views corresponding to different abstraction levels: the Functional, Logical and Physical Architecture views, where the latter comprises the Hardware and Software views.

The FoReVer approach is based on a hierarchical decomposition approach: during the whole modeling process, from the early phases - at a higher level of abstraction - down to the later phases - at a lower level of abstraction -, the system is modelled in terms of architectural components described formally with their well-defined interfaces (i.e. ports according to the SafeCer component meta-model definition) and related properties. During each phase the components are considered as black boxes until they are refined into new lower level components in the next phase.

The FoReVer approach follows a contract-based approach: component’s properties are formalized in terms of contracts and thus composed of an assumption and a guarantee modeled as formal...
properties. In particular, from the initial modeling phase, system requirements must be expressed as formal contracts associated to the system and to the external interfaces representing interactions with the external environment.

The whole process runs through different conceptual models:

- from the higher level Functional architecture,
- to the Logical architecture,
- down to the lower level Physical architecture with specification of Software and its allocation on Hardware.

At each conceptual level, the model describes a vertical refinement with the decomposition of components into subcomponents and the refinement of contracts into a collection of contracts over subcomponents. Such vertical refinement is subject to formal verification and is the key-point in the overall verification process.

When stepping from one conceptual level to the next (e.g. from the Functional architecture to the Logical architecture), on the other hand, a new model is created and links are created to maintain the connection between corresponding entities in the two different architectures with different decomposition structure and contracts, mainly for the sake of traceability.

The process envisaged in the FoReVer methodology is an iterative top-down process where requirements of the higher level guide the decomposition and implementation choices at the lower levels. Due to the complexity of the systems, however, a simple waterfall process is usually not possible: in general it is not possible to define a “perfect” functional decomposition right from the earliest stages of the design, nor to identify feasible constraints and define a feasible allocation of functions to design entities from the earliest stages of the design. Constraints that stem from later decomposition and implementation choices may sometimes require changes to higher level assumptions or guarantees. Generally, before looking at changes of higher levels, the lower level decisions that would have caused the need for these changes must be re-thought and a different decomposition or allocation must be pursued. Changes that cannot be solved by rearrangements at the lower levels and require to be propagated upwards must be negotiated for compatibility with the higher level requirements. If such changes are accepted, the refinement verification must be executed again from the stage where the contracts are changed onwards.

The FoReVer methodology was originally defined for the space domain, however its applicability to other domains seems quite reasonable. One possible (not serious) implication could be that the number and the definition of the views here proposed could be adapted in order to support views commonly used in other domains and processes.

In the following sub-section the aforementioned functional, logical and physical views are further explained.

### 3.4.1 Functional Architecture

In the early steps of the design the system capabilities and their possible implementation are investigated through functional analysis. The goal of the functional analysis is to decompose the system functionalities in order to cope with the system’s complexity and define what is called functional architecture. System functionalities are represented in the function architecture as black-box functions, where data flow between the functions is introduced.

Functional architecture is modeled in SysML as a set of hierarchies of functional entities represented as Activity Blocks, using SysML Block Definition Diagrams (BDD) to depict the composition relationships and using Internal Block Diagrams (IBD) to depict connections between functional entities. The root of each hierarchy is a top level function to be performed by the system.
The Rationale construct available in SysML (Figure 7) can also help in this phase (and also in the other design phases) to store the information that a specific design exists to support another design element, or to fulfil specific requirements or contracts. It explicitly allows modellers to trace the design to specific requirements and it allows the user, for example, to specify a rationale that may reference more detailed documentation such as a trade study, analysis report or documents coming from domain safety standards.

![Figure 7: Using the SysML Rationale (from SysML specification)](image)

### 3.4.2 Logical Architecture

In case of complex systems it is often necessary to construct separate depictions of the system themselves and define mappings between them. For example, a complete system hierarchy may be built and maintained at an abstract level. In turn, it must then be mapped to another complete assembly hierarchy at a more concrete level. The set of models supporting complex systems development may include many of these levels of abstraction. The models concerning intermediate level of abstractions are part of the so called logical architecture.

Functions identified in the functional architecture are allocated to the entities of the system logical architecture.

The allocation activity reasonably involves only the leaf functions of the functional architecture: the finer-grained functions available in the function tree provide the baseline for functional allocation to the system logical architecture hardware and software elements.

The optimum partition has to be identified using trade-off studies: further functional refinement could be required for functional partition and allocation: functions are split or new functions are introduced to take into account the optimal allocation to the hardware composition as envisaged at this level; however the system logical architecture is still independent from specific physical artefacts and implementation strategies.

The allocation target entity of the system logical architecture must define (at least) a set of ports which is compatible with the data flows defined for the allocated functional entity.

### 3.4.3 Physical Architecture

The system physical architecture defines in detail all the concrete subsystems/components, in particular the software and hardware ones. The physical architecture is actually composed of two
distinct sub-architectures, the Hardware Architecture and the Software Architecture which allow introducing the actual HW and SW entities and their deployment, i.e. how SW component are allocated to the hardware resources.

System blocks to be implemented via software can be identified in the SysML diagrams and then mapped and refined as UML component.

Specific support for System and software co-engineering enables the transition from the architectural and functional information available in the SysML model to the initial software and hardware (i.e. hardware related to the software deployment) architecture as a seamless process.

### 3.4.4 Tracing between the different design levels

Allocation of Functions from the Functional to the Logical Architecture and from the Logical to the Physical Architecture is supported by the SysML allocation relationship, allowing full traceability of design decisions across different design phases and thus highlighting the impact of design changes in different phases.

![Figure 8: tracing between different levels](image)

When allocation one entity from a view to another contracts (and so requirements) specified for the source entities are allocated on the target entities as well, in a certain sense the target entity inherited the contracts specified for the source one; then the inherited contracts can be further refined in the current view where the target entity is defined.

Within each architectural level models are progressively refined, whereas passing from one architectural level to the next requires a change in perspective and often implies also a change in involved responsibility and professional profiles. Mapping leaf elements from the higher level architecture onto elements in the lower level architecture is a valid support to ensure coherence and continuity in the development process and eases the communication between professionals in charge of the different phases in the lifecycle.

### 3.4.5 From software model to code

In general all tools which are employed within the development process of safety-critical software have to be qualified in order to be able to reuse the produced artefacts for certification/qualification purposes. Tools that facilitate design automation, as in particular model driven approaches for
code generation are requested to be qualified with the same rigour as the safety-critical software itself. Whereas tools that are intended for use in the safety assurance process, i.e. that support testing, validation or verification, can be qualified by a more light-weight assessment process (e.g. see [29]).

If code is generated from the models, the relation between model and code has to be clarified. If the semantics of the model itself is given by such a translation, all arguments relying on the model must be traced in the code it represents, which may be difficult to achieve. Otherwise, formal relations between the model and the code semantics must be established, once for all. One facet of this problem can be addressed by certifying the code generator as it has been done for SCADE (see section 4.1). If the generated code is modified later, be it for reasons of efficiency or platform compatibility, this must be reflected in the model (e.g. via roundtrip engineering) or addressed in the respective artefacts, e.g. the document concerning the software specification and its verification.

3.5 Verification

UML and associated standard profiles do not offer verification capabilities per se, but rely on different languages and tools for them. UML models can be taken as input and transformed via model to model or model to text transformations to feed the desired verification tools. To be able to define these transformations the semantic of the UML model in input has to be fixed to allow statically mapping each UML model element of interest into one or more entities of the target language.

In the context of the SafeCer project some tools are proposed to allow verification starting from a UML model, in particular:

- **MoMuT::UML** (proposed by AIT), see section 3.5.1.
- **OCRA** (proposed by FBK) allows formal verification of contract refinement. Example of how a UML model can be transformed to a corresponding OCRA one is available in SafeCer within the CHESS toolset (proposed by Intecs).
- **CHESS** allows schedulability analysis transforming models designed with a sub-set of MARTE language into the MAST model

6 MAST ([http://mast.unican.es/](http://mast.unican.es/)) defines a model to describe the timing behaviour of real-time systems designed to be analyzable via schedulability analysis techniques. MAST also provides an open-source set of tools to perform schedulability analysis or other timing analysis.

© The SafeCer Consortium
Using these mutants, abstract tests in the form of input-output-graphs are produced. They are built such that they provoke the mutant to behave different than the original model.

The current version of MoMuT::UML depends on a proprietary UML profile to mark the test interface of the system. This will be changed to just select the group of components to test (1 for unit testing, 2 or more for integration testing, all for system/acceptance testing). The interface will be subsequently derived from the composition of the component interfaces.

![Diagram](image)

**Figure 9**: MoMuT::UML

### 3.6 Certificate Preparation and Argumentation

The MDA approach, especially compared with paper-based documents approach, can be of help for and improve the certificate preparation and argumentation phases, in particular to ease the navigation of the information and to allow for diversity of presentation, delivery and re-use. An example comes from [19] where the authors propose a UML profile and a general approach that uses MDA techniques to aid preparation for certification with the adaptation of general modelling tools used in system development to manage evidence for certification.

During the different model design activities, safety related requirements and properties have to be distinguished and traced throughout the development phases. They are the major subject of the recommended verification and validation techniques and so of the certification preparation phase.

In general, if safety related properties are properly represented in UML models, in particular by using a dedicated profile (e.g. as the one proposed in section 3.3 concerning contracts modelling), then a tool can automatically generate reports containing safety and certification-related
information about the system. Those reports could also be used as evidences of system compliance with the identified requirements and standards (as proposed in [19]), and then presented to the external certification authority.

Employing UML models in the safety design and assurance process requires conclusive case arguments on several aspects:

1) the use of UML models in activities and through the development process has to be clarified as for any other artefact. It has to be demonstrated how and to which confidence level the requested safety objectives and quality characteristics can be achieved by UML-based techniques.

2) Safety standards recommend a rich portfolio of safety strategies ranging from defensive programming, design diversity or restriction of programming languages to elements that are statically verifiable to formal V&V techniques and testing. Those strategies are widely accepted in safety engineering and can be supported by a modelling approach.

Therefore, not only the SysML\UML model must contain different views on the architecture and the design; the requirements linked to the design and safety considerations leading to that design have to be represented in the model too. Then the designer may traverse the model guided by the stereotypes provided by the proper profile to comprehend the safety argumentation.

A model based approach can also be used to support the specification of safety cases, and so claims, arguments etc, as proposed by the Goal Structuring Notation (GSN) [27].

One further possibility to support the modelling of safety case comes also from OMG and its Structured Assurance Case Metamodel (SACM) [23]. SACM is a metamodel (see Figure 10) for representing structured assurance cases, i.e. a set of auditable claims, arguments, and evidence created to support the claim that a defined system/service will satisfy the particular requirements. A mapping between the SACM metamodel and the elements defined in the GSN is also provided in [23].
SACM metamodel can be implemented as UML profile, and so used to extend the modelling capabilities of UML to the safety case concerns. This integration can be particularly useful to see how the system lifecycle products (in particular system requirements and design) relate to and satisfy the assurance requirements. Figure 11 shows the current definition of the SafeCer Component model related to contract, properties and argumentation. In particular by using SACM and UML it could be possible to model the relationships between the components, component’s contracts and the arguments.

---

7 Please note that the definition of this part of the SafeCer component meta-model is currently ongoing. The final version could be different.
In general, in order to be able to generate certification information from UML models, software development tools must support the ability to search UML models based on the stereotypes that are applied to model elements and the values of the stereotypes attributes.

One possible way to extract safety information from UML models is to provide a mechanism to do so in the UML modelling tool itself. All the commercial UML modelling tools already offer some sort of search capabilities for the designers. This UML model-driven development tool allows designers to search UML models for uses of specific stereotypes. E.g., supposing to have the <<SafetyRequirement>> modelling safety-related requirements, to retrieve in the model the list of all design decisions that are a result of safety-related requirements, the following OCL-like query can be used:

\[
\text{Model.AllOwnedElement.iterate()} \rightarrow \text{select}( r : <<Rationale>> | r.\text{AnnotatedElement.IskindOf}(<<SafetyRequirement>>))
\]

The aforementioned query will return all the Rationale elements which have been attached to SafetyRequirement.

The Eclipse Modeling Framework (EMF) for Java [20] is a popular, and easily extensible, software development framework. Some of EMF’s extensions include the capability to use the Object Constraint Language (OCL) [12] to specify search queries on UML models, and then write Java code to execute them.

XMI [21] is an OMG standard for representing, and therefore exchanging, models and metadata in an XML-based language. In practice, UML modelling tools can export UML models in the XMI language. This would create an XML file containing all the model data. Since XMI is a standard format, it can be imported by any other tool, thus establishing a common format across different tools.
4 Complementary Specific Modeling Techniques and Tools

While developing complex system according to a model driven approach it can be the case that more than a single language, methodology and tool have to be used to support specification, design, verification and operation of complex safety properties. In particular different languages, methodologies and tools can usually be used to support the definition and analysis of system requirements, early system design validation, software verification, development of test procedures, support for units and subsystem tests, etc.

The use of SysML\UML as a modeling language shall allow to gather all the information on requirements, which is distributed in all the different formalisms and models and to offer at the highest level a synthesis of the requirements and system\component properties facilitating their evaluation of compliance.

A possible solution to better integrate SysML and UML with other complementary modeling language can be obtained by using the profile mechanism and so extending, as far as possible, the SysML language.

From the book “System Engineering with SysML UML” [22]:

“SysML is used to describe a system across disciplines. This is why the model’s degree of details should stop when a mixture of disciplines is no longer given.

If you have elements in your SysML model that can be fully allocated to one single discipline, then it is meaningful to mark them with a stereotype.

Discipline-specific elements are blocks or connectors, and they may be other elements that can be fully allocated to one single discipline. They generally form the bottom limit of the SysML model. A discipline-specific block is detailed in a model and in a language pertaining to that discipline.”

Moreover the advantage of this solution is that the language/methodology of the other specific system engineering disciplines can take advantage of the SysML\UML capabilities and so manifest possible improvements.

It is worth noting that one of the challenges in this approach is to ensure the consistency of the different adopted languages and so models, and to ensure that the overall verification logic allows confirming the fulfilment of the requirements at the end of the process. These challenges are currently subject of research; for instance the Project P\(^8\) aims to verify the semantic consistency of systems described using safe subsets of heterogeneous modelling languages, ranging from behavioural to architectural languages and presenting a synchronous (e.g. Matlab/Simulink) and asynchronous semantics (e.g. SysML, MARTE, UML).

The purposes of this section is to address two commonly languages and corresponding tools widely used for the development of safety system across the different domains, i.e. SCADE and Matlab/Simulink, while discussing impacts or solutions for their integration with SysML/UML.

4.1 SCADE

4.1.1 SCADE Overview

\(^8\) Project P is a three-years research projects funded within the French FUI 2011 funding framework. (http://www.open-do.org/projects/p/)
SCADE (the Safety Critical Application Development Environment), from Esterel\(^9\), is both a notation and a toolset that was specifically developed to describe and implement safety critical software for application domains such as aeronautics, automobile or medical; its semantic founds its basis upon the synchronous language LUSTRE [24].

SCADE is particularly adapted to computation-intensive software such as control loops, based on a synchronous computational model.

The SCADE notation includes both block diagrams and safe state machines, giving a rigorous description of the complete behaviour of the software. Its main characteristics are:

- Strong typing
- Explicit initialization of data flows
- Explicit management of time (delays, clocks, etc)
- Simple expression of concurrency (data dependencies)
- Deterministic execution

The SCADE toolset includes: the editor, the simulator, the generator of certifiable/qualifiable code\(^10\) and support for formal verification.

### 4.1.2 SCADE gateways

Several gateways from SCADE to other languages and formalisms are available.

For instance, regarding requirement management, SCADE is integrated with the IBM tool DOORS to add requirement management to SCADE development environment.

Another gateway implemented by SCADE is the Simulink Gateway. The Simulink gateway supplied with SCADE Suite is capable of generating a SCADE discrete controller from a Simulink model. This gives all the benefits of the SCADE formalism and it can therefore be formally verified and certifiable code can be generated using the SCADE code generator. The tool also allows integrating a SCADE model into the Simulink model by wrapping the generated code. It is then possible to co-simulate the two models using the SCADE simulator with data flowing in both directions.

SCADE Simulink gateway can also be useful to re-use Simulink models designed during system prototyping and for system requirement analysis. In fact, once the system requirements are mapped to software requirements, Simulink models used to derive these requirements could be imported in SCADE to start software analysis and design; however, given that not all Simulink blocks are supported in SCADE, this is not always possible.

A mapping from SysML blocks to SCADE has been defined in [25]; in particular the mapping regards only the static information available in a class/block diagram, so it considers components, interfaces, ports, parts, connectors, operations and attributes. UML behavioural information is not taken into account by the mapping.


\(^10\) SCADE Code Generator is qualifiable as a development tool under DO-178B level A requirements, IEC 61508 safety objectives at SIL 3, and EN 50128 safety objectives at SIL 3/4
Starting from this mapping Esterel has developed a gateway from SysML/UML models, in particular the tool allows to import class or blocks defined in a SysML model made with Rhapsody into a SCADE one. It is worth to note that in this approach, it is basically assumed that a mapping of SysML model entities to SCADE one is possible and automated, meaning that the SysML model decomposition is already defined down to the software design level.

This way the user can retrieve in the SCADE model the entities defined initially in the SysML model and start to model their behaviour, generate code and simulate. While this is fine for purely safety-critical application, in case the application is hybrid SCADE model and SysML model coexist at the same level: in this case the gateway tool from Esterel allows to generate a dedicated wrapping code that will serve to merge the code generated from SCADE into the code generated through the SysML tool (usually C or C++), allowing for co-execution/co-simulation of the whole system.

Also incremental import/re-import of SysML/UML models into SCADE and navigation from translated SCADE models back to the SysML/UML elements is supported.

### 4.1.3 SCADE and UML/SysML

While SCADE suite fits well for safety critical software design, UML/SysML are general purpose language and offer to system and software engineering a better support for architectural design, the latter particular useful to the system engineering in the early phases of the system design.

Starting from requirements allocated to software, in case of hybrid synchronous software applications SCADE and UML can be used together by using the aforementioned gateway.

Next figure summarizes the support of SCADE in relation to the SafeCer activity pattern; it is worth saying that SCADE does not support the concept of contract while it supports specification of safety properties.

![Figure 12: SCADE support to the SafeCer activity pattern](image)

### 4.2 Matlab/Simulink

#### 4.2.1 Matlab/Simulink Overview

The MATLAB product family, from MathWorks, provides a high-level programming language, an interactive technical computing environment, and functions for algorithm development, data analysis/visualization and numeric computation.

Simulink is an environment for multi-domain simulation and model-based design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test time-varying systems, including communications, controls, signal processing, video processing, and image processing.

---

11 [www.mathworks.com/](http://www.mathworks.com/)
Simulink allows hierarchical, usually top-down and design through the subsystem block. In addition Simulink can be integrated with Stateflow, a product to model design and simulate state machines and control logic.

Other add-on products extend Simulink software to multiple modeling domains, as well as provide tools for design, implementation, and verification and validation tasks.

Simulink is integrated with MATLAB, providing access to an extensive range of tools that allow develop algorithms, analyze and visualize simulations, create batch processing scripts, customize the modelling environment, and define signal, parameter, and test data.

Simulink Coder add-on allows generating code from a Simulink model for different platforms. However, given the semi-formal nature of Simulink, it can result poor for safety control system domain: in this case translation into another formal language (e.g. SCADE) and then code generation can be preferred.

Another plug-in is the Verification and Validation library which allows to:

- establish requirements for a Simulink model by linking them with model elements that satisfy them,
- verify proper function of the model by monitoring model signals during extensive testing,
- validate the model, making sure that all possible model decisions are taken through testing,
- customize the Model Advisor to analyze a model for settings that result in inaccuracies or inefficiencies.

The creation of a requirement and its association to a Simulink element can be done in Simulink through a dedicated editor. It is also possible to link a requirement with an actual requirements stored in a textual external document

The requirements library defines a special block called System Requirements which allows to list all the requirements associated with the model or subsystem in the current Simulink diagram.

Once you place the System Requirements block in a Simulink diagram, it automatically lists the requirements associated with the model or subsystem depicted in the current diagram. It does not list requirements associated with individual blocks in the diagram.

A system requirements block is illustrated in Figure 13; each of the listed requirements in a SystemRequirement block can be an active link to the associated external requirement.
It is allowed to place a SystemRequirement block anywhere in a Simulink diagram, but not to connect it to other Simulink blocks, for instance to represent trace or satisfy relations.

It is not possible to have more than one System Requirements block in a diagram; this can result in a possible flattening of the requirement hierarchy.

Simulink Verification and Validation also provides the Requirements Management Interface for IBM DOORS software to associate DOORS requirements with Simulink model objects. Figure 14 (taken from the Simulink documentation) shows the Simulink and DOORS relations.
4.2.2 Simulink and SysML/UML

Although there is some overlapping, e.g. with regard to state-machines modelling, the two environments can complement each other. SysML is more powerful with regard to requirements modelling and the overall system design, while MATLAB/Simulink has its strengths in the simulation and control algorithmic area. If both environments are used a chain of tools is needed to ensure consistency between the models. Figure 15 summarizes the SysML/UML and Simulink comparison.
- Advantages from object orientation (reuse, modularity, etc)
- Better for system specification (functionality, QoS requirements)
- Requires much user-provided source code/third part code generators
- Better for expressing time-continuous and time-discrete problems
- Complete code can be automatically generated from a Simulink model (e.g. using Simulink Coder)
- Lack of OO concepts decrease maintainability

Figure 15: SysML/UML vs Matlab/Simulink

Two approaches have initially been used to merge the UML and the Matlab/Simulink environments: co-simulation and a common underlining executable language; they are discussed in the following, together with a new approach more MDA oriented.

4.2.2.1 Co-Simulation

Co-simulation considers UML as an executable language and, starting from an existing tool capable of executing UML models, tries to integrate it with the Matlab/Simulink simulation environment to produce a combined simulation. This solution has the advantage to allow engineers to continue working with their well-known mono-disciplinary tools, without having to redo their modelling work in some multi-disciplinary modelling environment; however it is far from trivial, especially if its implementation doesn’t want to rely on a specific UML tool only.

One solution comes with Artisan Studio\textsuperscript{12} which allows SysML/UML and Simulink simulating together. During the simulation, all connected models on all participating computers are executed as parallel, separate tasks (co-simulation) in their native simulation tools.

4.2.2.2 Common executable language

A second feasible approach founds its basis in a common executable language, for instance C/C++. Following this approach executable code is generated from the UML and Simulink model and then coupled to obtain a single executable model: the main advantage of this solution can be found in a faster simulation speed.

Clearly this solution is even more powerful if the integration between the two environments happens to an upper level, i.e. at model level, instead of code level, accordingly to the MDA principle. For instance this solution has been recently implemented by IBM tool Rhapsody; in fact this tool allows the user to include control algorithm and plant models developed in Simulink into a Rhapsody design, thereby enabling a hybrid modelling, execution and code generation environment.

\textsuperscript{12} www.atego.com
In Rhapsody the Simulink blocks can be imported in the current SysML/UML model to be represented as a stereotyped SysML block with flow ports: each of these blocks, once imported, can be linked to the other blocks of the design. Simulink functional blocks can be imported in the UML model together with the associated code generated through Simulink Coder: this is mandatory for the execution phase.

Rhapsody offers another possibility to the users, i.e. to plug a block representing the Rhapsody model into the Simulink model. This allows to co-simulate software and systems designs captured in Rhapsody with plant, software and system models designed directly in Simulink.

![Common language from SysML and Simulink](image)

**Figure 16: Common language from SysML and Simulink**

### 4.2.2.3 Switching between SysML/UML and Simulink models

Another possible solution, that is an application of the MDA, is to generate a Simulink model starting from the (structural and behavioural) information modelled with SysML.

![From SysML to Simulink](image)

**Figure 17: From SysML to Simulink**

For instance [28] proposes an automatic mapping from UML to Simulink: the mapping is based on sequence and deployment diagram and it shows how some UML constructions can have a direct mapping to Simulink.
Also it seems quite feasible that the news coming with SysML would enhance new mappings toward Simulink. In particular the extensions provided in SysML for the UML activity diagram to express continuous behaviour and control operators seems to fit well for this scope; e.g. Figure 18, taken from the SysML specification, shows an activity diagram13 modelling a numerical solution (as possible with Simulink) for the differential equation \( x'(t) = -2x(t) + u(t) \).

![Activity diagram](image)

**Figure 18: Activity modeling numerical solution**

Model transformation of Simulink to UML/SysMML ones is also feasible. E.g. in [26] the authors proposed mapping rules to transform a Simulink model to a UML model; the mapping targets class diagrams and object diagrams, where each class or object corresponds to a subsystem block in a Simulink model that represents a part of control logic.

### 4.2.2.4 Considerations about the integration

As a conclusion, the first two approaches currently seem to be more suited to analysis, verification of low level design models, for instance the ones available during the software engineering process activities; this is due to the fact that these approaches rely on commercial UML tools support, in particular for what concern code generation and simulation, and currently these support is more focused on UML than SysML.

From our point of view, SysML and Matlab/Simulink do not fully apply on the same part of the development process, in particular we do not foresee the use of Simulink for requirements management.

We should highlight Simulink offer very early in the process some simulation/model animation capability in order to perform quick prototyping of the designed system, but we have to be very careful on this point: if you go deeper in the detail (by instantiating a Simulink model early in the process), you start the implementation and you have a loss of reusability of the element.

Our opinion is that it is important to use the tools for what they have been designed first. Matlab/Simulink offers more and more capabilities but initially it was mainly developed for technical engineering, modelling, simulation but not for requirements management (some kind of adding layer which brings confusion inside the model from our point of view and a lack of separation in the concerns).

For us, only the solution depicted in Figure 17 is the one which can bring more advantages, in particular by making possible to iterate between the two models\textregistered; environments, for instance by

---

13 The \(<\text{streaming}>\) and \(<\text{nonStreaming}>\) stereotype indicate which activities take inputs and produce outputs while they are still executing.
starting with the SysML/UML regarding the definition of requirements and then the software components and properties, then moving to Simulink to perform implementation design and validation, and then going back to the SysML/UML to possibly address some modification upon the architecture definition or about component properties, and so on, exploiting the code generation of the tools when the overall design results satisfactory.

Next figure summarizes the support of Matlab/Simulink in relation to the SafeCer activity pattern.

Figure 19: Matlab/Simulink support to the SafeCer activity pattern
5 Tool’s Integration Approaches

The SafeCer tool framework is based upon the CAR and CTF tools; existing tools related to the system/software development life-cycle are meant to be integrated in the tool framework via the CTF by using two separate dedicated plug-in, a launcher and an importer [5].

The importer is responsible for generating the descriptor(s) of the artefact(s) produced by the given tool; each descriptor comes with meta-data information about the kind of artefact that needs to be registered in the SCTF, its properties and the tool used to generate the artefact itself; eventually the descriptor can also store the dependencies with other artefacts which can be considered in input or output to the current one.

The launcher is responsible for running the given tool, possibly providing the needed artefacts that are required in input. This software can then be used to manually launch the tool through the CTF, or can be used by the CTF, guided via the CAR, to automatically launch the tool.

The second scenario is particularly interesting and useful when the tool to be launched can produce some artefact in an automatic way; examples of such kind of tools are model validators, test executors and code generators. For instance the CAR can detect that an artefact corresponding to some source code has been changed and then, according to the registered artefact dependencies, it can invoke the proper tool to run the tests, so to update the artefact corresponding to the test results. The same scenario does not fit well with UML editor tools given that this kind of software is meant to produce artefacts, i.e. models, via human interactions.

The following consideration should be taken into account while trying to define a SCTF instantiation for a given application scenario:

1. considering the list of artefacts that need to be produced, e.g. according to the activities defined in the particular adopted process model instantiation
2. selecting a set of tool to support the creation/management of the artefacts
3. integrating/adapting the set of tool in the SCTF

The last step can require the creation of the proper launcher and importer for new tool not yet integrated in the CTF; it can be evaluated if some parts of the identified tool chain can be activated automatically by developing tools integration, e.g. based upon model transformations. This can be particularly feasible if a model driven engineering process is followed which considers:

1. the use of models at various levels of abstraction as a vehicle for system specification, in the place of source code artefacts and informal diagrams that do not qualify as models;
2. the use of automated transformations to progressively turn the user model into a software product ready for final compilation, binding and deployment.

Another possibility regarding tool’s integration in SafeCer is offered by the WEFACT tool proposed by AIT. WEFACT is part of the SCTF by supporting the tool integration; furthermore it supports the instantiation of the process model for various domains including the healthcare domain. WEFACT will support traceability between the requirements and the associated elements of different UML diagrams. WEFACT supports the integration of tools with OSLC\textsuperscript{14} automation specification, for instance MoMuT (see also chapter 3.5.1 on page 19).

\textsuperscript{14} http://open-services.net/
6 Contribution to overall SafeCer objectives

The purpose of nSafeCer WP611 is to investigate cross-domain languages and tool integration with the SafeCer tool framework and their usage along the SafeCer certification process. In particular UML language and editor has been identified as one important tool to be integrated for the healthcare SafeCer demonstrator.

The purpose of this deliverable is to provide guidelines for the instantiation of the UML language to support the SafeCer generic process model activities and for the integration of UML editors in the SafeCer common tool framework. SCADE and Matlab/Simulink modelling tools\languages have been also considered, possibly as complementary to the UML.
7 References

[1] CHESS Artemis JU deliverable D2.3.2 “Multi-concern Component Methodology and Toolset” available from http://www.chess-project.org/page/results


[3] C Pankaj Vyas, RK Mittal, Eliciting additional safety requirements from use cases using SFTA, Recent Advances in Information Technology (RAIT), 2012 1st International Conference


[6] pSafeCer deliverable D3.1.5 “CAR Prototype”


[8] pSC D3.2.1/D3.2.2 “Design and implementation of the support to the design of components architectures with the specification of certification properties and safety contracts”- “Design and implementation of the support to the validation and verification of the components and their composition”, pSafeCer deliverables, v1.0, 14/01/2013


[15] IBM Rational DOORS

[16] pSC D2.2.3-D2.4.2 "Definition of the meta-models and languages for CP & SC. Verification and validation constructs in meta-model targeting the SafeCer methods for certifying components”, version 1.4


[18] pSC D2.2.4 Deliverable: “Generic component meta-model” version 1.4.


