Security and Safety Modelling

Artemis JU Grant Agreement no.: 295354

D4.1 – Integrated Design and Evaluation Methodology
Version 01
31 October 2013
Preliminary

Dissemination level: PU

Contributing partners
INT, ADEL, IFAG, SAG, IKV, CTU, CITY, GM

Project Partners:
Intecs, Institute of Informatics and Telematics - CNR, AKHELA, Università degli Studi di Roma La Sapienza, Technical University of Denmark, FTW Forschungszentrum Telekommunikation Wien, Adelard, UniControls, Czech Technical University in Prague, PSA Peugeot Citroën, SYSGO, ikv++ Technologies, eesy-id, Infineon Technologies AG Deutschland, EADS DEUTSCHLAND, Électricité de France, SYSGO s.r.o., Siemens AG Österreich, City University London - Centre for Software Reliability, General Motors Research & Development

Every effort has been made to ensure that all statements and information contained herein are accurate, however the Partners accept no liability for any error or omission in the same.
# TABLE OF CONTENTS

1 Introduction ............................................................................................................................. 6

2 Domain Specific Methodologies ............................................................................................. 7
  2.1 Automotive Domain .................................................................................................................. 7
    2.1.1 Lifecycle of the automotive safety standard ISO26262 ............................................................ 7
    2.1.2 Approach for a safety-security combined process definition ................................................... 8
  2.2 Industrial Control Domain ..................................................................................................... 14
    2.2.1 Industrial Domain Safety Lifecycle (IEC 61508) ...................................................................... 14
    2.2.2 Industrial Domain Safety Lifecycle Overview .......................................................................... 14
    2.2.3 Concept .................................................................................................................................. 15
    2.2.4 Overall Scope Definition ......................................................................................................... 15
    2.2.5 Hazard Analysis and Risk Assessment ....................................................................................... 16
    2.2.6 Overall Safety Requirements ..................................................................................................... 16
    2.2.7 Overall Safety Requirements Allocation ...................................................................................... 17
    2.2.8 Overall Safety Requirements Specification .................................................................................. 17
    2.2.9 Safety-Related Systems Realization .......................................................................................... 18

3 Relevant Background .................................................................................................................. 21
  3.1 Chess Methodology .................................................................................................................. 21
    3.1.1 SafeCer extensions .................................................................................................................... 22
  3.2 UML based Profiles (MARTE, SysML) ..................................................................................... 24
  3.3 EVITA ....................................................................................................................................... 25
    3.3.1 Security requirements analysis ................................................................................................... 25
    3.3.2 Secure on-board architecture design and prototype ..................................................................... 26
    3.3.3 A SysML-based environment for formal verification of Safety and Security Properties .............. 27
  3.4 Tolerability of risk and ALARP .................................................................................................. 30
  3.5 Safety cases ............................................................................................................................. 31

4 Concepts and terminology ........................................................................................................ 35
  4.1 Terminologies and taxonomies ................................................................................................. 35
    4.1.1 Security terminology ................................................................................................................. 36
    4.1.2 Safety terminology .................................................................................................................... 37
    4.1.3 Discussion ................................................................................................................................. 37
  4.2 Levels and classification ........................................................................................................... 39
    4.2.1 Safety integrity levels ............................................................................................................... 39
    4.2.2 Security levels .......................................................................................................................... 40
    4.2.3 Evaluation assurance levels ....................................................................................................... 40
    4.2.4 Discussion ................................................................................................................................. 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Security controls</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>SESAMO Generic Methodology</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>Principles</td>
<td>43</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Risk-based</td>
<td>43</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Model-based</td>
<td>43</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Separation of concerns</td>
<td>45</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Multi-view</td>
<td>45</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Component-based</td>
<td>46</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Incremental and iterative</td>
<td>49</td>
</tr>
<tr>
<td>5.1.7</td>
<td>Assurance and evidence</td>
<td>49</td>
</tr>
<tr>
<td>5.1.8</td>
<td>Decision support</td>
<td>50</td>
</tr>
<tr>
<td>5.2</td>
<td>Formalism</td>
<td>50</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Unified Modeling Language (UML)</td>
<td>50</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Business Process Modeling Notation (BPMN)</td>
<td>51</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Relevance for SESAMO</td>
<td>52</td>
</tr>
<tr>
<td>6</td>
<td>Towards a risk-based methodology</td>
<td>54</td>
</tr>
<tr>
<td>6.1</td>
<td>Generic process definition</td>
<td>54</td>
</tr>
<tr>
<td>6.2</td>
<td>Security-informed risk assessment</td>
<td>55</td>
</tr>
<tr>
<td>6.3</td>
<td>Security-informed safety case methodology</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>References</td>
<td>58</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1: Automotive Domain Safety Lifecycle ................................................................. 7
Figure 2: V-Model applied in ISO26262 ........................................................................... 8
Figure 3: Formalized V-Model for a safety-security combined process .................... 9
Figure 4: Safety and security activities of the Concept Phase ........................................ 10
Figure 5: Safety and security activities of the System Design Phase ......................... 12
Figure 6: Safety and security activities of the Hardware/Software Design Phase ........ 13
Figure 7: Industrial Domain Safety Lifecycle – Overview .............................................. 14
Figure 8: Industrial Domain Safety Lifecycle – Detailed ............................................. 20
Figure 10: The AVATAR Methodology Stages ............................................................... 27
Figure 11: AVATAR security requirements block diagram for a FLASH programming process ... 29
Figure 12: TEPE description of safety related properties of an elevator system .......... 30
Figure 13: The ALARP principle .................................................................................. 31
Figure 14: Toulmin’s formulation of a claim ................................................................ 33
Figure 15: Model of system failure behaviour .............................................................. 37
Figure 16 Component Assembly .................................................................................. 47
Figure 17 Component and Container .......................................................................... 48
Figure 18 Connector as explicit interaction constructs between components/containers .......... 48
Figure 19 – Integration between threat/risk assessment and security assurance .......... 49
Figure 20 Example for a business process modelled in BPMN ................................... 52
Figure 21 – Risk management process ...................................................................... 54
Figure 22 – Risk treatment ......................................................................................... 54
EXECUTIVE SUMMARY

In the definition of a generic integrated methodology and process for safety and security, two approaches are essentially possible. A *top-down* approach starts with a domain-independent perspective on a single process and set of activities which could be instantiated or tailored for specific domains. A *bottom-up* approach, to first have a look onto the processes and activities applied in the different domains, then to collect relevant background information, concepts and terminologies before starting to give an outlook towards a generic process, is considered to be the more practical and realizable approach in the context of SESAMO.

Domain specific process descriptions may be found in the domain standards and taken as points of departure for study and modification. In the automotive domain, the ISO26262 standard for safety has been augmented with security specific activities and work products to create a first integrated approach in that domain. In the industrial domain, the IEC 61508 standard provides a further example of a safety-oriented standard defining a lifecycle and activities amenable to augmentation with security related characteristics.

The approach in SESAMO is informed by the technological and methodological background brought into the project by the consortium. Model based development technology and competence are provided by CHESS in the form of cross-domain component modelling techniques, separation of concerns, and separation of platform-independent and dependent modelling. Competence in real-time (MARTE) and systems development ( SysML) profiles of the Unified Modelling Language is available from several consortium members. Background from the EVITA project has brought in several components of a joint security and safety approach that feed into the work.

Specialized competence in key areas of safety and security related standards is necessary to complete the context. In particular, modern approaches both to security and safety standards hinge upon basic principles such as the ALARP approach to risk management, and the use of safety cases (and their extension to security cases) within the framework of assessment and assurance activities.

For the definition of a generic safety-security integrated methodology and process a number of aspects are considered. A first, essential activity is a definition and harmonization of concepts and activities from the respective security and safety sectors, since different concepts are used in those domains, often with conflicting interpretations. A second activity consists in the affirmation of a set of principles to govern the definition of a generic methodology. These principles were laid out in the technical annex, to be further developed, including the general model based approach, the centrality of risk, separation of concerns, multiple views, incremental and iterative lifecycle, and the coupling of assurance with evidence, with an ultimate goal of providing decision support in whatever form is appropriate for the end user.

A formalism or restricted set of formalisms is necessary to define a process in a form amenable to tool support. UML and its various profiles is the focus, augmented by more specialized formalisms such as business process modelling notation.

The final result is expected to be the definition of a risk-based methodology for joint safety and security in embedded systems. The first considerations are contained in this document, to be expanded in Deliverable D4.2.
1 INTRODUCTION

The purpose of this deliverable in its preliminary version is to describe the chosen concept to come up with the definition of a generic, domain-independent design methodology and process suitable for use during the application case studies in WP5.

During the work on Task 1 of WP4 it became evident that in order to define a generic methodology and process a detailed look on the processes applied in the different domains is helpful. These domain specific processes described for instance in the avionics or automotive standards (e.g. ARP4754A, ISO26262) are currently focussing on safety aspects. Corresponding security aspects are added to the processes in this deliverable.

The structure of the document reflects this bottom-up approach starting with the process definitions displaying the life cycles in the different domains (chapter 2). These domain specific considerations will help to identify the similarities within the processes and this will directly impact the definition of a generic design methodology and process later on in the final version of this deliverable.

The ideas for a generic methodology and process are discussed in chapter 6. The principles this methodology and process has to follow and the technologies which are used to formalize this process are described in chapter 5 and chapter 3 and 4 introduces the relevant background which was considered.


2 DOMAIN SPECIFIC METHODOLOGIES

2.1 AUTOMOTIVE DOMAIN

This chapter gives an outlook onto a combined process regarding safety as well as security aspects. It is based on the safety standard for the automotive domain ISO26262 [51]. UML activity diagrams are used as a formalization method.

2.1.1 Lifecycle of the automotive safety standard ISO26262

The automotive domain safety lifecycle is defined in the standard ISO26262 (see Figure 1).

![Automotive Domain Safety Lifecycle Diagram]

Figure 1: Automotive Domain Safety Lifecycle

The lifecycle depicted in the ISO26262 corresponds exactly to the essential phases of automobile production. It defines three levels: Starting with the Concept Phase it runs through a series of development phases until it describes the activities for production and operation after the Start Of Production (SOP). Therefore the well-known V-Model is used (see Figure 2).
Furthermore the ISO26262 distinguishes between system level development and then below it, both hardware and software development.

### 2.1.2 Approach for a safety-security combined process definition

Figure 3 shows the V-Model of the ISO26262 already enriched by security aspects as an UML activity diagram.
Figure 3: Formalized V-Model for a safety-security combined process

It starts with the Concept Phase which produces the Functional Safety and Security Concept. The System Design activity takes this as an input in order to generate the Technical Safety and Security Concept. This concept will directly influence the development and is also used later to integrate and test the implementation. The produced test results and the formerly defined requirements from the Functional Safety and Security Concept are the base for the safety and security validation activities.

In the following these activities and their inputs and outputs are described in more detail.

**Concept Phase**

The Concept Phase (see Figure 4) comprises the Item Definition, the Hazard and Risk Assessment and the Functional Safety concept. All the activities emphasized in red are security related activities and thereby form an extension to the life cycle defined by the ISO26262.
The Item Definition should collect all information relevant to the safety and security analysis and design for the item:

- purpose and description,
- function(s) and relations between functions,
- requirements for each function,
- draft architecture/outline,
- additional nonfunctional constraints,
- borders or interfaces to other items/systems,
- legal requirements,

whereby an item is a system that implements a function at vehicle level. An important aspect is the identification of some malfunctioning behaviour of the considered system which is later analysed during the Hazard and Risk Assessment. This could be supported by using the HAZOP methodolo-
Analogously the Item Definition is extended by a security related activity which tries to identify possible security violations.

As a result of the Item Definition a functional description of the item, a set of malfunctions and a set of security violations are defined.

The hazard analysis and risk assessment consists of three fundamental steps. The situation analysis and hazard identification considers the potential malfunctioning behavior of the item in combination with certain operational situations which then could lead to a hazardous event. These events have to be classified in the next step, the hazard classification, in respect to their severity (S), their exposure (E) and their controllability. Finally the automotive safety integrity level (ASIL) has to be determined accordingly. The third step is to derive safety goals which have to be fulfilled in order to prevent the hazards.

Similar activities can be defined for the security related aspects. During a threat analysis the threats have to be described and considered within certain operational situations. These threats can then be estimated in respect to their criticality and a corresponding criticality level might be derived. In the end security goals should be defined in order to reduce the risk of an attack.

The safety and security goals are now the input to derive functional safety and security requirements. In this phase first interference analyses have to be undertaken in order to identify their impact on each other.

System Design

The System Design Phase (see Figure 5) comprises the activities to derive technical safety and security requirements out of the functional requirement and to define a corresponding architecture. Again all the activities emphasized in red are security related activities and thereby form an extension to the life cycle defined by the ISO26262.

---

1 This varies from the definition of the Item Definition in the ISO26262. There the usage of the HAZOP methodology is rather a part of the Hazard and Risk Analysis. Nevertheless the ISO26262 allows for the tailoring of the process therefore it is possible to also use the HAZOP here.
In the safety area supporting methods to derive technical requirements as well as to analyse the system architectures are qualitative and quantitative Fault Tree Analysis (FTA) as well as a Failure Mode and Effects Analysis (FMEA).

With regard to the security aspect method like Attack Trees and Security FMEA help to evaluate the system design.

The found design solutions for safety and security have to undergo an interference and impact analysis. Thus it can be assured that some built-in security mechanisms do not cause a malfunctioning of the safety mechanisms and vice versa.

Figure 6 shows that the same activities executed on the system design level are repeated for a corresponding hardware resp. software design.
Figure 6: Safety and security activities of the Hardware/Software Design Phase

The result of the System Design Phase is the Technical Safety and Security Concept which is implemented later during the Development Phase.

The implementation has to be tested and verified against the requirements outlined in the Technical Safety and Security Concept as well as in the Functional Safety and Security Concept.
2.2 **INDUSTRIAL CONTROL DOMAIN**

Within the following chapter a rough overview of the underlying IEC 61508 safety lifecycle is given. To be read as a basis for an upcoming joint safety and security methodology.

2.2.1 **Industrial Domain Safety Lifecycle (IEC 61508)**

The Industrial Domain Safety Lifecycle is based on the standard IEC 61508 – Functional Safety (see Figure 7).

![Figure 7: Industrial Domain Safety Lifecycle – Overview.](image)

2.2.2 **Industrial Domain Safety Lifecycle Overview**

- **Concept Phase.** The goal of this phase is to reach a basic level of understanding for the equipment under control (EUC) and its environment.
- **Overall Scope Definition.** Objective is a definition of system boundaries, describing the relation between EUC and EUC control system (ECS), and the elicitation of a preliminary hazard list.
- **Hazard Analysis and Risk Assessment.** In this phase, hazards and risks for EUC and ECS are identified, safety integrity levels (SIL) determined, as well as the necessary safety functions identified.
• **Overall Safety Requirements.** Target safety integrity requirements and safety function requirements are determined.

• **Overall Safety Requirements Allocation.** Overall safety function requirements are mapped onto designated SRSs.

• **System Safety Requirements Specification.** Objective is the definition of system safety function requirements and system safety integrity requirements.

• **Safety-Related Systems Realization.** The safety-related system (SRS) is implemented conforming to the specification including safety function requirements and safety integrity requirements.

For clarification, the following gives a brief introduction into the objectives and outputs of each corresponding phase.

### 2.2.3 Concept

**Objectives**

- Describe the system with focus on
  - EUC
  - Environment
  - Control Functions

- Make first considerations about hazards:
  - Identify possible sources of hazards
  - Collect Information about known hazards and safety regulations
  - Consider hazards resulting from interaction with other systems

**Output**

- Documentation that…
  - Enables an easy understanding of EUC/Environment/Control Functions concept
  - Provides information about probable preliminary hazards and their implications regarding system safety and safety regulations

### 2.2.4 Overall Scope Definition

**Objectives**

- Define system boundary and relation between EUC and ECS

- Define the scope for the following Hazard and Risk Analysis:
• Physical Equipment
• External Events
• Equipment and systems that are associated with hazards

Output

➢ Documentation:

• Relates EUC and ECS and defines the system boundaries with respect to preliminary identified hazards

2.2.5 Hazard Analysis and Risk Assessment

Objectives

➢ Determine hazards and risks for the EUC

• Organize (independent) assistance personal
• Identify hazards for EUC and ECS
• Establish a risk matrix through risk analysis
• Identify safety-related functions
• Assess the SRS by reliability modeling
• Check if the intended target SIL for the SRS is reasonable
• Check if the already taken risk reduction measures (RRM) are sufficient (ALARP – as low as reasonably practicable)

Output

➢ Safety Report containing:

• Hazard scenarios (with consideration of foreseeable scenarios, event sequences leading to hazards, and misuse) containing consequences and likelihood of hazardous events
• Risk and safety level classification matrix
• Risk reduction measurement (SRS)
• Identified safety-related functions
• Proven reliability diagram for SRS

2.2.6 Overall Safety Requirements

Objectives

➢ Implement a safety specification
2.2.7 Overall Safety Requirements Allocation

Objectives

- Map overall safety function requirements onto designated SRSs
  - Specify the designated SRSs
  - Map safety functions requirements
  - Specify safety integrity requirements for each safety function
  - Map safety integrity requirements
  - Allocate target failure measure and an associated SIL to each safety function

Output

- Safety report containing:
  - Safety function allocation diagram showing the relationship between hazards, safety functions, safety function requirements, SILs and the allocation of safety functions onto SRS or other RRM
  - Table with target failure rate and SIL allocated to safety functions

2.2.8 Overall Safety Requirements Specification

Objectives

- Define system safety requirements
• Define system safety function requirements
• Define system safety integrity requirements

Output

➢ System Safety Requirements Specification containing:

➢ Requirements for safety functions (with associated SILs) fit for development
  • Time and data interfaces
  • Operation modes
  • Information relevant for system safety

➢ Requirements for safety integrity
  • SIL for each safety function
  • Operation mode for each safety function
  • Required duty-cycle, lifetime
  • Possible environmental conditions
  • Constraint and limiting conditions

2.2.9 Safety-Related Systems Realization

Objectives

➢ Implement SRS conforming to specification

➢ Implement the SRS based on
  • System safety function requirements
  • System safety integrity requirements

Output

• SRS Realization
• Implementation documentation

A graphical overview of the shown Industrial Domain Safety Lifecycle is depicted in the following Figure 8.
<table>
<thead>
<tr>
<th>Concept</th>
<th>Overall Scope Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Describe the system with focus on:</strong></td>
<td><strong>Define system boundary and relation between EUC and ECS</strong></td>
</tr>
<tr>
<td>• EUC</td>
<td>• Define the scope for the following Hazard and Risk Analysis</td>
</tr>
<tr>
<td>• Environment</td>
<td>• Physical Equipment</td>
</tr>
<tr>
<td>• Control Functions</td>
<td>• External Events</td>
</tr>
<tr>
<td><strong>Make first considerations about hazards:</strong></td>
<td>• Equipment and systems that are associated with hazards</td>
</tr>
<tr>
<td>• Identify possible sources of hazards</td>
<td></td>
</tr>
<tr>
<td>• Collect Information about known hazards and safety regulations</td>
<td></td>
</tr>
<tr>
<td>• Consider hazards resulting from interaction with other systems</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Overall Safety Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Documentation:</strong></td>
<td><strong>Implement a safety specification</strong></td>
</tr>
<tr>
<td>• Enables an easy understanding of EUC/Environment/Control Functions concept</td>
<td>• Specify overall safety function requirements (functionally)</td>
</tr>
<tr>
<td>• Provides information about probable preliminary hazards and their implications regarding system safety and safety regulations</td>
<td>• Specify target safety integrity requirements for each safety function (e.g. required risk reduction)</td>
</tr>
<tr>
<td></td>
<td>• Specify overall safety integrity requirements</td>
</tr>
<tr>
<td></td>
<td>• Determine SRSs (included in ECU or separate SRS) or other RRM</td>
</tr>
<tr>
<td></td>
<td>• Assign SILs to safety functions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hazard Analysis and Risk Assessment</th>
<th>Safety Report containing:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Determine hazards and risks for the EUC</strong></td>
<td>• Hazard scenarios (with consideration of foreseeable scenarios, event sequences leading to hazards, and misuse) containing consequences and likelihood of hazardous events</td>
</tr>
<tr>
<td>• Organize (independent) assistance personal</td>
<td>• Risk and safety level classification matrix</td>
</tr>
<tr>
<td>• Identify hazards for EUC and ECS</td>
<td>• Risk reduction measurement (SRS)</td>
</tr>
<tr>
<td>• Establish a risk matrix through risk analysis</td>
<td>• Identified safety-related functions</td>
</tr>
<tr>
<td>• Identify safety-related functions</td>
<td>• Proven reliability diagram for SRS</td>
</tr>
<tr>
<td>• Assess the SRS by reliability modeling</td>
<td></td>
</tr>
<tr>
<td>• Check if the intended target SIL for the SRS is reasonable</td>
<td></td>
</tr>
<tr>
<td>• Check if the already taken RRM are sufficient (ALARP)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Safety specification with:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Safety Report containing:</strong></td>
<td>• Safety function requirements (independent from the implementation technology)</td>
</tr>
<tr>
<td>• Hazard scenarios (with consideration of foreseeable scenarios, event sequences leading to hazards, and misuse) containing consequences and likelihood of hazardous events</td>
<td>• Safety integrity requirements</td>
</tr>
<tr>
<td>• Risk and safety level classification matrix</td>
<td>• Table with SILs assigned to safety functions</td>
</tr>
<tr>
<td>• Risk reduction measurement (SRS)</td>
<td></td>
</tr>
<tr>
<td>• Identified safety-related functions</td>
<td></td>
</tr>
</tbody>
</table>
### Overall Safety Requirements Allocation

Map overall safety function requirements onto designated SRSs
- Specify the designated SRSs
- Map safety functions requirements
- Specify safety integrity requirements for each safety function
- Map safety integrity requirements
- Allocate target failure measure and an associated SIL to each safety function

### System Safety Requirements Specification

Define system safety requirements
- Define system safety function requirements
- Define system safety integrity requirements

### Safety-Related Systems Realization

Implement SRS conforming to specification
- Implement the SRS based on
- System safety function requirements
- System safety integrity requirements

### Output:

**Safety report containing:**
- Safety function allocation diagram showing the relationship between hazards, safety functions, safety function requirements, SILs and the allocation of safety functions onto SRS or other RMs
- Table with target failure rate and SIL allocated to safety functions

**System Safety Requirements Specification**
- Requirements for safety functions (with associated SILs) fit for development
- Time and data interfaces
- Operation modes
- Information relevant for system safety
- Requirements for safety integrity
- SIL for each safety function
- Operation mode for each safety function
- Required duty-cycle, lifetime
- Possible environmental conditions
- Constraint and limiting conditions

**Output:**
- SRS Realization
- Implementation documentation

---

**Figure 8: Industrial Domain Safety Lifecycle – Detailed**
3 RELEVANT BACKGROUND

3.1 CHESS METHODOLOGY

A specific support for software development comes from the CHESS (Composition with Guarantees for High-integrity Embedded Software Components Assembly) ARTEMIS JU Call 2008 project, aimed at developing solutions to property-preserving component assembly in real-time and dependable embedded systems [18]. The CHESS focus was to build languages for modelling of non-functional properties, develop tools for evaluation of these properties, adapt component infrastructures for the integration of real-time and dependable patterns, and validate the approach through multi-domain case studies.

The CHESS project has promoted the adoption of Component-based Development and Model Driven Engineering to support the development of High Integrity Systems. The combination of these two approaches enables better mastery of complexity, increased reusability, robustness and quality, as well as easier maintenance, thus reducing the costs and risks of development and deployment. For the specific safety and security oriented needs of SESAMO we propose to build upon the extensions that were introduced to the CHESS methodology and tool for the SafeCer (Safety Certification of software-intensive systems with reusable components) ARTEMIS JU Calls 2010 and 2011 project [49], which we describe in 3.1.1.

The CHESS project provided a methodology and the related support technology that is based on four distinct pillars:

1. **Separation of concerns** by design views enacted directly in the user modelling space
2. A cross-domain **component model** and a coherent component based development methodology
3. **Correctness-by-construction**, by adoption of a declarative approach to the specification, verification and implementation of non-functional concerns.
4. Strict separation between the platform-independent modelling performed by the user and the platform-specific modelling, which is derived by formally verified transformation engines as a correct-by-construction product.

The separation of concerns concept is a long-known best practice for the development of complex systems, according to which software system must be decomposed into parts that overlap in functionality as little as possible (for more details see section 5.1.3). CHESS offers the developer an environment in which this best practice has explicit methodological and tool support. The methodological and technical means by which CHESS provides the developer with support for separation of concerns is the popular, well-accepted modelling concept of views, most prominently acknowledged in modern modelling languages such as UML. The CHESS Methodology associates a distinct view with a distinct concern pertinent to system software modelling, so that the developer required to address a particular concern (e.g. timing aspects) is provided with a working environment devoted entirely to and specialized for that concern. In fact, the developer is only allowed to work on one concern at a time – with access to the modelling environments of other concerns limited to read-only permissions.

The view-oriented development approach [7] of CHESS has required advances in component representation. The CHESS **component model** extends traditional component models by explicitly separating functional from non-functional aspects. Components internals are modelled in a dedicated
view, in which the designer describes only the functional aspects. A distinct design view permits to annotate the component description with the declaration of the intended non-functional attributes. The declarative specification of those attributes is used for the automated generation of a container, which can be considered as a component wrapper that is responsible for the realization of the non-functional attributes declared in the design model. As a consequence, in the CHESS approach pure components encompass functional concerns only; in particular, tasking or time-related constructs are a container concern.

At the same time components (and containers) are independent of interaction concerns, which are dealt by a dedicated software entity: the connector. The binding between two components in the design model is used to generate the connector, which manages the communication between the components – which is actually a mediated communication between their containers. The use of connectors ensures that components and containers do not require any adaptation under different binding and deployment specifications.

As a result, separation of concerns is directly supported in the CHESS model both at specification and implementation level: the former, by way of design views to address distinct concerns; the latter with the careful allocation of different concerns to distinct software entities.

In particular the component model developed in CHESS is fully consistent and compatible with the space component model that is part of the OBSW-RA developed in the CORDET2 project (http://cordet.gmv.com/).

The CHESS tool-chain guarantees the implementation of the correct by construction [22] paradigm by supporting:

- the analysis and verification of non-functional properties (predictability and dependability) in the component-based software system modelling and assembly
- the automatic generation of code which is consistent with respect to the properties asserted at model/analysis level and the deployment on the target platform.

Moreover the CHESS Infrastructure, with its dedicated execution platforms, is developed to guarantee that the non-functional properties asserted at model level are provably preserved at run time (e.g. by means of specific monitoring functionality).

By accompanying the developer every step of the way, from presenting clean, uncluttered, single-concern design spaces to providing constant, informative verification feedback to his/her modelling actions, the CHESS Methodology encourages a disciplined, productive development in the complex world of dependable real-time embedded systems through a correct-by-construction derivation approach, and the work of the developer results at the end in concrete, platform-specific implementations with high-integrity runtime guarantees.

### 3.1.1 SafeCer extensions

The main goal of the SafeCer project is to increase efficiency and reduce time-to-market by composable safety certification of safety-relevant embedded systems. A primary objective is to provide support for system safety arguments based on arguments and properties of system
components, as well as to provide support for generation of corresponding evidence in a similar compositional way.

In the following we describe the principal contributions that the SafeCer project provided to the evolution of the CHESS methodology and toolset. The SafeCer extensions to the CHESS methodology are mainly in the perspective of a systematic formalization of the process with a strong focus on early verification and are thus in line with SESAMO’s needs.

With SafeCer the CHESS methodology extended its concerns from concentrating only on the software level to embracing also the **System level** design phase. A “System View” was introduced for this purpose in the CHESS editing environment where to model the System architecture with contracts, relying on a specific profile of SysML. System level design entities are linked to corresponding software level entities by means of the SysML «allocate» dependency: this way system and software co-engineering is implemented as a seamless process enabling the modeller to design system and software using one tool (CHESS).

The adoption of a **contract based** approach is an important contribution of SafeCer to the CHESS methodology: system and components’ properties are specified into formal expressions structured in contracts, i.e. pairs of assumption and guarantee, where the assumption is a property that must be satisfied by the (system/component) environment, while the guarantee is a property that must be satisfied by the system/component (provided that the environment satisfies the assumption).

According to the extended CHESS methodology, **formal verification and validation** steps shall be performed right from the earliest stages of the process, implementing an enhanced version of the traditional V-model development process as depicted in Figure 9. Indeed formal techniques, applied early in the development process, have the potential to reduce the overall verification costs, increase the level of consistency of models and increase the level of automation of the system lifecycle, through formal specification of system requirements, model refinements and formal reasoning. The application of such techniques from the early stages of the design process implements the so-called “left-shift”: a systems engineering approach that promotes earlier analyses and decision-making, thereby increasing the effectiveness of the engineering of the system and reducing the overall development and maintenance costs.

The whole development process envisaged in the CHESS extended methodology runs through different conceptual models (from the higher level functional architecture and logical architecture, down to the lower level physical architecture with specification of Software and its allocation on hardware). The modelling of different conceptual level architectures is responsibility of professionals with different profiles. When stepping from one conceptual level to the next (e.g. from the functional architecture to the logical architecture) a new model is designed 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.

Within each conceptual level the model describes a **step-wise refinement** with the hierarchical decomposition of the system into components and of components into subcomponents. Alongside with this architectural refinement, also the components’ contracts are refined into a collection of contracts over subcomponents. Such refinement is subject to formal verification and is a key-point in the overall verification process, according to the approach first envisaged in the ESA funded FoReVer study [19] and further elaborated within the SafeCer project.
The core of the step-wise refinement formal verification reasoning is based on the following points:

- the system is correct if it satisfies the requirements associated to the system block
- every block satisfies the associated requirements if it satisfies the corresponding contracts (i.e. the contacts associated to a block corresponds to the requirements associated to the block)
- a parent block satisfies a contract if the contract refinement is correct and the sub-blocks satisfy their contracts.

Based on this reasoning we can state that if the refinement steps are proven correct by formal verification, any implementation of the leaf components that satisfies the components contracts can be used to implement the system satisfying the system’s contracts. This provides a powerful aid in the model based system development process and strongly encourages the use of standard qualified components.

In synthesis according to the extended CHESS methodology, formal verification and validation steps shall be performed right from the earliest stages of the process, implementing an enhanced version of the traditional V-model development process as depicted in Figure 9.

**Figure 9 Extended V-model in the CHESS development process**

### 3.2 UML Based Profiles (MARTE, SysML)

UML is a general-purpose modelling language for specifying, visualizing, constructing, and documenting the artefacts of the development of large and complex (not only software) systems. UML can be tailored to specific domains or platforms through a set of extension mechanisms. These mechanisms allow the customization and extension of the UML syntax and semantics while maintaining interoperability across tools. Most relevant UML profiles today are SysML, and MARTE (Modelling and Analysis of Real-Time and Embedded systems).

**SysML** [46] adapts UML to offer a variety of modelling elements and diagrammatic views to support system engineering modelling activities: it can be used for specifying, analysing, designing,
and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities.

SysML provide support for:

- Capturing the system’s information in a precise and efficient manner that enables it to be integrated and reused in a wider context
- Analysing and evaluating the system to identify and resolve system requirements and design issues, and to support trade-offs
- Communicating the system’s information correctly and consistently among various stakeholders and participants

**MARTE** UML profile [45] aims to replace UML capabilities for model-driven development of Real-Time and Embedded Systems (RTES), and for analysing schedulability and performance of UML specifications. It provides valuable capabilities such as the support for specification, design, and verification/validation stages, the definition of non-functional properties, time and time related concepts and analysis frameworks.

**MARTE** is comprised of two main parts, a design model package to model hardware and software aspects of real-time and embedded systems and an analysis model package to annotate application models so as to permit quantitative analysis of a range of properties of interest. These two parts take a common and consistent approach to the description of time and to the use of concurrent resources. Application modelling is based on interacting component blocks for structural aspects; for behaviour, block-diagrams are amenable to activity charts, or finite-state machines. This level of modelling is further decorated with timing and other non-functional attributes. Execution platform modelling comprises the description of the hardware and (middleware) software layers and interconnects that compose the platform. Platform components can be described at the same level of abstraction as the application, and they may thus contain also timing information along with structural and behavioural aspects. The allocation model describes the association matching applicative functions onto execution platform resources.

### 3.3 EVITA

The objective of the EVITA project was to design, verify and prototype an architecture for automotive on-board networks, where security relevant components are protected against tampering and sensitive data is protected against compromise. The project was conducted from July 2008 to December 2011 and was financed from EU’s 7th framework programme.

The project’s main scientific and technological results are the following:

- Security requirements analysis
- Secure on-board architecture design and prototype
- A SysML-based environment for formal verification of Safety and Security Properties

These points are covered in more detail in the following subsections.

#### 3.3.1 Security requirements analysis

While the safety requirement analysis process is well understood and described in the ISO 26262 standard, security requirements analysis is a more or less new topic in the automotive domain. The EVITA project considered the following five use cases:
• communication between cars,
• communication between car and road infrastructure,
• integration of mobile devices such as phones,
• aftermarket applications (e.g. feature activation) and
• workshop / diagnostics processes.

They identified possible attack targets (ECUs, sensors, actuators and communication links) and stakeholders (vehicle occupants, other road users, vehicle manufacturers and suppliers, infrastructure providers and civil authorities). Based on this, the aim was to develop a process to derive, justify and prioritize security requirements.

In summary, the following generic security requirements can be identified in vehicular systems:

1. Integrity of hardware security modules
2. Integrity and authenticity of in-vehicle software and data
3. Integrity and authenticity of in-vehicular communication
4. Confidentiality of in-vehicular communication and data
5. Proof of platform integrity and authenticity to other (remote) entities
6. Access Control to in-vehicle data and resources

The definition of the whole Model Based Security Requirement Engineering process [7] was one of the remarkable outcomes of EVITA, which should be considered within SESAMO.

### 3.3.2 Secure on-board architecture design and prototype

The other achievement of EVITA was development of secure on-board architecture. The main and novel component in the proposed architecture is Hardware Security Module (HSM), which provides secure and efficient implementation of cryptographic operations and secure implementation of various other security mechanisms. Three different versions of HSM were proposed and implemented in FPGA. The different versions are meant to be used in different scenarios:

1. **HSM Full Version** is the most capable and also most expensive module. It is expected that only one such module will be present in a single vehicle. This module will be mainly used for extra-vehicular communication such as in vehicle-to-infrastructure and similar applications.

2. **HSM Medium Version** will secure in-vehicle communication such as communication with power-train components.

3. **HSM Light Version** is the cheapest and least capable HSM. It will be present in security critical sensors and actuators.
Besides defining hardware architecture, the aim of EVITA was to integrate the software application interfaces to HSM into AUTOSAR.

The EVITA project also developed several extensions of automotive protocols that provide additional security features such as integrity and authenticity to in-vehicle communication.

### 3.3.3 A SysML-based environment for formal verification of Safety and Security Properties

The EVITA project also produced AVATAR (Automated Verification of reAl Time software), which is a graphical modeling environment based on SysML. AVATAR models are able to express both safety and security properties and provide means for their formal verification.

The AVATAR models contain an attacker model by default and it is not required to model attackers explicitly. Safety and security properties are expressed in terms of SysML requirement diagrams. There, safety properties are further refined within parametric diagrams and security properties are described within specific pragmas of block diagrams.

An AVATAR model can be automatically transformed to UPPAAL[8] in order to prove safety properties and the ProVerif tool to prove security properties. By using ProVerif they are able to verify confidentiality and authenticity properties.

Moreover the AVATAR methodology was included into an open-source UML toolkit called TTOOL[9] that now, provides AVATAR editing capabilities, and offers a press-button approach for property proof.

**The AVATAR Methodology[10]:**

The AVATAR profile reuses eight of the SysML diagrams (Package diagrams are not supported). AVATAR supports the following **methodological phases** which are depicted in Figure 10 and discussed below:

![Figure 10: The AVATAR Methodology Stages](image-url)
1. **Requirement capture:**
   Requirements and properties are structured using AVATAR Requirement Diagrams. At this step, properties are just defined with a specific label as test cases.

2. **System analysis:**
   A system may be analyzed using usual UML diagrams, such as Use Case Diagrams, Interaction Overview Diagrams and Sequence Diagrams.

3. **System design:**
   The system is designed in terms of communicating SysML blocks described in an AVATAR Block Diagram, and in terms of behaviors described with AVATAR SMDs.

4. **Property modeling:**
   The formal semantics of properties is defined within TEPE (TEmporal PProperty EExpression Language)[11]Parametric Diagrams (PDs). Since TEPE PDs involve elements defined during the system design phase e.g., a given integer attribute of a block), TEPE PDs may be defined only after a first system design has been performed.

5. **Formal verification:**
   Safety properties can finally be verified over the system design, and for each test case.

6. **Code generation:**
   This feature can finally be used to generate a fully executable code. The latter can be compiled and executed on the SoCLib[12] prototyping platform directly from within TTool.
Modelling and proof of security properties within AVATAR

Security properties are treated in two different stages of the previously shown AVATAR methodology, at requirement capture stage, and at property modeling stage. The first part, the SysML based security requirement engineering process, is described in [7] while further details on the property definition and formal verification can be found within [11]. Basically the AVATAR profile defines a new stereotype for security requirements to make a clear distinction between functional requirements and security requirements of the system. A ‘Kind’ parameter is defined to specify the category of the security requirement such as, confidentiality, access control, integrity, freshness, etc[14]. Following example taken from the EVITA use case shows a security requirements block diagram for a FLASH programming process.

![AVATAR security requirements block diagram for a FLASH programming process](image)

AVATAR relies on ProVerif [15] which provides a formal framework for security proofs. ProVerif is a toolkit that relies on Horn Clauses resolution for the automated analysis of security properties over cryptographic protocols. ProVerif takes as input a set of Horn Clauses, or a specification in pi-calculus (a process algebra) and a set of queries. ProVerif outputs whether each query is satisfied or not. In the latter case, ProVerif tries to identify a trace explaining how it came to the conclusion that a query is not satisfied.

TTool fully supports AVATAR, including its security extensions. No or little knowledge of ProVerif is necessary to perform security proofs. Even if ProVerif is a strong formal approach, it only targets verification of confidentiality and authenticity properties thus limiting AVATAR security proof capabilities.

Modelling and proof of safety properties within AVATAR

AVATAR includes TEPE[5], a graphical TEmporal Property Expression language based on SysML parametric diagrams. TEPE enriches the expressiveness of other common property languages in particular with the notion of physical time and unordered signal reception. In TEPE, each
property is expressed as a graph of Signals, Attributes, Constraints (Equations, Logical Constraints, Temporal Constraints) and properties.

Following example shows a TEPE description of the safety properties of an elevator system[16]:

![Figure 12: TEPE description of safety related properties of an elevator system](image)

Four functional safety-related requirements have been identified and modeled in a Requirement Diagram:

- **Req1**: The door does not open when the elevator is moving.
- **Req2**: The elevator does not depart with an open door.
- **Req3**: The operational profile requires the elevator to accelerate after being set in motion and to decelerate before stopping.
- **Req4**: Deceleration must be accomplished between 1 and 5 seconds before the selected floor is reached.

The TEPE language can be used to model and verify safety properties. The formal semantics of AVATAR is defined by model transformation to timed automaton. TTool implements that transformation, and relies on the UPPAAL model checker to evaluate properties. More precisely, the following properties can be directly evaluated in TTool with a press-button approach:[10]

- **Deadlock freedom**: A deadlock situation arises when no logical progress is possible in the application. The property is satisfied if no deadlock situation is possible.
- **State Reachability**: A state St within a SMD is reachable, if there exists a sequence of traversable transitions starting from the initial state and leading to the state St.
- **State Liveness**: A state St within an SMD satisfies liveness property if St is eventually reached independently of the sequence of transitions that is traversed from the initial state.

### 3.4 TOLERABILITY OF RISK AND ALARP

The design of high hazard installations and their supporting safety or security systems focuses on minimizing and controlling risks [26]. The cost and rigour of such activities must be proportionate to those risks. Calculations of risk, taking into account severity and likelihood, must demonstrate that the risk has been reduced to a tolerable level.

The “ALARP principle” (the principle that certain risks have to be demonstrated to be “As Low As Reasonably Practicable”) is key to discussing risks and the stopping rules associated with additional design and operational measures. The ALARP principle is based on the assumption that it is possible to compare marginal improvements in safety (marginal risk decreases) with the marginal costs of the risk reduction measures. Safety risks may offer this possibility when they are quantified (e.g.,...
in terms of event probability and of radiation releases), and when the failure rate improvements of the systems controlling the relevant events can be evaluated. Note that the application of the ALARP concept does not necessarily need a quantification of risk reduction. For example, the simple addition of a further safety feature, which costs relatively little, may be obviously worthwhile—qualitative judgements of this nature can often be readily made. Also, marginal does not mean one just considers incremental or small perturbations to the design: sometimes, creative design changes (e.g. substitution of hazardous materials with benign ones) are needed to justify that the risks have been reduced to ALARP.

ALARP found its expression in the well-known “carrot diagram” (see Figure 13 below), which has become the standard means for the exposition of the principle. Levels of risk are divided into three bands, with the width of the wedge representing the level of risk. There are two significant boundaries: the upper one, beyond which risks are not acceptable at all and cannot be justified on any grounds, and a lower one, beyond which risks are considered negligible and no detailed assessment is required. Regulators do not usually require further action to reduce risks unless reasonably practicable measures are available. Within these two boundaries is the ALARP region. At the upper, more risky, end of the ALARP region the risks are only tolerable if costs are judged grossly disproportionate to the risk reduction gained.

![Figure 13: The ALARP principle](image)

A key part of assessing tolerability is the effective communication of safety and risk. Many sectors with safety critical systems use the concept of a safety case to facilitate this.

### 3.5 SAFETY CASES

Safety cases are an important part of goal based safety regulation and corporate governance [26]. Explicit safety cases are required for military systems, the off shore oil industry, rail transport and the nuclear industry.
An early definition of a safety case [36] was

“a documented body of evidence that provides a convincing and valid argument that a system is adequately safe for a given application in a given environment”

Each domain has variants of definitions. For example, a nuclear safety case is defined by the HSE [37] as

“…the totality of documented information and arguments which substantiates the safety of the plant, activity, operation or modification in question. It provides a written demonstration that relevant standards have been met and that risks have been reduced as low as reasonably practicable (ALARP).”

In regulated sectors, the licensee is legally responsible for the safety case. Given the magnitude and complexity of the legislative and technical requirements that have to be met, safety cases have to be structured in a logical manner and be demonstrably adequate.

Safety cases can be seen to support the following [38]:

- **Reasoning and argumentation.** A safety case can be seen as an over-arching framework that allows us to argue whether the claims are substantiated by the evidence. The case might be mainly narrative, using prose to explain the connections between claims and evidence. However cases deal with highly technical subjects and hence they might use specialist notations from the particular discipline concerned (e.g. from fluid mechanics, computer science). The case will then integrate a selection of technical analyses and other evidence using a formal or graphical notation to show whether the claims have been met; how the evidence is integrated; and the overall structure of the case and the thrust of the argument.

- **Negotiation, communication, trust.** The safety case represents a boundary object between the different stakeholders who have to agree (or not) the claims being made about the system. To this end it has to be detailed and rigorous enough to effectively communicate the case and allow challenges and the subsequent deepening of the case. A safety case has to support an argument that the requirements placed upon a system are met. As such, the safety case contains claims about the properties of the system and, following a systematic approach, has arguments that demonstrate that these claims are substantiated or rebutted by evidence.

Thus, to implement a safety case we need to

- make an explicit set of claims about the system
- produce the supporting evidence
- provide a set of arguments that link the claims to the evidence
- make clear the assumptions and judgements underlying the arguments
- allow different viewpoints and levels of detail

Current safety case practice makes use of the basic approach developed by Toulmin [39] where claims are supported by evidence and a “warrant” or argument that links the evidence to the claim, as shown in Figure 14. There are variants of this basic approach that present
the claim structure graphically such as Goal Structuring Notation (GSN) [40] or Claims-Arguments-Evidence (CAE) [26].

There are several different ways of constructing such a justification. The three main approaches can be characterized in terms of a safety justification “triangle” [8]:

- Claims about the systems’ safety behaviour (positive properties).
- The use of accepted standards and guidelines.
- Analysis of potential vulnerabilities (negative properties).

The first approach is claim-based—where specific safety claims for the systems are supported by arguments and evidence at progressively more detailed levels. The second approach is based on demonstrating compliance to a known safety standard. The final approach is a vulnerability-based argument where it is demonstrated that potential vulnerabilities within a system do not constitute a problem—this is essentially a “bottom-up” approach as opposed to the “top-down” approach used in goal-based methods. These approaches are not mutually exclusive, and a combination of approaches can be used to support a safety justification, especially where the system consists of both off-the-shelf (OTS) components and application-specific elements.

Security considerations have an impact on each aspect of the safety justification triangle. It is necessary to make claims about security properties as well as safety properties, demonstrate compliance to both security and safety standards, and consider a broader set of potential threats and vulnerabilities. The hazards remain the same but the judgments we make about the likelihood of a hazard leading to an accident might be different because we are no longer dealing with a benevolent threat model.

To summarise, the motivation for a safety case is to

- provide an assurance viewpoint - for efficient review
- provide a focus and rationale for activities - leading to efficient analysis and evaluation
- provide a reviewable approach - so that all stakeholders can be involved
- demonstrate discharge duty to public and shareholders
- allow interworking between standards and innovation
The emphasis should be on the behaviour of the product and not just the process used to develop it: a useful slogan is "What has been achieved not how hard you have tried".
4 CONCEPTS AND TERMINOLOGY

In order to create systems that are both safe and secure, we need to ensure that we can deploy the large amount of expertise and work that has been done separately in each domain. The safety and security domains have much in common, but these commonalities are obscured by the use of different concepts and terminologies. To achieve a shared understanding of the key concepts within each domain, we need to establish a lingua franca or even a common ontology.

In this chapter we report on some initial work that is aimed at developing a unified conceptual model for dealing with safety and security. In particular, we look at the following issues:

- terminologies and taxonomies
- levels and classification
- security controls

We first discuss the importance of terminology and taxonomy and review the definitions of security and safety with respect to the dependability taxonomy[26] developed by IFIP WG 10.4. We continue by discussing some more specific aspects of security and safety, namely the need to categorise behaviour and assurance in terms of levels, and the concept of security controls.

Based on our analysis, we draw the following conclusions:

- Although there is considerable variation in terminology both within and between the safety and security communities, the IFIP WG 10.4 dependability taxonomy offers a means for defining a consistent set of terms. In particular, it makes a clear distinction between cause and effect and highlights the need to be clear about system boundaries.

- A generic risk identification, analysis, and management life cycle provides a framework around which both safety and security activities can be structured. However, the relationship between security controls and safety mitigations needs to be explored in more detail. In particular, the concept of a security control embraces a wide range of different interventions covering process, product and organisation. In principle it should be possible to relate safety mitigations to security controls, but in order to perform such an analysis, it will be necessary to define a common way of classifying controls and mitigations.

- There is a need to explore further how security risks can be addressed according to the ALARP principle, especially given the uncertainties and changing nature of threats. The issue of defining what is “reasonably practicable” for a security measure in a changing environment needs to be addressed.

4.1 TERMINOLOGIES AND TAXONOMIES

It is important to be clear about what we mean by safety and security, particularly if we are attempting to identify what the two domains have in common and how they differ.

The dependability taxonomy presented in [26] provides a way of addressing this issue. According to this taxonomy, safety and security can both be viewed as forms of dependability, which is defined as “the ability of a system to deliver a service that can justifiably be trusted”.

The service delivered by a system is its behaviour as perceived by its users, which might be other systems. Correct service is delivered when the system implements the intended system function. A
D4.1: Integrated Design and Evaluation Methodology

failure occurs when the delivered service deviates from the correct service. The underlying cause of the failure is called a fault and the manifestation of the fault within the system is called an error. The failure of a component at one level of the system manifests itself as a fault at the next level of the system, so there is a causal chain of fault propagation between different levels of the system.

The dependability taxonomy includes a detailed classification of different kinds of faults and failures, which can be used to explain the distinction between safety and security.

4.1.1 Security terminology

Security is concerned with protecting systems against malicious attacks that seek to compromise the confidentiality, integrity or availability of the system.

Historically, the security community has been distinct from the dependability community and has developed its own terminology, but the most recent version of the dependability taxonomy attempted to document a minimum consensus on the concepts of dependable and secure computing in order to facilitate more technical interaction between the two communities.

In particular, security can be considered to be a form of dependability that focuses on the attributes of confidentiality, integrity and availability in the presence of malicious faults. However, security terminology uses a variety of terms such as attack, vulnerability and intrusion to describe security-related faults, failures and errors.

An intrusion is a malicious interaction fault that compromises the security of a system [27]. This is not the same as an attack because a system can be attacked unsuccessfully. If the system can withstand attack, intrusions can be prevented. In other words, an attack is an intrusion attempt, and an intrusion is the result of a successful attack.

In fact, every intrusion has two underlying causes:

1. a malicious act or attack that attempts to exploit a weakness in the system
2. at least one weakness, flaw or vulnerability in the system that the attacker is able to exploit successfully

Note that vulnerabilities may be introduced during development of a system or during operation. Furthermore, such vulnerabilities can be introduced accidentally or deliberately, with or without malicious intent.

To summarise, the security concepts of attack, vulnerability and intrusion can be expressed in terms of dependability concepts as follows:

- **attack** – a malicious interaction fault, through which an attacker aims to deliberately violate one or more security properties; an intrusion attempt.
- **vulnerability** – a fault created during development of the system, or during operation, that could be exploited to create an intrusion.
- **intrusion** – a malicious, externally-induced fault resulting from an attack that has been successful in exploiting a vulnerability.
4.1.2 Safety terminology

According to the dependability taxonomy, safety is an attribute of dependability. In this sense, safety can be considered to be a particular kind of dependability that focuses on failures of systems that are designed to protect against loss of life or damage to the environment. In this context, it is interesting to note that within the railway community, safety is considered to be one of a set of system properties that need to be considered, namely reliability, availability, maintainability and safety (RAMS), each of which is a different attribute of dependability.

However, the safety community uses its own terminology for discussing safety-related faults, errors and failures. In particular, it talks about accidents, incidents and hazards. An accident is an undesired event that results in loss or damage whereas an incident is an event that involves no loss (or only minor loss) but has the potential for loss in different circumstances. A hazard is a potential source of harm, a hazardous situation is a situation in which harm is possible, and a hazardous event is an event that may result in harm. A harmful event occurs when a hazardous situation or hazardous event results in harm.

It is also possible to make a distinction between a safe failure and a dangerous failure. If there is a known safe state, the design of the system can be biased towards failing in this direction. This is illustrated in Figure 15, which is taken from the Adelard Safety Case Development Manual [28]:

![Figure 15: Model of system failure behaviour](image)

4.1.3 Discussion

Terminology can be a real source of confusion between different communities and even within the same community, particularly if there is a lack of consensus or clarity about the concepts being discussed. However, although using the same word to mean different things or different words to mean...
the same thing can cause confusion, it is ultimately the underlying concepts that matter, not the labels that are attached to them.

Safety and security often involve multiple layers of defence and multiple systems, so it is important to be clear about issues such as system boundaries, the scope of each failure, and fault containment; in other words, mechanisms that prevent the failure of one system leading to the failure of another.

Broadly speaking, safety is concerned with protecting the environment from the system whereas security is concerned with protecting the system from the environment. However, since the environment can be viewed as a system in its own right, this suggests a duality or even an equivalence between safety and security – both are concerned with protecting one system from another.

In both safety and security something is being protected and the issue is how to ensure that the protection is adequate. Thus, it is necessary to identify what is being protected, what it is being protected against, what might cause the protection to fail, what the consequences of failure might be, what can be done to reduce those consequences to an acceptable level, and how to determine whether this has been achieved.

Safety is concerned with preventing accidents by identifying potential weaknesses, initiating events, internal hazards and potentially hazardous states and then identifying and applying appropriate mitigations to reduce the risks to a tolerable level. Security is concerned with protecting assets against internal and external threats and vulnerabilities that compromise the asset in some way. Assets are protected using controls that reduce the risk of compromise to an acceptable level.

Juxtaposing these two accounts of safety and security suggests the following equivalences:

- accident = compromise that results in a loss
- weakness = vulnerability
- initiating event = external threat
- hazardous substance (sometimes referred to as just hazard) = internal threat
- hazardous state = compromise
- mitigation = control
- prevention = protection

These equivalences are approximate but indicate the essential similarities between the two disciplines. However, they also highlight some differences in emphasis. For example, although accident and compromise are broadly equivalent, security considers different kinds of compromise. In particular, there is a notion of security attributes: confidentiality, integrity and availability. Accidents are not classified in the same way, although it is easy to conceive of a classification based on what is being harmed (e.g., people, property, environment) as well as the cause of the harm (e.g., loss of integrity). Similarly, security considers compromises to individual assets whereas safety just considers accidents in a broad sense. This reflects the fact that security is concerned with protecting a known set of assets from external threats whereas safety is concerned with preventing a known system from causing harm to an external environment. There are also parallels between safety mitigations and security controls that need to be explored further.
Some aspects of safety and security are domain specific. For example, nuclear safety distinguishes between different kinds of accident (“uncontrolled shutdown”, “release of radiation”, “core meltdown”, “containment breach”). Similarly, military security distinguishes between the consequences of compromising particular assets by grading assets using a classification scheme that reflects their importance (“confidential”, “restricted”, “secret”).

4.2 LEVELS AND CLASSIFICATION

In order to make objective statements about safety and security properties, it is useful to have a scale against which these properties can be measured. A particular level of safety or security can then be defined as a point on this scale.

For example, ISO/IEC 61508 defines the concept of “safety integrity level” whereas ISO/IEC 62443 defines the concept of a “security level”:

\[
\text{safety integrity level} = \text{discrete level (one out of a possible four), corresponding to a range of safety integrity values, where safety integrity level 4 has the highest level of safety integrity and safety integrity level 1 has the lowest}
\]

\[
\text{security level} = \text{level corresponding to the required effectiveness of countermeasures and inherent security properties of devices and systems for a zone or conduit based on assessment of risk for the zone or conduit}
\]

In addition to having a scale for expressing the desired level of safety or security, it is also useful to have a scale for expressing the desired level of confidence in the safety or security of the system. For example, ISO/IEC 15408 defines a (security) evaluation assurance level as:

\[
\text{evaluation assurance level} = \text{set of assurance requirements [...] representing a point on [a] predefined assurance scale}
\]

Before we explore the concept of level in more detail, we first explain these definitions in more detail by summarizing the relevant parts of the standards in question.

4.2.1 Safety integrity levels

IEC 61508 expresses safety levels in terms of a safety integrity requirement:

\[
\text{safety integrity} = \text{the probability of an E/E/PE safety-related system satisfactorily performing the specified safety functions under all the stated conditions within a stated period of time}
\]

Depending on the frequency of demands, the target failure measure is interpreted as either the average probability of dangerous failure on demand (for low demand systems) or the average frequency of dangerous failures (for high demand and continuous demand systems).

IEC 61508 specifies four levels of performance for safety functions. These are called Safety Integrity Levels (SILs) and correspond to a range of safety integrity values. SIL 1 is the lowest level of safety integrity and SIL 4 is the highest level. The range of safety integrity values associated with each SIL is shown in Table 1.
4.2.2 Security levels

The ISA 99 (IEC 62443) family of standards uses security levels as a qualitative approach to expressing security requirements. As shown in Table 2, there are four different security levels, which are characterised in terms of the threats that they protect against. The definitions of these security levels uses “intentionally vague” terms like “causal”, “coincidental”, “simple”, “sophisticated” and “extended” in order to be generally applicable.

<table>
<thead>
<tr>
<th>Security Level</th>
<th>Level of protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Protection against casual or coincidental violation.</td>
</tr>
<tr>
<td>2</td>
<td>Protection against intentional violation using simple means with low resources, generic skills, and low motivation.</td>
</tr>
<tr>
<td>3</td>
<td>Protection against intentional violation using sophisticated means with moderate resources, system specific skills and moderate motivation.</td>
</tr>
<tr>
<td>4</td>
<td>Protection against intentional violation using sophisticated means with extended resources, system specific skills and high motivation.</td>
</tr>
</tbody>
</table>

Table 2: ISA 99 Security levels

4.2.3 Evaluation assurance levels

The philosophy that underpins the Common Criteria (ISO/IEC 15408) approach to assuring the security of IT systems is that assurance should be based on an evaluation (active investigation) of the IT system that is to be trusted, and that greater assurance arises from the application of greater evaluation effort. Thus, the standard defines a series of Evaluation Assurance Levels (EALs) that are ranked according to the scope, depth and rigour of the evaluation, as shown in Table 3.

<table>
<thead>
<tr>
<th>Evaluation Assurance Level</th>
<th>Nature of evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAL 1</td>
<td>Functionally tested</td>
</tr>
<tr>
<td>EAL 2</td>
<td>Structurally tested</td>
</tr>
<tr>
<td>EAL 3</td>
<td>Methodically tested and checked</td>
</tr>
<tr>
<td>EAL 4</td>
<td>Methodically designed, tested and reviewed</td>
</tr>
<tr>
<td>EAL 5</td>
<td>Semi-formally designed and tested</td>
</tr>
<tr>
<td>EAL 6</td>
<td>Semi-formally verified design and tested</td>
</tr>
<tr>
<td>EAL 7</td>
<td>Formally verified design and tested</td>
</tr>
</tbody>
</table>

Table 3: Evaluation assurance levels
4.2.4 Discussion

Levels are used widely within safety and security standards in order to make precise statements about safety and security properties. However, it is important to distinguish between the required level of safety or security and the required degree of confidence in that level of safety or security.

Thus, it is helpful to consider two kinds of level: requirement levels, which express a safety or security requirement, and assurance levels, which express the degree of confidence required in the assessment of a system.

The safety integrity levels defined in IEC 61508 are requirement levels. They are defined in terms of a target failure measure for a safety function that is based on the probability or frequency of dangerous failures. The concept of a security level in IEC 62443 is similar but based on a more subjective and qualitative ranking of security requirements that models the capabilities of the attacker.

In contrast, the evaluation assurance level in IEC 15408 is an assurance level rather than an integrity level. It is actually a requirement on the assurance process rather than the system itself.

Underpinning the idea of a level is an ordinal scale on which safety and security or confidence can be measured – higher levels imply more safety and security or more confidence. This measurement scale can be either quantitative or qualitative. For example, the safety integrity levels of IEC 61508 are based on a target failure measure that is defined as a probability or frequency. In contrast, the security levels of IEC 62443 and the evaluation assurance levels of IEC 15408 are based on a qualitative measure of effort, either the effort expected from the attacker or the effort required from the assessor.

There is clearly a link between risk and level – indeed, risk is also a value that can be measured on an ordinal scale. Safety and security levels arise from the identification of measures that are needed to reduce the risk to a tolerable level. Assurance levels arise from the degree of confidence that is required in the effectiveness of those measures, which is perhaps commensurate with the magnitude of the risk reduction that is being claimed.

Standards often make a link between requirement levels and the methods used to achieve those levels, but this link requires some justification. For example, IEC 61508 specifies the methods that should be used in order to achieve a required level of safety integrity. However, there is no justification for the claim that the application of a particular method will achieve the required level. The standard is simply codifying what is believed to be necessary. This is less of an issue for assurance levels, where it is more defensible to make a link between assurance methods and confidence but similar issues of justification and generality apply.

4.3 Security Controls

Broadly speaking, security controls are techniques and measures that are used to address security requirements and reduce the risk of a security breach to an acceptable level. There is no consensus about the precise definition of security control, and different standards choose to emphasis different aspects of this broad characterisation.

For example, the NIA glossary [33] uses the following definition of security controls:
**Security controls**: The management, operational, and technical controls (i.e., safeguards or countermeasures) prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information.

The NIST definition of a security control [34] is similar but differs in two significant ways:

**Security control**: A safeguard or countermeasure prescribed for an information system or an organization designed to protect the confidentiality, integrity, and availability of its information and to meet a set of defined security requirements.

Firstly, NIST do not distinguish between “management, operational and technical” controls, and secondly, NIST make an explicit link between a security control and “a set of defined security requirements”.

In contrast, the ISO 27000 definition of control [29] is based on the concept of managing risk and is rather different in style:

**Control**: means of managing risk, including policies, procedures, guidelines, practices or organizational structures, which can be administrative, technical, management, or legal in nature.

*NOTE Control is also used as a synonym for safeguard or countermeasure.*

The ISA 99 standard [32] prefers to use the term “countermeasure” to avoid confusion with “control” as in “process control”:

**Countermeasure**: action, device, procedure, or technique that reduces a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken.

This definition focuses on reducing the impact of threats, vulnerabilities and attacks, and is essentially about risk management, although the term “risk” is not used explicitly.

The Common Criteria [31] refer to security functional components rather than controls, but the underlying concept appears to be similar:

“Security functional components express security requirements intended to counter threats in the assumed operational environment of the target of evaluation and/or cover any identified organizational security policies and assumptions.”

In particular, countermeasures are considered to be security objectives for the target of evaluation and are expressed as security functional requirements, formulated in a standardized language defined using security functional components.
5 SESAMO GENERIC METHODOLOGY

5.1 PRINCIPLES

5.1.1 Risk-based

Safety and security are both properties of the overall system rather than any given sub-component. The top-level requirement is to maintain the safety or security of the plant, aircraft or other system. Thus, the construction of such systems focuses on minimizing and controlling risks. This is achieved by adopting a risk management approach, which aims to reduce the safety or security risk associated with the system to a tolerable level.

Risk management starts with a process of risk analysis that identifies the potential hazards or threats to the safety or security of the system and then analyses the consequence of an accident or breach of security. It is then necessary to decide how to deal with each risk, which is a process known as risk treatment. Possible strategies include risk avoidance, risk reduction, risk sharing, risk transfer, or risk acceptance. Some of these strategies involve making changes to the design or operation of the system. In particular, risk avoidance attempts to remove the underlying cause of the risk (the hazard or threat) by changing some aspect of the system’s design or operation. Similar, risk reduction involves introducing mitigations or controls to manage the risk. These choices are recorded as additional safety or security requirements for the system, but since the system has now been modified, it is necessary to repeat the whole process of risk analysis and risk treatment until all risks have been reduced to an acceptable level. Moreover, in order to ensure that the system remains safe or secure, it is necessary to continually monitor the effectiveness of the risk management strategy and repeat the risk analysis to identify changes in risks.

The safety case or security case for the overall system makes a claim about the effectiveness of the risk management process. Targets are set for the tolerable frequency and severity of safety or security violations, and the top-level case argues that the implemented safety and security features ensure that the frequency of such violations is within limits. There is also a requirement to show the risk is as low as reasonably practicable (ALARP), which implies that further risk reductions should only be implemented if the costs do not outweigh the gains.

5.1.2 Model-based

Model-based systems engineering (MBSE) is the formalized application of modelling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases[47].

A model is an approximation, representation, or idealization of selected aspects of the structure, behaviour, operation, or other characteristics of a real-world process, concept, or system[48], i.e. an abstraction.

The use of models as an abstraction of a system is as old as engineering itself. Models play an important role in engineering by providing different involved stakeholders with a means to communicate, collaborate and focus on a simplified representation of a system. Although a model is an imperfect representation of the reality, it provides a powerful tool for managing the complexity of a system: its strength relies mainly on its ability to abstract from irrelevant aspects and concentrate only on the meaningful ones. Different views of a system model can be used to address the different aspects of a system (e.g. structural model, functional model, performance model).
In the following we outline an overview of the main benefits of MBSE:

- Shared understanding of system requirements and design
  - Validation of requirements
  - Common basis for analysis and design
  - Facilitated identification of risks
- Assistance in managing complex system development
  - Separation of concerns via multiple views of integrated model
  - Supports traceability through hierarchical system models
  - Facilitates impact analysis of requirements and design changes
  - Supports incremental development & evolutionary acquisition
- Improved design quality
  - Reduced errors and ambiguity
  - More complete representation
- Supports early and on-going verification & validation to reduce risk
- Provides value through life cycle (e.g. training)
- Enhances knowledge capture

When dealing with software systems, the principle of model-based engineering can be extended in a very interesting way by exploiting the fact that the model and the “real thing” are both software artefacts, leading to the concept of “Model Driven Engineering” (MDE) [43], [20] briefly illustrated hereafter.

In the quest for increased quality and productivity, Model Driven Engineering 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) the use of automated transformations to progressively turn the user model into a software product ready for final compilation, binding and deployment.

The possibility to generate a software product through automated model transformation is a unique opportunity that arises thanks to the fact that software models are software themselves.
5.1.3 Separation of concerns

The “Separation of Concerns” term was probably first coined by Dijkstra, referring to intelligent thinking, in his 1974 paper "On the role of scientific thought" [44]. The general idea is that of focusing one's attention upon some aspect without ignoring the other aspects, but acknowledging the fact that from this aspect's point of view, the others are irrelevant. In Dijkstra’s words it is being one- and multiple-track minded simultaneously: it is important to maintain a holistic vision of the system, while realizing that it must also be considered under different perspectives.

In software engineering the principle of separation of concerns is an important guideline for writing programs that are understandable and easy to maintain: it is strictly related to the concepts of modularity and encapsulation and leads to the development of programs that are structured into modular parts, each of which can be seen as a black box with its well-defined interfaces. Each module can be developed independently without any knowledge of the internals of the other modules: the only information needed is exposed through the interfaces.

The concept of separation of concerns is a fundamental best practice for the successful development of complex systems, in which developers “divide and conquer” the problem by focusing, separately and yet consistently, on key aspects such as functionality, real-time behaviour, deployment – and even other stakeholder concerns such as economic costs – rather than by trying to capture all aspects simultaneously in one exceedingly complex big picture.

Separation of concerns is well supported by the concept of design views, currently addressed by the ISO/IEC 42010 standard [23] that provides guidance for structuring and organizing architectural descriptions according to different views. Views should be defined to conform to the different perspectives taken by the system stakeholders and enable them to cover their own respective concerns. The concept of view was also used in the initial definition of the UML, derived from the “4+1” model (static structural, interaction, activity, and state viewpoints, with a use case viewpoint tying them together) of Kruchten [21]. The viewpoints are partially interdependent; for example, entities used in the interaction viewpoint must also be in the static structural viewpoint.

5.1.4 Multi-view

In model-based systems engineering the separation of concerns principle may be achieved using two distinct approaches:

1. designing the system using distinct models, one for each concern;

2. designing the system using a single model with dedicated views, which are specialized projections of the system in specific dimensions of interest. This can be referred to as the “multi-view” approach.

The former approach is not trivial from the theoretical point of view. Distinct models are used to address different concerns. The models may or may not use the same specification language. The overall description of the system requires the weaving of the models. The major difficulty in this activity lies in assuring that the semantics of the models (hence of their metamodel) do not overlap so that models address non-overlapping concerns.

With the multi-view approach separation of concerns is achieved by providing distinct system views to the designer, each one for a different relevant aspect. This corresponds to the “Conceptual
Frameworks” systems engineering approach, based on the identification of different “viewpoints”, where a viewpoint identifies a set of concerns, representations and modelling techniques that are specific to a stakeholder. A “view” is the result of applying a viewpoint to a particular system. In order to provide a valuable support for modelling and development, it is necessary to capture the system at different levels of abstraction and at various viewpoints, thus promoting a “multi-view” approach: a single model of the system, with different views addressing different concerns. With this approach the system as a whole is specified in a single model that conforms to a single metamodel (hence the various views use the same language).

The most difficult part of the multi-view approach is to check (in an a-posteriori fashion) or better to guarantee a-priori the consistency of views, so that the different aspects of the system can be consistently aggregated to form a meaningful and coherent description of the system.

The presented approaches are just the two extreme paradigms. An adopted approach could also stand in between them (using a single model and a set of views for a set of design aspects and some separate models to address other concerns).

5.1.5 Component-based

Component-based Software Engineering (CBSE) [24] is a software methodology that emerged in the late '90s. CBSE has the potential of fostering systematic software reuse and thus is a good candidate to cope with the ever growing pressure for shorter time-to-market and decreased cost of software production.

A component model [24], [25] is a framework that is used to design and implement a system in accordance with CBSE. The system is built as a composition of components, which are reusable software units. In particular, the long-term goal of CBSE is the rapid creation of systems, designed as an assembly of reused components.

Two important concepts in the CBSE approach are the principles of composability and compositionality. For the principle of composability properties of components are retained after the component is assembled with other components and deployed to the target system. For the principle of compositionality, instead, it is possible to derive the properties of the overall system by applying some system-specific function to the properties of the components the system is comprised of.

An example of the use of composability and compositionality is the adoption of a partitioned approach. A system is partitioned into several logical partitions; each single partition is segregated from the rest of the system by means of some form of temporal and space isolation. The properties verified on partitions in isolation shall not be violated when all partitions are assembled to compose the final complete system, thus attaining composability for those properties. Additionally, it shall be possible to calculate system-wide properties (for example, the end-to-end delay of communications between partitions) compositionally, that is, only using the properties of the partition expressed in the partition interface, without knowing the internal details and implementation of the logical partition.

Although not always presented explicitly in every CBSE approach, we can recognize three different entities that are essential elements of a component model: components, connectors and containers.
5.1.5.1 Component

Components are reusable software units that encapsulate a distinct functional part of the system. Components can be accessed only through their exposed interface, which comprises: (i) a set of provided services, which are the services offered to the system, and (ii) a set of required services, which are the services that the component requires from other components or the environment in order to discharge its own obligations towards its service users.

The interface represents a contract of the component with the system: as long as all the needs of the component are satisfied, then the component is able to operate successfully and to deliver its services as expected. The role of the component interface is then fundamental, since it contains all the elements that regulate the allowable bindings of distinct components. The properties for which component composability and compositionality shall apply thus have to be explicitly sanctioned in the interface.

![Component Assembly](image)

Figure 16 Component Assembly

5.1.5.2 Connector

The connection or binding between two distinct components entails a notion of interaction between the two parties. Typically, two scenarios may occur: (i) either the interaction between the components is left implicit; or (ii) the interaction is explicitly modelled as a connector [6].

The first scenario makes it harder to obtain reusable components, since the notion of interaction between components is buried inside and implicitly bound to the component itself.

In practice, the majority of component models offer a set of built-in connectors, which represent the complete range of supported interactions between components (such as procedure call, remote procedure call, I/O operation on a file, etc.), but there can be some cases in which it is possible for the user to define their own connectors (typically as instantiation of a generic connector type, or as a composition of atomic connectors).

The connector may also be used to specify properties of the connection (for instance the level of required fault tolerance, i.e. best effort, at most once, exactly once...).

The use of an explicit notion of connector is interesting because it clearly separates the functional/computational part of the problem (which resides in the component) from the interaction part of the problem [42], thus easing the reuse of the former independently of the latter. This separation in fact permits to cleanly disentangle the component from the other parties in the...
communication. Hence the component is left unmodified irrespective of the actual binding of the connector and thus the component is independent from the type and number of senders/receivers at the opposite side of the communication.

5.1.5.3 Container

In component models the container is the software entity responsible for the realization of non-functional properties.

It can be considered as a wrapper placed around the component, which is employed to:

1. realize the extra-functional properties declared for the component;
2. provide the platform/middleware services required by the component (realized by the container itself and/or the underlying execution platform);
3. realize the interactions with other components, possibly with an implementation of the notion of connector;
4. interface the components with the system in general and the execution platform in particular.

![Figure 17 Component and Container](image)

The fundamental role of the container is to decouple the functional concerns of the component, from the extra-functional concerns that the realization of the component is required to exhibit at run-time.

In particular, the container layer is in charge of the realization of tasking, timing behaviour, fault containment and fault tolerance, security, configuration management, etc...

![Figure 18 Connector as explicit interaction constructs between components/containers](image)
5.1.6 Incremental and iterative

The SESAMO methodology should be incremental and iterative. In particular, design and evaluation should be interleaved, so that evidence from design informs evaluation and the results of evaluation inform design. This is illustrated in Figure 19, which shows the interaction between a design process based on threat/risk assessment, and an evaluation process, based on security assurance.

Figure 19 – Integration between threat/risk assessment and security assurance

5.1.7 Assurance and evidence

Assurance is about gaining confidence. Depending on the context, this could be confidence in the system as a product, or confidence in the operation of the system in a given environment. In the context of safety and security, assurance is our confidence that the safety or security risks associated with the system have been reduced to a tolerable level.

Assurance requires evidence, which can take many forms and is generated during the product development or operation of the system. For example, evidence from the development of a system would include the usual engineering documentation such as requirements specification, analysis and design documents, test specifications, implementation artefacts such as source code, and outputs from any verification or validation activities. Evidence from the operation of the system might include configuration data, change management history, performance statistics, incident logs and reports.

The degree of confidence that can be obtained in the assurance of a system depends not only on the strength of the claim that the evidence supports (or rebuts) but also on the extent to which the evidence itself is trustworthy. Thus, there needs to be confidence in the process for gathering, recording and maintaining evidence. Moreover, there needs to be some scientific basis for the techniques that are used to analyse the evidence and the conclusions that are drawn from this analysis.
5.1.8 Decision support

A structured safety case provides a method of making an informed decision about the trade-offs between safety and security. The primary goal is for the system to be safe, so we will always err on the side of safety rather than security. However, a system that is not secure cannot be safe. Therefore, we will construct a security-informed safety case that makes explicit claims about the safety and the security of the system, supported by appropriate arguments and evidence. This will enable us to make decisions about trade-offs and conflicts between safety and security in a principled way, and provide a justification for accepting that the risks associated with the system are as low as reasonably practicable.

5.2 FORMALISM

This section gives an overview of two modelling techniques that can be used to formalize a process model, the Unified Modeling Language (UML) and the Business Process Modeling Notation (BPMN). It also discusses which technique will be used later on in SESAMO.

5.2.1 Unified Modeling Language (UML)

UML is a relatively open standard, controlled by the Object Management Group (OMG). The OMG specification states:

“The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.” [1]

Thus it comprises techniques like entity relationship diagrams, business work flows, object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies [2]. It offers a standard way to visualize a system's architectural blueprints, including elements such as:

- activities
- actors
- business processes
- database schemas
- (logical) components
- programming language statements
- reusable software components.[3]

UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language [4]. UML aims to be a standard modeling language which can model concurrent and distributed systems.

Especially for business process modelling UML provides a set of useful diagrams [5]:

- **UML Use Case Diagrams** can describe business segments. The participants in the use case can represent the roles of the partners that interact with the process.
• **UML Component Diagrams** can be used to refine the dependencies described in the use case diagrams. In doing so, the component symbol is used for depicting a Web service. Both the participants of the business process and the business process itself can be modelled as Web services.

• **UML Communication Diagrams** can be used for modelling the business and market layers. They can provide an overview of a collaboration of business process participants and their relationships.

• **UML Activity Diagrams** are suitable for modelling protocols and business processes for Web services. Besides control-flow elements (decision, fork, join, etc.), the Activity Diagrams contain the necessary basic activities for interacting with the partner services. Activity Diagrams can also define transactional processes, which refine collaborations defined by the UML Communication Diagram. Activity Diagrams are the most detailed form of process modelling within UML.

• **UML Class Diagrams** are suitable for defining the structure of data handled by the business processes: attributes of the objects and content of the messages, passed between the objects. Likewise, they can provide details of the different port types by defining their operations and the parameters involved.

• **UML State Machine Diagrams** can define states of objects that are used as pre- and post-conditions of transactions.

• **UML Interaction Diagrams** show an overview of interactions for a complex use case or a system. They mix an Activity Diagram and a Sequence Diagram, focusing on events instead of actions compared to an Activity Diagram.

5.2.2 **Business Process Modeling Notation (BPMN)**

The Business Process Modeling Notation (BPMN) was developed by the Business Process Management Initiative (BPMI), a non-profit consortium consisting of such companies like for instance Intalio, Adobe, PeopleSoft, SAP. The goal of the BPMI was to create an integrated open standard for the modelling, the implementation and the execution of business processes. Thereby the commercial aspects as well as the IT aspects of the business processes are considered.

BPMN allows the graphical modelling of business processes from a management point of view by means of a standardised and comprehensive notation. The notation claims on the one hand side that process models are easy to read and quickly comprehensible for business analysts independent on the used (BPMN-compliant) tool. On the other side it allows for the modelling of very complex business processes including Business-To-Business process because of its capabilities.

The business process model developed by means of BPMN shall be implemented in standardised, XML-based, executable business process languages, like Business Process Modelling Language (BPML) or Business Process Execution Language for Web Services (BPEL4WS) without any further transformations of the model. The business process execution languages are controlled and executed by so-called Business Process Management Systems (BPMS). This approach developed by the BPMI is therefore a bridge between business process modelling, business process implementation and business process execution.

Business processes in BPMN are modelled and presented in so-called Business Process Diagrams (BPD) by means of graphical icons (Shapes). In order to define a process work flow the triggering events are introduced, the subsequent steps are modelled and the end of the process is denoted by an end event. Branches and joins of the process flow are pictured as diamond-shaped gateways, which
control the process flow. The performed steps for the execution of a business process are called Activities. BPMN knows three activity types: Process, Sub-Process and Task.

According to BPMN the term Process follows a generic approach and is defined as a set of Activities which are executed within and/or between organizations. Hereby the definition of what can be described as a process is not defined and its definition is open to the modeller. A Sub-Process is an Activity in which Activities are described. A Task is an atomic Activity, whose function is not described in more detail.

![Diagram](image)

**Figure 20 Example for a business process modelled in BPMN**

Of course, also the responsibility for Activities can be defined by representing a participant of a process, e.g. an organisation, as a so-called Pool and its acting units as Lanes in the Pool. The acting unit might be for instance a function, a role or a IT-system. The Lanes can be nested.

The modelling of the flow of actions does not differ much from other notations. The process is depicted as a sequence of Activities whereby the Activities are connected by a Sequence Flow which is presented as a bold line. The Sequence Flows cannot cross the Pool borders. The Pools have to interact via Message Flows which are shown as dashed lines.

The flow of actions is often characterized by branches and joins. These branches and joins are described by so-called Gateways, which are represented as diamond-shaped icons.

### 5.2.3 Relevance for SESAMO

In the context of SESAMO UML is suitable and sufficient in order to describe a generic process and its corresponding activities. The advantages are:

- UML provides a graphical notation for business process modelling mostly by Activity Diagrams.
- UML supports and encourages the modelling of parallel behaviour.
- UML is public and it is available for download from the OMG web site.
- There are a lot of modelling tools, including open source tools.
The BPMN is targeting business people and its Activity Diagram is more technically oriented. It does not provide any additional advantage for SESAMO.
6  TOWARDS A RISK-BASED METHODOLOGY

6.1  GENERIC PROCESS DEFINITION

The SESAMO generic process will be based on the principles of risk management outlined in ISO 27005. Figure 21 provides a high level view of such a process:

![Figure 21 – Risk management process](image)

Having established the system context, the first step is to perform a risk assessment, which involves identifying, analyzing and evaluating the risks associated with the system. This is followed by a process of a risk treatment, in which decisions are taken about how best to deal with each risk.

Both risk assessment and risk treatment can be iterative, the goal being to reduce the residual risks to an acceptable level, as shown in Figure 22, which shows the risk treatment process in more detail:

![Figure 22 – Risk treatment](image)
There are four basic approaches to risk treatment:

- **Risk modification**: manage the level of risk by introducing appropriate controls
- **Risk retention**: make an objective and informed decision to accept the risk
- **Risk avoidance**: modify the system to avoid the condition that gives rise to the risk
- **Risk sharing**: pass some or all of the risk onto some other system

These choices need to be recorded as part of the safety case for the system.

### 6.2 SECURITY-INFORMED RISK ASSESSMENT

The purpose of a case is to demonstrate that the risks associated with a system and well understood and reduced to ALARP. Thus, one approach to developing a security-informed safety case is to perform a security-informed risk assessment.

Our method for performing a security-informed risk assessment is based on our experience of using such techniques to analyze large-scale critical infrastructure systems that need to be both safe and secure. The process consists of eight iterative steps to perform the risk assessment (see Table 4).

<table>
<thead>
<tr>
<th>Step</th>
<th>Brief description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1 – Establish system context and scope of assessment</td>
<td>Describe the system to be assessed and its relationship with other systems and the environment. Identify the services provided by system and the system assets. Agree the scope of and motivation for the assessment and identify the stakeholders and their communication needs. Identify any existing analyses, e.g. safety cases.</td>
</tr>
<tr>
<td>Step 2 – Identify potential threats</td>
<td>Define the threat sources and identify potential threat scenarios.</td>
</tr>
<tr>
<td>Step 3 – Refine and focus system models</td>
<td>Refine and focus system models in the light of the threat scenarios to ensure that they are at the right level of detail for an effective risk analysis.</td>
</tr>
<tr>
<td>Step 4 – Preliminary risk analysis</td>
<td>Undertake architecture based risk analysis, identifying consequences and relevant vulnerabilities and causes together with any intrinsic mitigations and controls. Consider doubts and uncertainties, data and evidence needs.</td>
</tr>
<tr>
<td>Step 5 – Identify specific attack scenarios</td>
<td>Refine preliminary risk analysis to identify specific attack scenarios. Focus on large consequence events and differences with respect to existing system.</td>
</tr>
<tr>
<td>Step 6 – Focused risk analysis</td>
<td>Match threat sources to attack scenarios and prioritize possible consequences according to the level of risk. As with Step 6 the focus is on large consequence events and differences with respect to existing system.</td>
</tr>
<tr>
<td>Step 7 – Finalize risk assessment</td>
<td>Finalize risk assessment by reviewing implications and options arising from focused risk analysis. Review defense in depth and undertake sensitivity and uncertainty analysis. Consider whether design-basis threats are appropriate. Identify additional mitigations and controls.</td>
</tr>
<tr>
<td>Step 8 – Report results</td>
<td>Report the results of the risk assessment to stakeholders at the appropriate level of detail.</td>
</tr>
</tbody>
</table>

**Table 4. Risk assessment process**

In parallel with this process, the security/risk case is developed progressively throughout the risk analysis process to synthesize risk claims, arguments and evidence. The details of how security
risks are mapped onto claims are very dependent on the specific case. Also, the case can be developed and issued at different levels of detail, depending on the intended stakeholder audience.

6.3 SECURITY-INFORMED SAFETY CASE METHODOLOGY

In principle, all safety cases should be “security-informed” because a system that is not secure cannot be safe. However, many safety cases assume that the system is operated in a benign environment, and therefore ignore security threats. This is not an unreasonable position to take, providing the safety case makes this assumption explicit, because the safety of the system can only be guaranteed if the environment in which it is operated remains secure. Thus, security must be considered as part of the safety case and any assumptions about the security of the environment must be properly documented as part of the safety case.

Our experience has shown that a significant portion of a security-informed safety case will need to address security explicitly. In some instances this will lead to substantial changes to the design, the implementation process and the justification. For example, the following areas are particularly significant from a security perspective and need more scrutiny in a security-informed safety case:

- Supply chain integrity.
- Malicious events post deployment, that will also change in nature and scope as the threat environment changes.
- Weakening of security controls as the capability of the attacker and technology changes. This may have major impact on proposed lifetime of installed equipment and design for refurbishment and change.
- Design changes to address user interactions, training, configuration, and vulnerabilities. This might lead to additional functional requirements that implement security controls.
- Possible exploitation of the device/service to attack itself or others.

In order to address these additional security risks within a case, it is necessary to find a way of combining safety and security risk assessment. With this in mind, we are developing an adapted process that can be used where safety cases and risk assessments already exist but need augmenting to make them security-informed. Thus, our approach is different from other work in avionics, for example, where the idea is to develop an integrated approach from scratch.

We start by expressing the safety justification for a system in terms of a structured safety case, with explicit claims, arguments, and evidence. This enables us to identify any gaps or omissions in the safety case or implicit assumptions that need to be made explicit.

We can then investigate the impact that security might have on a case by considering the three aspects of Claims-Arguments-Evidence, and deciding whether we need to

- Change the (top level) claims, if any
- Augment the arguments
- Change how we deal with evidence

The next step is to identify security controls that can be used to provide the evidence or argument that is necessary to satisfy a particular security claim. However, the introduction of such controls will have an impact on the system, either during development, operation or maintenance. Thus, it is
necessary to consider the impact of deploying security controls on the system life cycle and iterate the process.

In terms of methodology, the steps are:

- Express safety case about system behaviour in terms of Claims-Arguments-Evidence
- Review how the claims might be impacted by security
- Review security controls to see if these can be used to provide an argument and evidence for satisfying the claim
- Review architecture and implementation impact of deploying controls and iterate the process

Table 5 provides a brief description of each of these steps.

<table>
<thead>
<tr>
<th>Step</th>
<th>Brief description</th>
</tr>
</thead>
</table>
| Step 1 – Develop structured safety case | • Express the safety justification as a structured safety case  
• Make claims, arguments, and evidence explicit  
• Identify any gaps or omissions  
• Highlight any assumptions that need to clarified |
| Step 2 – Develop security-informed safety case | • Identify security context and specific attack scenarios  
• Review impact of security on existing claims, arguments, evidence  
• Develop additional claims and arguments, identify required evidence, and revise structured safety case as necessary |
| Step 3 – Apply appropriate security controls | • Identify security controls that can be used to satisfy security claims, arguments, evidence  
• Review impact of deploying controls on system life cycle |

Table 5: Security-informed safety case methodology
7 REFERENCES


[18] “D2.3 – Multi-concern Component Methodology (MCM) and Toolset” Version 1.1, 21 January 2011


[34] R. Kissel (Ed.), Glossary of Key Information Security Terms, NIST IR 7298, Revision 1, February 2011.


