Project: COMPASS
Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

Initial Report on Guidelines for Architectural Level SoS Modelling

Document Number: D21.2
Date: March 2013
Public Document

http://www.compass-research.eu
Contributors:
Finn Overgaard Hansen (FOH), AU
Stefan Hallerstede (SHA), AU
Simon Perry (SAP), ATEGO
Jon Holt (JDH), ATEGO
Klaus Kristensen (KRT), Bang-Olufsen

Editor:
Finn Overgaard Hansen, AU

Reviewers:
Reviewer 1, UNUT: Claire Ingram
Reviewer 2, UFPE: Juliano Iyoda
Reviewer 3; YU: Alvaro Miyazawa
Reviewer 4, INSIEL: Margherita Forcolin
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>16.10.2012</td>
<td>F.O.Hansen</td>
<td>Initial draft outline version</td>
</tr>
<tr>
<td>0.11</td>
<td>10.11.2012</td>
<td>F.O.Hansen</td>
<td>Update of outline based on received feedback</td>
</tr>
<tr>
<td>0.12</td>
<td>18.12.2012</td>
<td>F.O.Hansen</td>
<td>Update of contents for all sections</td>
</tr>
<tr>
<td>0.13</td>
<td>13.02.2013</td>
<td>F.O.Hansen</td>
<td>Update of section 3.</td>
</tr>
<tr>
<td>0.14</td>
<td>21.02.2013</td>
<td>S.Hallerstede</td>
<td>Added section 3.1</td>
</tr>
<tr>
<td>0.15</td>
<td>25.02.2013</td>
<td>S. A. Perry</td>
<td>Added Section 2.1 and some references</td>
</tr>
<tr>
<td>0.16</td>
<td>26.02.2013</td>
<td>K.Kristensen</td>
<td>Added Section 4.1</td>
</tr>
<tr>
<td>0.17</td>
<td>26.02.2013</td>
<td>F.O.Hansen</td>
<td>Additions to section 3.</td>
</tr>
<tr>
<td>0.18</td>
<td>4.03.2013</td>
<td>F.O.Hansen</td>
<td>Further input to section 3.</td>
</tr>
<tr>
<td>0.19</td>
<td>4.03.2013</td>
<td>K.Kristensen</td>
<td>Additions to section 4.1</td>
</tr>
<tr>
<td>0.20</td>
<td>7.03.2013</td>
<td>F.O.Hansen</td>
<td>Additions to section 1.</td>
</tr>
<tr>
<td>0.21</td>
<td>8.03.2013</td>
<td>S.Hallerstede</td>
<td>Additions to section 3.3.</td>
</tr>
<tr>
<td>0.22</td>
<td>10.03.2013</td>
<td>F.O.Hansen</td>
<td>Additions to section 3.1 and 3.2.</td>
</tr>
<tr>
<td>0.23</td>
<td>12.03.2013</td>
<td>F.O.Hansen</td>
<td>Minor corrections</td>
</tr>
<tr>
<td>0.24</td>
<td>13.03.2013</td>
<td>F.O.Hansen</td>
<td>Added summary section</td>
</tr>
<tr>
<td>0.25</td>
<td>21.03.2013</td>
<td>F.O.Hansen</td>
<td>Updated based on reviews</td>
</tr>
<tr>
<td>1.00</td>
<td>27.03.2013</td>
<td>F.O.Hansen</td>
<td>Updated to EU-delivery doc.</td>
</tr>
</tbody>
</table>
Abstract

This deliverable contains an initial report on guidelines to be used in architectural level System of Systems (SoS) modelling for defining an appropriate architecture. The guidelines relate to different SoS and Constituent System (CS) types, a set of SoS characteristics and to different SoS development contexts.

The report starts by defining an Architectural Framework (AF), called the COMPASS Architectural Framework Framework (CAFF). CAFF can be used for defining concrete architectural frameworks and will be employed in the next COMPASS phase to develop project specific AF.

The architectural guidelines have been being tested initially against the COMPASS Bang & Olufsen case study as a reality check of the usability. The guidelines will be tested, adjusted and refined during the next phase of the COMPASS project.

The final version of this report will be issued as D21.5 Final report on Guidelines for Architectural level SoS modelling.
# Table of Contents

1. **INTRODUCTION** ................................................................................................................. 9  

   1.1. **SCOPE** ......................................................................................................................... 9 
   1.2. **CONTEXT** ..................................................................................................................... 11  

2. **GUIDELINES FOR ARCHITECTURAL FRAMEWORKS** ......................................................... 12  

   2.1. **EXISTING ARCHITECTURAL FRAMEWORKS** .............................................................. 12 
   2.2. **OVERVIEW OF AF PROCESS APPROACH** .................................................................. 12 
   2.3. **OVERVIEW OF ARCHITECTURAL FRAMEWORKS** .................................................... 13 
   2.4. **THE COMPASS ARCHITECTURAL FRAMEWORK FRAMEWORK** ............................ 13 
   2.4.1. **The Needs for the CAFF - The AF Context View** .................................................. 13 
   2.4.2. **The Ontology for CAFF - The Ontology Definition View** ..................................... 15 
   2.4.3. **The Viewpoints and Perspectives for CAFF - The Viewpoint Relationships View** 19 
   2.4.4. **The Viewpoints of CAFF – Viewpoint Definition** .................................................. 20 
   2.4.5. **The Rules Constraining the CAFF – The Rule Definition View** .............................. 20  

3. **GUIDELINES FOR ARCHITECTURAL SOS MODELLING** .................................................... 22  

   3.1. **Determine Characteristics of the SoS and CS** ............................................................... 22 
   3.1.1. **SoS Categories** ......................................................................................................... 22 
   3.1.2. **CS Categories** ......................................................................................................... 23 
   3.1.3. **Determine SoS Characteristics** .................................................................................. 26 
   3.1.4. **Mapping Characteristics to Categories** ..................................................................... 28 
   3.2. **Determine SoS Development Context and Lifecycle** ................................................... 29 
   3.2.1. **SoS and CS Lifecycles** ............................................................................................... 29 
   3.2.2. **SoS Top-down Development Context** ...................................................................... 30 
   3.2.3. **SoS Bottom-up Development Context** ................................................................... 32 
   3.2.4. **SoS Middle-in Development Context** ...................................................................... 32 
   3.3. **SoS Requirements Analysis** ........................................................................................ 35 
   3.3.1. **Determine the SoS Context Diagram** ....................................................................... 35 
   3.3.2. **Determine the Architecture-centric SoS Use Cases** ................................................. 35 
   3.3.3. **Prioritize the Non-functional SoS Requirements** ...................................................... 36 
   3.3.4. **Tracing SoS Requirements into the SoS Architecture** .............................................. 36 
   3.4. **Define SoS Architecture** .............................................................................................. 36 
   3.4.1. **Define Communication Paradigm** ............................................................................ 36 
   3.4.2. **Define Architectural Styles and Patterns** .................................................................. 37 
   3.5. **Identify and Specify CS** ............................................................................................... 37 
   3.6. **Validate SoS Architectures against Requirements** ....................................................... 38 
   3.7. **Define SoS-CS Structure and Behaviour** .................................................................... 38 
   3.8. **Define Interfaces and Contracts for CS** ..................................................................... 39 
   3.9. **Document the selected SoS Architecture with SysML** ................................................ 40  

4. **RELEVANCE IN RELATION TO COMPASS CASE STUDIES** ............................................... 41  

   4.1. **Relevance in relation to the B&O Case Study** ............................................................... 41 
   4.1.1. **Determine B&O SoS Type** ....................................................................................... 43 
   4.1.2. **Determine B&O SoS Development Context** ............................................................. 46  

5. **SUMMARY** ............................................................................................................................. 50  

6. **REFERENCES** ........................................................................................................................ 50  

APPENDIX A **VIEWPOINTS OF CAFF: VIEWPOINT DEFINITIONS** .............................................. 53  

   A.1. **The Viewpoints of CAFF – Viewpoint Definition** ......................................................... 53 
   A.2. **AF Context Viewpoint (AFCV)** ................................................................................... 53 
   A.3. **Ontology Definition Viewpoint (ODV)** ...................................................................... 55 
   A.4. **Viewpoint Relationships Viewpoint (VRV)** ................................................................. 57 
   A.5. **Viewpoint Context Viewpoint (VCV)** ........................................................................ 60 
   A.6. **Viewpoint Definition Viewpoint (VDV)** ..................................................................... 62 
   A.7. **Rules Definition Viewpoint (RDV)** ............................................................................ 64
APPENDIX B  PROCESSES FOR AF DEFINITION AND CONSTRUCTION .....................................67

B.1 REQUIREMENTS CONTEXT VIEW ..........................................................................................67
B.2 STAKEHOLDER VIEW ..........................................................................................................68
B.3 PROCESS CONTENT VIEW ..................................................................................................69
B.4 PROCESS BEHAVIOUR VIEWS ..........................................................................................72
B.4.1 PBV – AF DEFINITION PROCESS .................................................................................72
B.4.2 PBV – VIEWPOINT DEFINITION PROCESS ..............................................................73
B.4.3 PBV – ONTOLOGY DEFINITION PROCESS .................................................................74
B.4.4 PBV – CONTEXT PROCESS ..........................................................................................74
B.5 INFORMATION VIEWS .........................................................................................................75
B.5.1 IV – AF DEFINITION PROCESS ....................................................................................75
B.5.2 IV – VIEWPOINT DEFINITION PROCESS .................................................................75
B.5.3 IV – ONTOLOGY DEFINITION PROCESS ......................................................................76
B.5.4 IV – CONTEXT PROCESS ..............................................................................................76
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope of the SoS Architecture Guidelines in relation to the other SoS</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Relations between Architectural Frameworks, Architecture, Processes and</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>AF Context View Showing the Requirements for the CAFF</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>Ontology Definition View for Architectural Frameworks</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Ontology Definition View Expanding on Context</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>Viewpoint Relationships View Showing Viewpoints and Perspectives that make</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>AF Framework</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>SoS consisting of enabling and constituent systems</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Self-contained CS as a Blackbox</td>
<td>24</td>
</tr>
<tr>
<td>10</td>
<td>Dependent CS as a Blackbox</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>Greybox CS</td>
<td>25</td>
</tr>
<tr>
<td>12</td>
<td>Whitebox CS</td>
<td>26</td>
</tr>
<tr>
<td>13</td>
<td>Mapping SoS Characteristics to Categories</td>
<td>28</td>
</tr>
<tr>
<td>14</td>
<td>Three development contexts for SoS development</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>SoS maturity and SoSE Focus points [Lane et. al. 2012]</td>
<td>29</td>
</tr>
<tr>
<td>16</td>
<td>SoS and CS lifecycle maturity states</td>
<td>30</td>
</tr>
<tr>
<td>17</td>
<td>Top-down SoS Development</td>
<td>31</td>
</tr>
<tr>
<td>18</td>
<td>Bottom-up SoS Development</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Middle-in SoS Development</td>
<td>33</td>
</tr>
<tr>
<td>20</td>
<td>SysML SoS Context Diagram</td>
<td>35</td>
</tr>
<tr>
<td>21</td>
<td>SysML Use Case Diagram</td>
<td>35</td>
</tr>
<tr>
<td>22</td>
<td>SoS-CS and CS Use Case diagram</td>
<td>38</td>
</tr>
<tr>
<td>23</td>
<td>SysML Block Definition Diagram for SoS-UC3</td>
<td>39</td>
</tr>
<tr>
<td>24</td>
<td>SysML Sequence Diagram for SoS-UC3</td>
<td>39</td>
</tr>
<tr>
<td>25</td>
<td>Multiple product scenario</td>
<td>41</td>
</tr>
<tr>
<td>26</td>
<td>B&amp;O streaming SoS characteristics</td>
<td>46</td>
</tr>
<tr>
<td>27</td>
<td>B&amp;O streaming architecture style</td>
<td>47</td>
</tr>
<tr>
<td>28</td>
<td>B&amp;O streaming communication style</td>
<td>47</td>
</tr>
<tr>
<td>29</td>
<td>Viewpoint Context View for the AF Context Viewpoint</td>
<td>53</td>
</tr>
<tr>
<td>30</td>
<td>Viewpoint Definition View for the AF Context Viewpoint</td>
<td>55</td>
</tr>
<tr>
<td>31</td>
<td>Viewpoint Context View for the Ontology Definition Viewpoint</td>
<td>56</td>
</tr>
<tr>
<td>32</td>
<td>Viewpoint Definition View for the Ontology Definition Viewpoint</td>
<td>57</td>
</tr>
<tr>
<td>33</td>
<td>Viewpoint Context View for the Viewpoint Relationships Viewpoint</td>
<td>58</td>
</tr>
<tr>
<td>34</td>
<td>Viewpoint Definition View for the Viewpoint Relationships Viewpoint</td>
<td>59</td>
</tr>
<tr>
<td>35</td>
<td>Viewpoint Context View for the Viewpoint Context Viewpoint</td>
<td>60</td>
</tr>
<tr>
<td>36</td>
<td>Viewpoint Definition View for the Viewpoint Context Viewpoint</td>
<td>61</td>
</tr>
<tr>
<td>37</td>
<td>Viewpoint Context View for the Viewpoint Definition View</td>
<td>62</td>
</tr>
<tr>
<td>38</td>
<td>Viewpoint Definition View for the Viewpoint Definition View</td>
<td>63</td>
</tr>
<tr>
<td>39</td>
<td>Viewpoint Context View for the Rules Definition Viewpoint</td>
<td>65</td>
</tr>
<tr>
<td>40</td>
<td>Viewpoint Definition View for the Rules Definition Viewpoint</td>
<td>66</td>
</tr>
</tbody>
</table>
Figure 41 - Requirements context view (RCV) for the architectural framework definition processes .......................................................... 67
Figure 42 - Stakeholder view (SV) for the architectural definition processes ................................................................................. 68
Figure 43 - Process content view (PCV) for the architectural framework definition processes ................................................................................. 69
Figure 44 - PBV for AF Definition Process .......................................................................................................................................... 73
Figure 45 - PBV – Viewpoint Definition Process ........................................................................................................................................ 73
Figure 46 - PBV – Ontology Definition Process ........................................................................................................................................ 74
Figure 47 - PBV – Context Process ................................................................................................................................................. 74
Figure 48 - IV – AF Definition process ............................................................................................................................................. 75
Figure 49 - IV – Viewpoint Definition Process ........................................................................................................................................ 75
Figure 50 - IV – Ontology Definition Process ........................................................................................................................................ 76
Figure 51 - IV – Context Process ...................................................................................................................................................... 76
1. Introduction

This document contains the results of an initial study into architectural level modelling of System of Systems (SoS). The following subsections establish the scope of the document and the context in which it has been produced.

1.1. Scope

This document presents a set of initial guidelines for architectural level SoS modelling.

It focuses on giving concrete guidelines for the architectural SoS modelling based on the architectural styles and patterns surveyed and described in the COMPASS deliverable D22.3 [D22.3 2013] together with other SoS examples found in different application domains.

Figure 1 shows the set of guidelines, which are planned as deliverables of the COMPASS project, where the SoS Engineering Guidelines is the main document describing the systems of systems engineering process with links to the three other planned guideline documents to be produced. The Engineering Guidelines document includes guidelines for establishment of appropriate constituent systems (CS) and guidelines for the SoS acceptance test activity. The guideline documents are presented in the context of the classical V-model for systems and software development, which also can be related to development or realizations of SoS.

Figure 1 indicates that this document, the “Initial report on guidelines for architectural level SoS modelling” will evolve in the final version to be the “SoS Architecture Guidelines”. It is also indicated that the Architecture Guidelines depends on and relates to the “SoS Requirements Engineering Guidelines”, where the overall goals or objectives for a given SoS are defined.
Developing an SoS architecture is an essential SoS Engineering activity which as discussed in sections 2.4.2 and 2.5 of COMPASS deliverable D22.3 “Report on Modelling Patterns for SoS Architectures” [D22.3 2013] can be characterized by:

- **SoS Architectures** are core to systems and SoS engineering; production of architectures is not an optional activity.
- **SoS Architectures** should be produced according to defined **Architectural Viewpoints** codified in an **Architectural Framework** that includes **consistency rules** defined between the various views produced and the information contained in them.
- **SoS Architectures** should be produced to address the concerns of **stakeholders** using relevant architectural viewpoints; it is important to know why a particular viewpoint is being used.
- It is essential that an **SoS architectural design process** is defined and followed. This must cover the definition, analysis, evaluation, documentation and maintenance of an architecture.
- **SoS Architectures** should address both **structure and behaviour**, including communications, functionality and data flow.
- Modelling is essential to the development of **SoS architectures**.

Figure 2 shows how the guidelines relate to the other architectural activities and scopes the content in this document.

Section 2 introduces the COMPASS Architectural Framework Framework (CAFF), which is a meta-framework to be used in the construction of a SoS architectural framework for a given SoS project. To assist in construction of a concrete SoS Architectural Framework a process has been defined and described in Appendix B. This process is planned to be used in the next COMPASS project phase for cre-
ating of project specific AF and will be further described and detailed in the final version of this architecture guidelines document.

Section 3 presents an initial set of guidelines for architectural SoS Modelling, which provides guidelines for the production of a concrete SoS architecture for a given SoS project.

Section 4 describes how the guidelines from Section 3 have been used in one of the COMPASS case studies and works as an initial feasibility check of the guidelines.

This guidelines document repeats definitions presented in other COMPASS documents with the purpose of being easier to read and use.

The primary target group for this document is system and software engineers involved in architecting an SoS.

1.2. Context

The guidelines for architectural SoS modelling presented in this document are to be used and tested in the second phase of the COMPASS project as a set of initial guidelines for the industrial case studies in the COMPASS project.

During the second phase, the COMPASS Architectural Framework will be refined and concrete SoS architectural frameworks will be defined for the case studies. The guidelines in this version will be extended and refined based on the experiences obtained during the second phase of the COMPASS project and will be aligned with the architectural frameworks produced based on CAFF.

This document is planned to be continuously updated with the experiences obtained and further research on the topics during the COMPASS project and is to be released as D2.5 “Final Report on Guidelines for Architectural Level SoS Modelling” for public use, at the end of the COMPASS project.
2. Guidelines for Architectural Frameworks

Architectures and Architectural Frameworks (AFs) are an essential part of systems engineering and SoS engineering as well, where SoS Architectures should be produced according to a set of defined Architectural Viewpoints codified in an Architectural Framework. The following sub-sections introduce existing architectural frameworks, outline an approach to developing an AF and introduce the COMPASS Architectural Framework Framework (CAFF).

2.1. Existing Architectural Frameworks

While there are a number of widely-used architectural frameworks, this does not necessarily mean that they are suitable for all systems engineering and SoS engineering projects; it is important that the correct AF, fit for purpose, be used. For example:

- Defence frameworks, such as MODAF, DoDAF and NAF, are intended to be used in the acquisition of systems.
- TOGAF is intended to provide an approach for developing IT architectural frameworks.
- Zachman is a framework approach for defining IT-based enterprise architectures.

For a discussion of these frameworks see [D22.1 2012].

The use of an AF is a prerequisite for the development of a robust architecture. Before any existing AFs can be assessed for suitability, it is essential that the needs for the architecture, and hence its guiding AF, are understood. If an existing AF is not suitable, then a project will have to define its own AF.

D22.1 concluded that "there is no single technique that stands out as being ideal as an off-the-shelf solution that can be used for the COMPASS project". Therefore there is a need for an approach for defining a bespoke architectural framework.

2.2. Overview of AF Process Approach

A full set of processes for the definition of an architectural framework has been defined in Appendix B, which may be summarised as:

1. Identify needs for the architectural framework. Identify the needs that the architectural framework is to address, placing them into context.
2. Define basic concepts and terms. It is essential that the underlying ontology is defined and understood.
3. Identify an initial set of views. Relate these views together in the form of a framework and classify them into perspectives.
4. Define each of these views in terms of its needs and concepts. Understand the needs that the viewpoint must address, placing them into context.
5. Define the viewpoint using concepts from the ontology so that the identified needs are addressed.
6. Define any rules that constrain the viewpoints and framework. These processes are defined in Appendix B.

2.3. Overview of Architectural Frameworks

An architectural framework has been defined in Section 2.4 that should be used when defining an architectural framework. It presents six views that are used to ensure that:

1. The needs of the AF are understood.
2. The concepts and terms that the AF can model are defined.
3. The views are identified.
4. The needs for each view are understood and are related to the overall needs of the AF.
5. The definition of each view is specified and based on the concepts and terms identified.
6. Any rules that constrain the AF are captured.

The complete definition of the AF framework is given in Section 2.4 below. It should be used along with the AF processes defined in Appendix B and summarised in Section 2.2 above.

2.4. The COMPASS Architectural Framework Framework

The following sections define a framework that is used for defining other architectural frameworks – the COMPASS Architectural Framework Framework (CAFF). It presents a number of viewpoints that are the minimal set needed for the definition of an AF. What is more, the viewpoints are defined using CAFF – that is, the CAFF is defined in terms of itself. This self-referentiality is not uncommon in modelling; for example, the Unified Modelling Language (UML) is defined using the Unified Modelling Language.

The five sub-sections:
1. Define the needs (i.e. requirements) for the CAFF.
2. Define the ontology for the CAFF.
3. Identify the viewpoints that make up the CAFF, along with their grouping into related viewpoints; such a grouping is known as a perspective.
4. Define each CAFF viewpoint.
5. Define some rules that constrain the CAFF.

The sections make use of the Systems Engineering Modelling Language (SysML). For details of the SysML see [SysML2010] and [Holt&Perry2008].

2.4.1. The Needs for the CAFF - The AF Context View

This section defines the needs for the CAFF, presented using a SysML use case diagram. It is an example of a CAFF Architectural Framework Context View.
Figure 3 - AF Context View Showing the Requirements for the CAFF

Figure 3 shows the Architectural Framework Context for the definition of the CAFF. That is, it shows the needs (i.e. requirements, known as concerns in ISO42010) that must be fulfilled when defining a Framework that is to be used for the definition of other Architectural Frameworks.

The main need that must be fulfilled is to "Define an architectural framework for creating architectural frameworks", constrained by the need to "Comply with best practice" such as Architectural Framework Standards (for example, ISO42010). In order to "Define an architectural framework for creating architectural frameworks" it is necessary to:

- "Allow needs that the AF is to address to be captured" - When defining an Architectural Framework, it is important that the needs that the Architectural Framework is to address can be captured, in order to ensure that the Architectural Framework is fit for purpose.
- "Support definition of ontology for AF domain" - When defining an Architectural Framework, it is essential that the concepts, and the relationships between them, are defined for the domain in which the Architectural Framework is to be used. This is the Ontology that forms the foundational basis of the definition of the Architectural Framework's Viewpoints. Such an Ontology ensures the consistency of the Architectural Framework. The Architectural Framework must support such a definition of an Ontology.
- "Support identification of required viewpoints" - The Viewpoints that make up the Architectural Framework need to be identified. As well as supporting such an identification, the Architectural Framework must also
"Support identification of relationships between viewpoints" and "Support identification of grouping of viewpoints into perspectives".

- "Support definition of viewpoint needs" - In order to define the Viewpoints that make up an Architectural Framework, it is essential that the needs of each Viewpoint be clearly understood in order to ensure each Viewpoint is fit for purpose and that the Viewpoints defined meet the overall needs for the Architectural Framework.

- "Support definition of viewpoint content" - An Architectural Framework is essentially a number of Viewpoints that conform to an Ontology. Therefore, when defining an Architectural Framework it is essential that each Viewpoint can be defined in a consistent fashion that ensures its conformance to the Ontology.

- "Support definition of architectural framework rules" - Often, when defining an Architectural Framework, it is often necessary to constrain aspects of the Architectural Framework through the definition of a number of constraining Rules. It is therefore essential that an AF Framework supports the definition of such Rules.

The key Stakeholder Roles involved are:

- Architectural Framework Sponsor - the role involved in sponsoring the creation of the Architectural Framework.
- Architectural Framework Modeller - the role involved in the modelling and definition of an Architectural Framework.
- Architectural Framework Standard - the role of any appropriate standard for Architectural Frameworks. An example of a standard that could fill this role is ISO42010.
- Domain Expert - the role of an expert in the domain for which the Architectural Framework is to be used.

2.4.2. The Ontology for CAFF – The Ontology Definition View

This section defines the ontology for the CAFF, presented using SysML block definition diagrams. The diagrams are examples of a CAFF Ontology Definition View.
Figure 4 defines the ontology associated with Architectures and Architectural Frameworks:

The [Package] Ontology Definition View [AFs & Architectures] diagram defines the ontology associated with Architectures and Architectural Frameworks:

- **Architectural Framework** - a defined set of Viewpoints and an Ontology. The Architectural Framework is used to structure an Architecture from the point of view of a specific industry, stakeholder role set, or organisation. The Architectural Framework is defined so that it meets the needs defined by its Architectural Framework Concerns. An Architectural Framework is created so that it complies with zero or more Standards.
- **Architectural Framework Concern** - defines a need (requirement) that an Architectural Framework has to address.
- **Ontology** - an element of an Architectural Framework that defines all the concepts and terms (Ontology Elements) that relate to any architecture structured according to the Architectural Framework.
- **Ontology Element** - the concepts that make up an ontology. Ontology Elements can be related to each other and are used in the definition of each viewpoint (through the Viewpoint Element that makes up a viewpoint). The provenance for Ontology Elements is provided by one or more standards.
- **Viewpoint** - a definition of the structure and content of a view. The content and structure of a Viewpoint uses the concepts and terms from the Ontology via the Viewpoint Elements that make up the Viewpoint. Each viewpoint is defined so that it meets the needs defined by its Viewpoint Concerns.
- **Viewpoint Concern** - defines a need (requirement) that a Viewpoint has to address.
- **Viewpoint Element** - the elements that make up a Viewpoint. Each Viewpoint Element must correspond to an Ontology Element from the Ontology that is part of the Architectural Framework.
- **Architecture** - a description of a System, made up of a number of Views. Related Views can be collected together into Perspectives.
- **View** - the visualisation of part of the Architecture of a System, that conforms to the structure and content defined in a Viewpoint. A View is made up of a number of View Elements.
- **View Element** - the elements that make up a View. Each View Element visualises a Viewpoint Element that makes up the Viewpoint to which the View conforms.
- **Perspective** - a collection of Views (and hence also their defining Viewpoints) that are related by their purpose. That is, Views which address the same architectural needs, rather than being related in some other way, such as by mode of visualisation, for example.
- **Rule** - a construct that constrains the Architectural Framework (and hence the resulting Architecture) in some way, for example by defining minimum required Viewpoints.
- **System** - set of interacting elements organised to satisfy one or more needs. The artefact being engineered that the Architecture describes.

Two concepts illustrated in Figure 4 that are important to understand are those of viewpoint and view. An architectural framework is made up of a number of viewpoints that define the information that can be presented. When an actual architecture is developed that is based on the architectural framework, then one produces views that conform to the definitions in the corresponding viewpoint. That is, a view is a realisation of a viewpoint. Not all architectural frameworks make this distinction. For example, the UK Ministry of Defence Architectural Framework (MODAF – see [MODAF2010]) makes no such distinction. It defines a number of views but does not differentiate between the definition and realisation in terms of the language and terms used. Even more confusingly MODAF does use the term viewpoint, but in MODAF viewpoint is the same as perspective in Figure 4, simply a collection of related views.

The ontology is based on concepts from a number of sources, including: [ISO42010:2011] and [TRAK 2013]. The concept of context is expanded further in Figure 5 below, which is based on ideas that can be found in [Holt et al 2011] and [D21.1 2012].
Figure 5 defines additional ontology elements associated with Needs (requirements) that define the Contexts for Architectural Frameworks and the Viewpoints that make them up:

- **Context** - the definition of the *Needs* (for something (such as a System or an Architectural Framework) from a *specific point of view*. A Context is made up of a Boundary that defines the point of view of the Context, one or more Stakeholder Roles and one or more Use Cases.
- **Boundary** - defines the boundary, and hence the point of view, of a Context. The Use Cases that make up a Context are *inside* the Boundary. The Stakeholder Roles that make up a Context are *outside* the Boundary.
- **Use Case** - represents a need (requirement, concern) that is given meaning by being considered in a specific Context. Use Cases yield observable results to Stakeholder Roles and interact with other Use Cases.
- **Stakeholder Role** - the role of *anything* that has an interest in a System. Examples of a Stakeholder Role include the roles of: a person, an organisational unit, a project, a standard, an enabling system, etc. Stakeholder Roles lie outside the Boundary of a Context.
- **Concern** - a type of Need (a requirement). Architectural Framework Concerns represent the Needs for an Architectural Framework; Viewpoint Concerns represent the Needs for a Viewpoint.
2.4.3. The Viewpoints and Perspectives for CAFF – The Viewpoint Relationships View

This section identifies the viewpoints that make up the CAFF, presented using a SysML block definition diagram. Perspectives are also shown, through the use of SysML packages. It is an example of a CAFF Viewpoint Relationships View.

Figure 6 - Viewpoint Relationships View Showing Viewpoints and Perspectives that make up the AF Framework

Figure 6 shows the various Viewpoints that make up the Architectural Framework, the relationships between them and the Perspectives to which the Viewpoints belong.

The Architectural Framework defined here is a meta-Architectural Framework: an Architectural Framework for defining Architectural Frameworks. It is made up of six Viewpoints:

- The AF Context Viewpoint, that defines the Context for the Architectural Framework. That is, it represents the Architectural Framework Concerns in Context.
- The Ontology Definition Viewpoint defines the Ontology for the Architectural Framework. It is derived from the AF Context Viewpoint.
- The Viewpoint Relationships Viewpoint, that shows the relationships between the Viewpoints that make up the Architectural Framework and groups them into Perspectives. It is derived from the Ontology Definition Viewpoint.
- The Viewpoint Context Viewpoint, that defines the Context for a particular Viewpoint. That is, it represents the Viewpoint Concerns in Context for a particular Viewpoint. It is derived from the AF Context Viewpoint.
- The Viewpoint Definition Viewpoint, that defines a particular Viewpoint, showing the Viewpoint Elements (and hence the Ontology Elements) that appear on the Viewpoint.
- The Rules Definition Viewpoint, that defines the various Rules that constrain the Architectural Framework.
The Viewpoints are collected into a single Perspective, the AF & Architectures Perspective as shown by the enclosing package.

### 2.4.4. The Viewpoints of CAFF – Viewpoint Definition

This section defines the six viewpoints that make up the COMPASS AF Framework. These viewpoints are identified in Figure 6. For each viewpoint defined here there are three parts to the definition:

1. A *Viewpoint Context View* defining the needs that the viewpoint addresses.
2. A *Viewpoint Definition View* defining the contents of the viewpoint.
3. An example of the viewpoint defined.

The viewpoints are defined in the following order:

1. AF Context Viewpoint (AFCV)
2. Ontology Definition Viewpoint (ODV)
3. Viewpoint Relationships Viewpoint (VRV)
4. Viewpoint Context Viewpoint (VCV)
5. Viewpoint Definition Viewpoint (VDV)
6. Rules Definition Viewpoint (RDV)

The detailed descriptions and definitions of each of the 6 CAFF viewpoints are described in Appendix A.

### 2.4.5. The Rules Constraining the CAFF – The Rule Definition View

This section identifies some rules that constrain the CAFF, presented using a SysML block definition diagram. It is an example of a CAFF *Rule Definition View*.

---

**Figure 7 - Rules Definition View for the AF Framework**
Figure 7 shows five example rules that constrain the AF Framework. In this diagram, each Rule is represented by a SysML block that has had the «Rule» stereotype applied. Associated with this stereotype is a tag, Rule Text, that holds the text of the Rule and which is displayed in a separate compartment.

There is no reason why SysML blocks have to be used for the definition of rules, particularly in this case when all the rules are essentially textual descriptions wrapped in SysML blocks. They could simply have been described using the rule text on a purely textual "diagram". However, using blocks does bring with it the advantage of support for traceability. If it is desirable to show that elements of the Architectural Framework conform to the various Rules, then having them defined using blocks allows this to be done using the SysML traceability mechanism of stereotypes dependencies. Defining them purely as text would not allow this to be done.

This section presents an initial set of guidelines for architectural SoS modelling described as activities organized as a set of steps. Each of the following activities is described in more details in the following subsections.

The following activities are involved in architectural modelling:
1. Determine characteristics of the SoS and CSs
2. Determine SoS development context and lifecycle
3. SoS requirements analysis
   3.1 Determine the architectural-centric SoS Use Cases
   3.2 Prioritize the non-functional SoS requirements
4. Define SoS architecture
   4.1 Define communication paradigm
   4.2 Define architectural styles and patterns
5. Identify and specify CS
6. Validate SoS architectures against requirements
7. Define SoS-CS structure and behaviour
8. Define interfaces and contracts for CS
9. Document the selected architecture with SysML

The ordering of the different activities is not intended to suggest a specific process. Any process will however have to cover those activities in some order.

The material for the guidelines presented in this section is developed collaboratively with the case study owners. It is expected to be extended and evolved during the project and should not be considered final. We expect that the document will continue evolving also after the end of the COMPASS project as guidelines supporting industrial practice usually do and should.

3.1. Determine Characteristics of the SoS and CSs

The purpose of this activity is to determine characteristics of the SoS and CSs to be developed. Once important characteristics are known, a suitable SoS architectural styles can be chosen.

3.1.1. SoS Categories

SoS can be distinguished into the following five categories:
1. Directed
2. Acknowledged
3. Collaborative
4. Virtual
5. Hostile

The first four categories are those suggested by [Maier1998] and [Dahmann&2008]. We have added the fifth category “Hostile” that is similar to “Collaborative” but assumes that collaboration does not happen voluntarily as is often the case when products of competing companies are combined.
For completeness the definitions of these categories will shortly be repeated here as defined in the COMPASS concept base.

1. **Directed SoS**
   “An SoS built and managed to fulfil specific goals. Although the constituents can operate independently, within the SoS they accept some central management to ensure that SoS-level goals are met” [Maier 1998].

2. **Acknowledged SoS**
   “Acknowledged SoS have recognized objectives, a designated manager, and resources for the SoS, however, the CS retain their independent ownership, objectives, funding, as well as development and sustainment approaches. Changes in the systems are based on collaboration between the SoS and the system” [Dahmann&2008].

3. **Collaborative SoS**
   “The SoS has no coercive power over the CS, but they voluntarily choose to collaborate in order to achieve the SoS goals” [Maier 1998].

4. **Virtual SoS**
   “Virtual SoS lack a central management authority and a centrally agreed-upon SoS-level goal. Large-scale behavior emerges and may be desirable - but there is no visible active management of the SoS or its goals” [Maier 1998].

5. **Hostile SoS**
   “The SoS has no coercive power over the CS and they don’t voluntarily choose to collaborate in a given SoS to achieve the SoS goals”

We expect these categories to evolve over time together with the rest of these guidelines.

The differences between the categories are used later in the guidelines to indicate suitable architectural styles. An SoS usually belongs to more than one category. In general different aspects of the SoS need to be modelled using different architectural styles. This is reflected in the guidelines by pointing to different styles for the different sub-problems of the SoS.

### 3.1.2. CS Categories

CS can be distinguished into the following two orthogonal main categories:

1. **CS-SoS Connection** (static or dynamic)
2. **CS Openness** (Blackbox CS, Greybox CS or Whitebox CS)
1. **CS-SoS Connection (static or dynamic)**

Figure 8 shows a subdivision of a SoS into zero or more enabling systems and a number of CSs. An enabling system is a subsystem, which is not characterized as a CS as it only works inside the SoS, with the sole purpose of enabling the interaction between the CSs. An example of an enabling system could be a naming service, where the CSs can look up SoS services. A given CS can be characterised to be either statically connected to the SoS or to be available on a dynamic basis, where a dynamic CS can join or leave the actual SoS on run-time. A dynamically connected CS requires dynamic reconfiguration functionality from the SoS Architecture.

![Figure 8 - SoS consisting of enabling and constituent systems](image)

2. **CS Openness (Blackbox CS, Greybox CS or Whitebox CS)**

The CS Openness category describes the degree of openness towards the integration into a given SoS. The categorization consists of blackbox, greybox and whitebox CS.

A blackbox CS has to be integrated into a SoS without allowing any changes to be made to the CS. It can be further subdivided into a self-contained blackbox CS and a dependent blackbox CS. Figure 9 shows a self-contained blackbox, with a number of provided interfaces to be accessed by the SoS. Commercial examples of a self-contained blackbox CS are the Global Positioning System (GPS) and Google Maps, which provides services to its users.

![Figure 9 - Self-contained CS as a Blackbox](image)
Figure 10 shows a dependent blackbox CS, which has both a set of provided as well as a set of required interfaces, which must be satisfied by other CSs in the SoS.

These interfaces defined by the CS will need to be adapted by the other CSs in the SoS as the blackbox CS is a type of CS which cannot be changed to suit the SoS, for different reasons.

Figure 11 shows a greybox CS. A greybox CS is characterized by allowing dynamic installation of SoS applications, which function as integration code between the greybox CS and the SoS. An example of such an environment is an Android based CS allowing loading of Android applications onto the CS.

Figure 12 shows a whitebox CS. A whitebox CS requires and allows code changes in the CS to enable it to be integrated into a given SoS. Such integrations could, as an example, be performed by the introduction of wrapper components in the CS.
3.1.3. Determine SoS Characteristics

The characteristics are instrumental in this activity. Depending on the answers given the SoS is placed in certain categories, usually more than one. The reason for this is that SoS are usually complex systems where different aspects fit into different categories. The categories are used later to help matching the SoS to architectural styles.

I. Is there one dedicated body overseeing the **development** of the SoS?
   - **YES**
     Needs and requirements can be directly traced into the architecture that can be designed by the known stakeholders and agreed by the dedicated body.
   - **NO**
     **Minimal requirements** must be agreed upon that make the SoS operational. **Optional requirements** must be agreed upon that extend the capabilities of the SoS and leading to a requirement for a **Service discovery** functionality.

II. Is there one dedicated body overseeing the **operation** of the SoS?
   - **YES**
     This body must have available all information by the CS providers necessary to run and maintain the SoS.
   - **NO**
     Maintenance of the SoS cannot be expected to be carried out consistently. If there is not a set of **minimal requirements** the SoS may become at times inoperational.

III. Is it **possible** to determine all stakeholders?
   - **YES**
     All required interfaces can be determined and evolved in a coordinated way.
   - **NO**
     The possible evolutions of the system must be carefully planned. Interfaces must be extensible to satisfy new needs. The SoS must be robust to deal
with incompatibilities. For instance, there could be competing capabilities.

IV. Can all CS be **forced** to share certain protocols?
- **YES**
  All CS can be assumed to comply with the specified protocols. The SoS can be evolved by forcing the CS to change (slowly).
- **NO**
  It needs to be investigated which capabilities are concerned and whether a set of minimal requirements is attainable. If this is not the case, the remaining CSs should be operational without this CS. Emerging properties depending on such CS cannot be guaranteed.

V. Can all CS be **influenced** to share certain protocols?
- **YES**
  Usually, this means that the CS can be assumed to be developed towards a shared standard. Evolution of the standard will evolve the SoS in operation (slowly). The SoS must be able to deal with coexisting versions of the standard.
- **NO**
  The SoS must assume that some CS may become inoperational within the SoS. The SoS must be resilient in this case and continue operating as if the concerning CS were non-existent.

VI. Can the behaviour of a CS be **changed** or **adapted**?
- **YES**
  If the behaviour can be **changed**, it is considered a whitebox CS. Whiteboxes can be evolved with the rest of the SoS, in particular, preserving emergent behaviour. If it can only be **adapted**, adding and maintaining functionality may be difficult and important emergent behaviour should not depend on such added functionality. By being limited to adapting we mean that the CS cannot be changed, e.g., often commercial OS kernels cannot be changed.
- **NO**
  The constituent is considered as either a blackbox or a greybox. Emergent behaviour must only depend on what is reliably known. Other CS must be adapted to achieve this.

VII. Is the core behaviour of a CS specified and public?
- **YES**
  If the CS is a whitebox, maintenance will be easy. If it is not a whitebox, it is classified as a greybox CS. A greybox provides reliable information about the CS’s behaviour. In this case emergent behaviour can be based on the specification of the CS. A CS may be white, black or grey with respect to different aspects of its behaviour.
- **NO**
  This means that only specific behaviours are known but not their specification. In this case robust (fault-tolerant?) adaptors need to be provided.
by other components. This will ensure stability of the SoS and safeguard against changes in the behaviour of the CS.

VIII. Can all CS be assumed to be **committed** to the SoS?
- **YES**
  It can be assumed that emergent behaviour of the system is a common objective of all stakeholders. This makes it possible to agree on a set of emergent properties and have all CS implement the necessary functionality. The SoS as a whole can be maintained and evolved.
- **NO**
  CS not committed to the SoS may break emergent properties. It is not possible to evolve the SoS as a whole and it must be ensured that the minimal requirements of the SoS do not depend on functionality specific to uncommitted CS.

3.1.4. **Mapping Characteristics to Categories**

Figure 13 permits mapping the SoS characteristics to the categories. A roman number with “+” means the question has been answered affirmative; otherwise the number carries a “-”. Categories where most of the given answers are found should be the basis for the architectural choices provided later in this guideline. Note, that the determined category need not be unique. The need for evolving this schema arises when it presents too many indeterminate classifications.
3.2. Determine SoS Development Context and Lifecycle

Development of a concrete SoS can have different contexts and background often decided by or governed by several factors as, for example, political reasons, existence of legacy systems, commercial competitions etc.

This section will describe three different development contexts as shown on Figure 14 for development of a new SoS or evolution of an existing SoS, called top-down, bottom-up and middle-in depending on the starting point. The development contexts are shown to overlap as a given real life situation can have aspects of more than one of these three perspective. These three development contexts introduce another dimension to be taken into account in definition of the concrete architectural guidelines, as the architectural freedom will be different for the three contexts. Each of these will be described in the following subsections.

![Figure 14 - Three development contexts for SoS development](image)

3.2.1. SoS and CS Lifecycles

Figure 15 shows three life cycle states for a given SoS i.e. NEW, MID-LIFE or AGING and describes the focus points from a SoS Engineering view. Currently most SoS are in one of the two first states, but in a few years as more SoS are developed the AGING SoS state will be actual as well.

<table>
<thead>
<tr>
<th>SoS Maturity</th>
<th>SoSE Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>Develop a new SoS (e.g. directed SoS)</td>
</tr>
</tbody>
</table>
| MID-LIFE     | Enhance an existing SoS capability  
                Develop a new SoS capability  
                Enhance an existing SoS architecture/framework |
| AGING        | SoS currently exists, but needs major restructuring or a new framework/architecture, potentially requiring refactoring of some/all of the CS |

![Figure 15 - SoS maturity and SoSE Focus points](image)
A NEW SoS can be developed based on either the Top-down or the Bottom-up development context. A SoS in the MID-LIFE state is maintained based on the Middle-in development context.

Figure 16 shows how a given SoS can be characterized to be in one of the three SoS maturity states and for each of these, the actual CSs can as well be characterized for each CS to be in one of the three states NEW CS, MID-LIFE CS or AGING CS and the set of CSs will certainly be in different life-cycle states.

![Figure 16 - SoS and CS lifecycle maturity states](image)

### 3.2.2. SoS Top-down Development Context

In this development context shown on Figure 17, it is decided to build a new SoS from scratch, based on using existing CS as far as possible, where some of them can be existing Commercial Off The Shelf (COTS) systems or systems owned by other authorities, which are willing to be part of the new SoS.

This development context is relevant for development of a new directed or acknowledged SoS type, where a central authority or body decides to build new SoS.

Development of a Virtual SoS system can also be seen as a kind of top-down process, where the SoS system and corresponding standards are defined by a central authority without knowing the actual CS. This was the case in the development of the World Wide Web, where a central authority defined the standards for communication and interactions and set up a corresponding network, but where the future CSs defines the actual interactions and behaviour.
Guidelines for the top-down development context can be described as:

1. Define the overall SoS goals, objectives and requirements for the new SoS functionality, using the COMPASS guidelines for SoS requirements described in [D21.1 2012].

2. Define an appropriate and wanted SoS architecture and communication paradigm, using the COMPASS SoS architectural styles and patterns described in [D22.3 2013]. The top-down approach enables room for more freedom in selecting an optimal architecture for the given set of SoS requirements.

3. Find appropriate and optimal CSs to be used in the SoS and make the necessary commercial agreements. Search for either commercial of the shelf systems or services (CS) or existing CS, which are willing to be integrated into the new SoS and to collaborate with the other CS in the SoS.

4. Design the SoS architecture using appropriate architecture style, patterns and communication middleware, using [D22.3 2013] and standard architecture and patterns literature.

5. Modify the set of identified CS to participate in the selected SoS architecture.

This activity will normally be performed by the CS owner's organization in collaboration with the SoS authority developers. These modifications can be performed as either a white-box or a gray-box approach with development of SoS wrappers.

6. If an appropriate CS is not found, a new CS is foreseen to be developed from scratch, as the needed SoS functionality must be supplied by one of the CS.

7. Integrate and test the SoS and its CSs against the requirements.

8. Deploy the new developed SoS into operation.
3.2.3. SoS Bottom-up Development Context

In this development context shown on Figure 18, an existing set of systems (CS) joins together to collaboratively form a new SoS.

This development context could be relevant for a collaborative SoS type, where the owners of a set of existing systems (CS) decide to join to form a new SoS for obtaining new emergent behaviours, which are not possible without this joining.

Guidelines for the bottom-up development context can be described as:

1. The needs and vision for the new SoS, the overall SoS goals, objectives and requirements are defined.
2. Analyse the current existing set of CS and their possibilities to join into the new created SoS, with the stated requirements.
3. Define an SoS architecture and select the communication paradigm. This is based on the existing architectures supported by the majority of the CS.
4. Adapt the other CSs to this architecture, either by using the white-box approach by changing the CS-code, using the gray-box or black-box approach by development of appropriate wrappers.
5. Design and implement the SoS functionality according to the selected architecture.
6. Integrate and test the set of CS into the SoS.
7. Deploy the new SoS into operation.

3.2.4. SoS Middle-in Development Context

This context describes the development context where an SoS already exists consisting of a number of CS as shown on Figure 19, indicating the situations where a new CS is either added to the SoS, removed from the SoS or replaced with a new version of a CS.
This development context is also describing the SoS maintenance phase of the new SoS developed after either the top-down or the bottom-up contexts, described in the two previous sections.

**Evolution**
The concept of system evolution applies both at the SoS-level and the CS-level. An SoS and its CS may have long lifetimes, with each CS often in a different stage of its individual system life cycle. Evolution is natural for these long-lived systems, where changes can result from technological changes, new or changed user capabilities, or new legal requirements, e.g. government legislation.

**Emergence**
The concept of emergent behaviour applies only at the SoS level. It is a characteristic that emerges at the SoS-level as a result of the interaction between a number of CS and is a behaviour which cannot be achieved by, or attributed to, any of the individual systems.

The middle-in development context can be subdivided into the following 6 different and more specific cases:

1. Adding a new CS
2. Removing a CS
3. Replacing a CS
4. Evolution of a specific CS
5. Evolution of SoS
6. Adding of new SoS emergent behaviour

Guidelines for these cases will shortly be presented below.

*Guidelines for case 1: Adding a new CS,* can be described as:

1. Determine if the new CS is to be statically or dynamically available for the SoS.
2. Study the existing SoS architecture and communication paradigm.
3. Adjust the new CS to this architecture and communication paradigm either by using the whitebox or greybox approach e.g. by development of wrappers. A blackbox CS can only be integrated into the SoS, if it fits into the existing architecture and communication paradigm.
4. Test the SoS with the newly added CS together with the existing set of CSs and test if unexpected emergent behaviour has been produced.

**Guidelines for case 2: Removing a CS**, can be described as:
1. Analyse the CS-SoS functionality provided by the removed CS.
2. Determine if this removed functionality still must be available.
3. If YES: implement the functionality in one or more of the other CS or look at case 3 – replacing a CS.
4. Test the modified SoS.

**Guidelines for case 3: Replacing a CS**, can be described as:
1. Analyse the CS-SoS functionality provided by the CS to be replaced.
2. Find a replacement CS, with the same functionality.
3. If NO ONE FOUND: implement a new CS with the same functionality and architecture.
4. Test the modified SoS with the replaced CS.

**Guidelines for case 4: Evolution of a specific CS**, can be described as:
1. Analyse the CS to be modified and determine if the modification concerns the CS-SoS functionality.
2. If YES to 1: implement the changed functionality.
3. Test the SoS with the updated CS to be sure that the SoS still works as specified.

**Guidelines for case 5: Evolution of SoS**, can be described as:
1. Analyse the nature of the SoS evolution and determine the impact on the set of CS.
2. Determine if this evolution involves changes to SoS architecture or communication paradigms.
3. If YES to 2: find the involved CS and modify these
4. If YES to 2: test the complete SoS
5. Determine if changes are related to existing SoS requirements.
6. IF YES to 5: find the involved CS and modify them to apply with the evolving requirements. Using the COMPASS requirement method can help to ensure that appropriate traceability is in place between requirements and CS.
7. If YES to 5: test the complete SoS.

**Guidelines for case 6: Adding of new emergent behaviour**, can be described as:
1. Analyse the new emergent behaviour and determine the set of CS involved in obtaining this behaviour.
2. Modify the identified set of CSs to implement the new behaviour.
3. Test the SoS against the requirements for the new emergent behaviour.
3.3. SoS Requirements Analysis

This activity is a requirements analysis activity where the SoS requirement specification is analysed with the purpose of defining the architectural drivers for the SoS architecture decisions.

3.3.1. Determine the SoS Context diagram

This activity determines the set of actors which are external to the SoS and representing either user roles, hardware units or other systems interacting with the SoS. External systems characterized as actors are not recognized as CS, as a CS is defined to be a part of the SoS, which means it is inside the SoS context. Figure 20 shows an example of a SysML SoS context diagram.

![Figure 20 - SysML SoS Context Diagram](image)

3.3.2. Determine the Architecture-centric SoS Use Cases

This activity assumes that the SoS functional requirements are described as Use Cases and optionally documented on a SysML Use Case diagram (Figure 21). If this assumption is not satisfied it is strongly recommended to develop such Use Cases describing the SoS functionality and optionally show these on a Use Case diagram. Based on these SoS Use Cases and diagram, a selection of the set of Use Cases that have an impact on the SoS architecture modelling seen from the functional perspective, are selected.

![Figure 21 - SysML Use Case Diagram](image)

The result will be a list of architecture-centric Use Cases, which are listed in a prioritized order.
3.3.3. Prioritize the Non-functional SoS Requirements

Identify and prioritize the subset of the non-functional SoS requirements, that have an impact on the SoS architecture. Prioritizing is made according to the defined SoS project goals and objectives. Examples of non-functional requirements are performance and safety related requirements.

3.3.4. Tracing SoS Requirements into the SoS Architecture

If a requirement addresses a CS directly, the CS stakeholders can validate it on the CS level using the appropriate approach. Whether this is the case or not may depend on concrete configurations of the SoS architecture.

If a requirement spans several CSs, it always concerns the architecture in the following way: it is necessary to put forward an argument for why the requirement is satisfied by the SoS. This argument will invoke all CSs that are necessary in order to realise the requirement. Such arguments serve two purposes:

1. It provides an informal proof of the fitness of the SoS for the required purposes.
2. It helps in identifying CS and stakeholders that are involved in the realisation of that specific requirement.

Usually the argument will depend on suspected, known and specified functionality of the CS and take account of the interfaces and communication techniques employed. This can be seen as a “test” of these entities. Usually, this will point to problems of the current architecture that need to be solved among the stakeholders. The characteristics analysis hints at possible strategies for this. This activity needs to be carried out continually while the architecture of the SoS is developed and made gradually more specific.

3.4. Define SoS Architecture

This section presents guidelines for a definition of the SoS architecture style and communication paradigm together with selection of appropriate architecture patterns to support definition of the architecture.

This selection will be based on the initial set of architecture styles and patterns described in the COMPASS document [D22.3 2013].

3.4.1. Define Communication Paradigm

Interfaces between SoS and the CS depends on the selected communication paradigm. Selecting a communication paradigm will decide the types of interfaces as well as the communication between the CS.

Software oriented SoS can have one of the following two communication paradigms:

1. A function oriented communication approach as described by e.g. Service Oriented Architectures (SOA) and Web Services (WS), leading to interfaces described by signatures for methods to be called.
A function oriented approach is specified using the *Interface Definition Pattern*, described in [D22.3 2013].

2. A *data oriented* publish/subscribe communication approach as described by the OMG Data-Distribution Service for Real-Time Systems (DDS) [DDS-std 2007], [DDS 2012], leading to interfaces described by a data model based on topics.

### 3.4.2. Define Architectural Styles and Patterns

Input for this activity is the selected communication paradigm, the list of architectural centric Use Cases and the prioritized list of non-functional requirements. This input is used together with the initial set of architectural styles and patterns described in [D22.3 2013] to define one or more candidates for the SoS Architecture. Architecture styles are dominating a given SoS architecture, where an architecture pattern is a more localized architecture solution, which can be used to realize a given architecture style.

The following architecture styles are identified and described in [D22,3 2013] in relation to SoS and the list is expected to be expanded with more styles and patterns during the COMPASS project:

1. Centralized architecture
2. Service Oriented architecture
3. Publish-Subscribe architecture
4. Pipes and Filters architecture
5. Blackboard architecture

### 3.5. Identify and Specify CS

The SoS requirements are allocated to a set of CS and each of these is identified, categorized and specified.

The categorization can use the categories defined in section 3.1.2:

1. CS-SoS Connection (static or dynamic)
2. CS Openness (Blackbox CS, Greybox CS or Whitebox CS)

For a given CS it can be useful to distinguish between the CS functionality used outside or before joining the SoS and the added SoS-CS functionality as indicated on Figure 22. In this figure Use Cases UC1 to UC5 are from the original CS, where they interact with the actors A1 to A3, which are outside the SoS. The SoS-CS part interacts with three other CSs modelled as actors and called SoS-CS1, SoS-CS2 and SoS-CS3. The SoS required functionality from the SoS-CS are modelled as the four Use Cases called SoS-UC1 to SoS-UC4. It is also indicated that SoS-UC2 interacts with the original UC3 for example by an extends relation and the same for SoS-UC4 which interacts with the original UC4. Such an overview diagram can be used to make a clear distinction between the original CS functionality and the added SoS-CS functionality for participating in the actual SoS. The diagram also
shows that future changes to the original Use Cases UC3 and UC4 could have an impact on the SoS-CS functionality described by the “SoS UC2” and “SoS UC4”.

3.6. Validate SoS Architectures against Requirements

This section describes an initial architectural validation method to be used in validation of the SoS architecture in relation to the SoS functional requirements and quality attributes, the type of SoS and the actual SoS development context.

This section will, as future work, address the Architecture Trade-off Analysis Method for SoS being investigated by the Software Engineering Institute (SEI).

This initial method is expected to be enhanced during the COMPASS project based on the experiences gained by using it in the industrial cases.

3.7. Define SoS-CS Structure and behaviour

In this activity the structure of the SoS with the identified CSs is shown on one or more SysML Block Definition Diagrams (BDD).

Figure 23 shows an example of a SysML block definitions diagram, where the SoS “Use case 3” (SoS UC3) is allocated to be satisfied by three CS i.e. CS1, CS2 and CS3.
The SoS-UC3 behaviour can be designed using a SysML sequence diagram as shown on Figure 24, where the interactions are shown between the actors and the identified CS. If some of these CS have state dependent behaviour the behaviour shall include a SysML state diagram as well.

### 3.8. Define Interfaces and Contracts for CS

A *function oriented* communication approach is specified using the *Interface Definition Pattern*, described in details in [D22.3 2013].

A *data oriented* communication approach is specified using the [DDS-std 2007] standard. In this approach a number of topic are defined together with the actual publishers and subscribers, where each publisher (e.g. a CS) and subscriber (e.g. another CS) can specify a large number of Quality of Services (QoS) for the interactions. These QoS forms contracts for the interaction.
3.9. Document the selected SoS Architecture with SysML

The selections and decisions made in the previous steps are documented with a set of architecture views using SysML together with descriptions of the design rationales used to define the actual architecture. This documentation will follow the views to be defined by the project specific AF defined using the meta-architectural framework presented in Section 2.
4. Relevance in relation to COMPASS Case Studies

The plan is to start with testing the relevance of the guidelines for the acknowledged and the collaborative SoS types as the two industrial cases in COMPASS i.e., Insiel and Bang & Olufsen, are categorized as being acknowledged for the Insiel case and collaborative for the Bang & Olufsen case.

4.1. Relevance in relation to the B&O Case Study

The following section will present the relations of the architectural SoS modelling guidelines for a Collaborative SoS using the B&O case study as example. For simplicity, only the streaming architecture part of the B&O SoS will be addressed in this section. This section begins with a brief introduction to basic SoS streaming concepts and Use Cases, and then the guidelines are used for modelling a SoS architecture that meets B&O’s SoS streaming requirements.

The streaming architecture concepts

The B&O streaming system(s) uses the concept of distributed flow graphs to deal with multimedia contents. A flow graph consists of a number of interconnected nodes. Source nodes produce data which filter nodes modify, and sink nodes render the data to either a screen or loudspeaker. Products providing source node services are named source products. Filter- and sink node services are provided by products named renderer products.

The protocol(s) used for streaming audio over an IP network is broadcasting stream events and buffers. Events contain information related to the state of the stream or actions to perform on the stream, like flush, pause or resume the stream. The Event ordering rule set allows source and renderer products to reach agreement regarding the distributed state of streams. Buffers contain raw media data. An example of the relation between events and buffers is before a source product sends the first media buffer, a format event is sent. Based on the event information, the renderer product can now correctly decode the incoming buffers. The streaming architecture will be explained further as the guidelines are applied to several views of the system.

Figure 25 - Multiple product scenario
Streaming architecture-centric SoS use cases

B&O wish to support higher interoperability with 3rd party products like Apple in the SoS while still preserving B&O’s streaming experiences. The simplified architecture-centric use case is:

Basic streaming
- Actors:
  - User
- Pre-condition:
  - No source or renderer products are streaming. User has selected content and source product to stream from and selected renderer product to stream to.
- Basic flow:
  - User presses play
  - Source product connects to renderer product
  - Source product sends stream events and data buffers
  - Renderer product consumes stream events and data buffers.
- Post-condition:
  - Selected media content is playing in the zone of the renderer product.

Join experience streaming (extend basic streaming use case)
- Actors:
  - User
- Pre-condition:
  - Source product s is streaming to renderer product r located in zone a. User is standing in zone b and has selected the current streaming experience of s and r.
- Basic flow:
  - User presses join
  - Source product s connects to renderer product r’ of zone b
  - Source product sends stream events and data buffers to renderer products r and r’.
  - Renderer products r and r’ consume stream events and present media buffer x with a max delay of 20 milliseconds between the renderers’ presentation of buffer x.
- Post-condition:
  - Media content is playing in zones a and b.

A detailed introduction to Bang & Olufsen case study is described in [D42.1 2013].
4.1.1. Determine B&O SoS Type

This section will determine B&O SoS type by answering the set SoS characteristic questions from section 3. The answers will formulate the B&O SoS problem domain and a SoS type will be derived from properties of the problem domain.

Is there one dedicated body overseeing the development of the SoS?

NO

"Minimal requirements must be agreed upon that make the SoS operational. Optional requirements must be agreed upon that extend the capabilities of the SoS. Service discovery is required"

Some CS of the SoS are controlled by B&O, but not all. The AirPlay\(^1\) protocol used by Play products is managed by Apple. Content provider CS like music services are not managed by B&O either, and can enforce constraints on the SoS.

The Minimal requirements for any renderer or source CS to join the B&O SoS are:

- Device discovery
- Service discovery
- Streaming protocol(s)

The set of Optional requirements is based on the CS’s completeness of the Minimal requirements. As an example the AirPlay streaming protocol does not provide support for multi-room synchronized sound, but B&O products do. So Optional requirements are defined for specific SoS product configurations.

Is there one dedicated body overseeing the operation of the SoS?

YES

"This body must have available all information by the CS providers necessary to run and maintain the SoS"

B&O manages the operation(s) of the SoS based on information from the CS. Information collected are standard specifications or by co-operation with the CS providers.

Is it possible to determine all stakeholders?

NO

"The possible evolutions of the system must be carefully planned. Interfaces must be extensible to satisfy new needs. The SoS must be robust to deal with incompatibilities. For instance, there could be competing capabilities"

Uncontrolled CS of the B&O SoS like AirPlay, DLNA\(^2\) and content providers indicate the possibility of invisible stakeholders whose evolution vision for their CS

---

1 AirPlay (previously called AirTunes when it was for audio only\([1]\)) is a proprietary protocol stack/suite developed by Apple Inc. that allows wireless streaming of audio, video, and photos, together with related metadata between devices

2 The Digital Living Network Alliance (DLNA) is a non-profit collaborative trade organization established by Sony in June 2003, that is responsible for defining interoperability guidelines to enable sharing of digital media between multimedia devices
is not compliant with B&O’s SoS vision. The Minimal requirement for Service discovery is an evolution strategy for satisfying new needs and handling incompatibilities. CS capabilities are described through service interfaces. The product of services for a SoS configuration defines the experience capabilities of the SoS.

**Can all CS be forced to share certain protocols?**

**NO**

“It needs to be investigated which capabilities are concerned and whether a set of minimal requirements is attainable. If this is not the case the remaining CS should be operational without this CS. Depending on such CS cannot be guaranteed”

Not all CS of a B&O SoS configuration provide the needed services or can be forced to share protocols for participation in, or contributing to the desired experiences of the SoS.

Desired Emerging properties like multi-room synchronized sound depend on synchronization services agreements among the CS. The general strategy for dealing with inadequate CS for B&O is:

1. If the service capabilities of the CS is inadequate and can degrade the whole experience capabilities of the SoS, exclude the CS from the SoS.
2. If the service capabilities of the CS is inadequate but do not degrade the whole experience capabilities of the SoS, include the CS in the SoS.

   - DRM content providers belong to this strategy. DRM content cannot be used for multi-room synchronized sound experience but can participate in multi-room browsing experiences.

Whether protocols can be shared or not depends on the collaboration arrangement between the CS’ owners. B&O are forced by Apple to use the AirPlay protocol for interoperability with iOS products, but B&O cannot force Apple to support B&O’s streaming protocol in iOS. On the other hand, Intel co-operates with B&O regarding support of B&O’s streaming protocol at the ISMD hardware flowgraph layer.

**Can all CS be influenced to share certain protocols?**

**YES**

“usually, this means that the CS can be assumed to be developed towards a shared standard. Evolution of the standard will evolve the SoS in operation (slowly). The SoS must be able to deal with coexisting versions of the standard”

UPnP, UPnP-AV and DNLA and standards for AV systems, and to some degree the industry supports these standards, but unfortunately with differing visions depending on the business strategy of the companies. This has resulted in very limited common required functionality, and a lot of optional functionality for these standards. None of the AV standards can implement B&O’s Minimal requirements. None of the standards support multi-user experiences or synchronization of streams. There are, however, specification proposals for both DLNA and UPnP-
AV regarding supporting synchronization of streams, so they might over time evolve to support B&O requirements. But currently B&O must support a mix of both proprietary technology and standards.

**Can the behaviour of a CS be changed or adapted?**

NO

"The constituent is considered a grey or blackbox. Emergent behaviour must only depend on what is reliably known. Other CS must be adapted to achieve this"

*AirPlay* cannot be changed or adapted (blackbox). UPnP-AV and DNLA are AV standards, which can be adapted with proprietary technology deployed on open eco-systems like Android or by collaborating with device makers. Desired *Emergent behaviours* cannot be achieved with current streaming blackbox or greybox CS’s without modification of the whitebox and greybox CS’s of the B&O SoS.

**Is the core behaviour of a CS specified and public?**

YES

"If the CS is a whitebox, maintenance will be easy. If it is not a whitebox, it is classified as a greybox. A greybox provides reliable information about the CS’s behaviour. In this case emergent behaviour can be based on the specification of the CS. A CS may be white, black or grey with respect to different aspects of its behaviour"

All CS’s provide public interfaces/specifications which to some extent define core AV behaviours. But often the core behaviours do not cover, or cannot contribute to B&O’s desired *Emergent behaviours* without modification and/or adaption strategies.

**Can all CS be assumed to be committed to the SoS?**

NO

"CS not committed to the SoS may break emergent properties. It is not possible to evolve the SoS as a whole and it must be ensured that the minimal requirements of the SoS do not depend on functionality specific to uncommitted CS"

Apple has no interest in contributing to B&O’s SoS. Often CS’s of the B&O SoS are not aware of their participation or contributions to the SoS’s operations. The challenge for B&O is translating the *minimal requirements* to requirements for dealing with *uncommitted CS*.

Mapping the characteristics answers to a SoS category using the approach described in section 3.2.1 places B&O’s streaming SoS in the union space of *collaborative* and *virtual* with some shared elements with the *hostile* SoS type.
It is important to understand that the SoS type represents several different “views” of the SoS. During the streaming operations, the CS’s co-operate towards a common goal. This is a collaborative property of the SoS. But the SoS lacks a central management authority which controls the evolution and goals of the SoS. So from a SoS management view, the SoS is Virtual.

4.1.2. Determine B&O SoS Development Context

Based on needs derived from the Use Cases and the existing architecture SoS characteristics, the development contexts guidelines selected for the streaming part of the B&O SoS is the Middle-in context described in Section 3.2.3. This section will only address the case where a new source-CS is added to the SoS and where the source-CS being added is an iOS-based device. The guidelines should produce an SoS candidate architecture which satisfies the needs determined by the streaming use cases.

SoS Middle-in Context - guidelines for case 1:
Adding a new iOS based CS to B&O Streaming SoS

Case 1-1: Determine if the new iOS based CS is to be statically or dynamically available for the B&O Streaming SoS

The new iOS based CS has to be dynamically available to the B&O Streaming SoS. The streaming protocols and device services of iOS products must support dynamic reconfiguration functionality.

---

3 iOS (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc.
Case 1-2: Study the existing SoS architecture and communication paradigm

SoS architecture:
The streaming SoS architecture style is a *Pipes and Filters Architecture* style. The pipes and filter architecture pattern is a candidate architecture for SoS where the CS are connected in a serial manner and working on a common stream of data or material, where each CS transforms the input stream to an output stream and where the CS can be either a filter-CS or a pipe-CS. This Architecture style corresponds to the concepts of streaming flow graph(s) where *Source-CS* produces media streams and *renderer-CS* operates on or consumes media streams.

*Figure 27 - B&O streaming architecture style*

The *Pipes and Filters Architecture* style is described in COMPASS deliverable D22.3 [D22.3 2013]

Communication paradigm:
Based on the communication concepts of flow graphs, the communication paradigm style for B&O’s streaming SoS is a *Data-Centric Publish Subscribe* style. Source product(s) “Publish” media data to “Subscribers” (the renderer products) in a *one-to-many communication* relation. The communication domain between publisher and subscribers is defined by the streaming protocol used. In this case the *AirPlay* protocol.

*Figure 28 - B&O streaming communication style*
The *Data-Centric Publish Subscribe* style is described in COMPASS deliverable D22.3 [D22.3 2013]

**Case 1-3: Adjust the new CS to this architecture and communication paradigm (i.e. development of wrappers)**

iOS products are blackbox products, with elements of grey. Apple provides application layer development frameworks with interfaces to network technology. This means B&O could implement their proprietary streaming protocols as an iOS application. But B&O wishes to support any iOS products that are compliant with the AirPlay protocol, without deployment of customized applications so that all iOS applications/devices can stream to the B&O SoS.

The only way to implement the streaming-centric use-cases is by supporting the AirPlay protocol. This means designing wrapper/bridging logic in B&O’s streaming architecture, which translates AirPlay protocol semantics to B&O streaming protocol semantics. The *enabling pattern* used for modelling the bridging logic is the interface pattern. The main aims of this pattern are to support:

- Identification of interfaces and their relation to the system elements that use them and the ports that expose them
  - AirPlay exposes *Service discovery* interfaces using multi-cast ports.
  - AirPlay exposes *Streaming service* interfaces based on the RTSP protocol.
  - AirPlay services are exposed by source products and used by renderer products.

- Definition of interfaces in terms of the operations they may provide and the flows of data, material, energy, personnel etc. that take place across an interface
  - Definitions of internal bridging interfaces for AirPlay to B&O streaming semantics. Bridging interfaces for *events, media buffers* and *time synchronization*.

- Definition of any protocols that an interface or port must conform to
  - The *public* ports and interfaces of B&O products must conform to Airplay protocols.
  - The internal *private* ports and interfaces of B&O products must conform to B&O’s proprietary streaming protocols.

- Identification of typical scenarios showing how interfaces are used
  - Identification of streaming scenarios for *Event, media buffers* and *time synchronization* interfaces.
  - Identification of scenarios for *service discovery* interfaces.

*The Interface pattern* is described in COMPASS deliverable D22.3 [D22.3 2013]
**Case 1-4: Test the SoS with the new added CS together with the existing set of CSs**

Designing test strategies for verification of the streaming candidate architecture is done using the *Test pattern*. The pattern consists of a number of *test views*.

The views used for defining B&O's V&V campaign for streaming are:

- **‘Testing Context View’** drives the entire testing activity and is the number of testing Use Cases that can be derived from the *Streaming architecture-centric SoS Use Cases*.
- **‘Test Case View’** identifies the elements under test and its associated system elements. In the case of B&O streaming *SoS*, the test elements are B&O renderer products and Airplay based source products. The *Test Case View* models the needed test environment structures for verification of the *Testing Context View’s* test cases.

The *Test pattern* is described in COMPASS deliverable D22.3 [D22.3 2013]
5. Summary

This deliverable contains the results of an initial study into architectural modelling of SoS.

The importance of having an architectural framework in place for development of an architectural description for a given SoS is described in Section 2. This section introduces the COMPASS AF Framework (CAFF), which is a meta-framework defined for developing project specific AF for SoS development.

Section 3 presents the initial set of guidelines for defining SoS architectures. It starts with an activity to define the actual SoS and CS characteristics for the SoS to be developed or implemented. In this activity the SoS as a whole is characterized as well as each of the CS. The CSs are categorized into two dimensions, one depending on the connection (static or dynamic) and the other depending on the degree of openness towards integration into the SoS (whitebox, greybox and blackbox). In the next activity the actual SoS development context is described as either Top-Down, Bottom-Up or Middel-In, with a suggested number of activities for the different situations. The guidelines for defining the architecture are using the initial set of SoS modelling patterns described in [D22.3 2013].

In the final version of this guideline document the documentation of the SoS architecture will be documented with the views specified in the architectural framework.

Section 4 describes an initial test of the guidelines presented in Section 3, where the guidelines are used on the COMPASS B&O case study.

Delivery D21.5 will present the final report of the Architecture guidelines, developed and tested during the remaining part of the COMPASS project.
6. References


|------------|---------------------------------------------------------------------------------------------------------------|
Appendix A   Viewpoints of CAFF: Viewpoint definitions

A.1. The Viewpoints of CAFF – Viewpoint Definition

This section defines the six viewpoints that make up the COMPASS AF Framework. These viewpoints are identified in Figure 6. For each viewpoint defined here there are three parts to the definition:

4. A Viewpoint Context View defining the needs that the viewpoint addresses.
5. A Viewpoint Definition View defining the contents of the viewpoint.
6. An example of the viewpoint defined.

The viewpoints are defined in the following order:
7. AF Context Viewpoint (AFCV)
8. Ontology Definition Viewpoint (ODV)
9. Viewpoint Relationships Viewpoint (VRV)
10. Viewpoint Context Viewpoint (VCV)
11. Viewpoint Definition Viewpoint (VDV)
12. Rules Definition Viewpoint (RDV)

The examples are all taken from CAFF itself.

A.2. AF Context Viewpoint (AFCV)

This section defines the AF Context Viewpoint (AFCV).

**Viewpoint Context View**

The needs that the AF Context Viewpoint (AFCV) is intended to address are shown in the following diagram, a CAFF Viewpoint Context View.

![AF Context Viewpoint Diagram](image-url)
Figure 29 shows the Viewpoint Context View for the AF Context Viewpoint. That is, it defines the needs that the AF Context Viewpoint must address, together with relevant Stakeholder Roles.

The main need, taken from Architectural Framework Context (see Figure 3) is to "Allow needs that the AF is to address to be captured"; the AF Context Viewpoint exists solely to capture the needs of the Architectural Framework being defined.

In order to do this, it is necessary to be able to:

- "Identify AF needs" - Identify the needs that the Architectural Framework is being created to address.
- "Understand relationships between needs" - Understand any relationships between the needs that the Architectural Framework is being created to address.
- "Identify AF stakeholder roles" - Identify the Stakeholder Roles involved in definition of the Architectural Framework and that have an interest in or are affected by the identified needs.

As identified on Figure 3, the two key Stakeholder Roles involved are (through the «include» relationships from "Define an architectural framework for creating architectural frameworks"): the Architectural Framework Sponsor and the Architectural Framework Modeller.

**Viewpoint Definition View**
The definition of the AF Context Viewpoint (AFCV) is shown in the following diagram, a CAFF Viewpoint Definition View.
Figure 30 addresses the needs defined on Figure 29.

It defines the content of the AF Context Viewpoint, which defines the Context for an Architectural Framework. It is made up of a Boundary, one or more Stakeholder Roles, that are outside the Boundary, and one or more Use Cases that are inside the Boundary.

Each Use Case represents an Architectural Framework Concern (a type of Need) for the Architectural Framework, that yields observable results to one or more Stakeholder Roles. Each Use Case may interact with a number of other Use Cases.

**Example View**
For an example of an AF Context View, see Figure 3.

**A.3. Ontology Definition Viewpoint (ODV)**
This section defines the Ontology Definition Viewpoint (ODV).

**Viewpoint Context View**
The needs that the Ontology Definition Viewpoint (ODV) is intended to address are shown in the following diagram, a CAFF Viewpoint Context View.
Figure 31 shows the Viewpoint Context View for the Ontology Definition Viewpoint. That is, it defines the needs that the Ontology Definition Viewpoint must address, together with relevant Stakeholder Roles.

The main need, taken from the Architectural Framework Context (see Figure 3) is to "Support definition of ontology for AF domain"; the Ontology Definition Viewpoint exists to define the Ontology that defines all the concepts and terms (Ontology Elements) that relate to any Architecture structured according to the Architectural Framework.

The main needs that must be addressed are to:

- "Identify ontology elements" - Identify the Ontology Elements for the domain in which the Architectural Framework will be used.
- "Identify ontology relationships" - Identify the relationships between the Ontology Elements. Such relationships are equally as important a part of the Ontology defined using this Viewpoint as are the Ontology Element.
- "Identify ontology areas" - When defining an Ontology it is often useful to group together related Ontology Elements. For example, when defining an Ontology for systems engineering, one could expect to see groupings of Ontology Element related to the concepts of System, Life Cycle, Process, Project etc. Such groupings are useful when defining an Architectural Framework, as they help to identify the Perspectives into which the Viewpoints, and the Views based on them, are grouped.
As identified on Figure 3, the three key Stakeholder Roles involved in the main need to "Support definition of ontology for AF domain" are (through the «include» relationships from "Define an architectural framework for creating architectural frameworks" and directly): the Architectural Framework Sponsor, the Architectural Framework Modeller and the Domain Expert.

**Viewpoint Definition View**
The definition of the Ontology Definition Viewpoint (ODV) is shown in the following diagram, a CAFF Viewpoint Definition View.

![Figure 32 - Viewpoint Definition View for the Ontology Definition Viewpoint](image)

Figure 32 addresses the needs defined on Figure 31.

It defines the content of the Ontology Definition Viewpoint, which is made up of an Ontology that is itself made up of one or more Ontology Element(s) that are related to each other.

It is important to note that the relationships between the Ontology Elements are themselves an important part of the Ontology.

**Example View**
For an example of an Ontology Definition View see Figure 4 and Figure 5.

**A.4. Viewpoint Relationships Viewpoint (VRV)**
This section defines the Viewpoint Relationships Viewpoint (VRV).

**Viewpoint Context View**
The needs that the Viewpoint Relationships Viewpoint (VRV) is intended to address are shown in the following diagram, a CAFF Viewpoint Context View.
Figure 33 shows the Viewpoint Context View for the Viewpoint Relationships Viewpoint. That is, it defines the needs that the Viewpoint Relationships Viewpoint must address, together with relevant Stakeholder Roles.

The main need, taken from the Architectural Framework Context (see Figure 3) is to "Support identification of required viewpoints", which includes the need to "Support identification of relationships between viewpoints" and "Support identification of grouping of viewpoints into perspectives". The Viewpoint Relationships Viewpoint exists to identify the Viewpoints that make up the Architectural Framework, show how they are related to each other and show how they are grouped into Perspectives.

The needs for this Viewpoint are subject to two constraints:

- The main need to "Support identification of required viewpoints" is constrained by the need to "Be consistent with overall needs of the architectural framework". The Viewpoints identified as being required in the Architectural Framework must meet the needs for the Architectural Framework as defined on the AF Context Viewpoint.

- The need to "Support identification of grouping of viewpoints into perspectives" is constrained by the need to "Be consistent with ontology areas defined for ontology". While it is not essential that the Perspectives identified on the Viewpoint Relationships Viewpoint should be the same as the groupings of Ontology Elements, as identified on the Ontology Defi-
As identified on Figure 3, the two key Stakeholder Roles involved in the main need to "Support identification of required viewpoints" are (through the «include» relationships from "Define an architectural framework for creating architectural frameworks"): the Architectural Framework Sponsor and the Architectural Framework Modeller. The Domain Expert is also a key Stakeholder Role with an interest in the need to "Be consistent with ontology areas defined on for ontology".

**Viewpoint Definition View**
The definition of the Viewpoint Relationships Viewpoint (VRV) is shown in the following diagram, a CAFF Viewpoint Definition View.

![Viewpoint Definition View for the Viewpoint Relationships Viewpoint](image)

Figure 34 addresses the needs defined on Figure 33. It defines the content for the Viewpoint Relationships Viewpoint.

The Viewpoint Relationships Viewpoint is made up of one or more Viewpoints and shows the relationships between them. It is also made up of one or more Perspectives and shows which Viewpoints are in which Perspective.

The Viewpoint Relationships Viewpoint is often known as the "Viewpoint Quagmire".

**Example View**
For an example of a Viewpoint Relationships Viewpoint see Figure 6.
A.5. Viewpoint Context Viewpoint (VCV)

This section defines the Viewpoint Context Viewpoint (VCV).

**Viewpoint Context View**

The needs that the Viewpoint Context Viewpoint (VCV) is intended to address are shown in the following diagram, itself a CAFF Viewpoint Context View.

![Figure 35 - Viewpoint Context View for the Viewpoint Context Viewpoint](image)

Figure 35 shows the Viewpoint Context View for the Viewpoint Context Viewpoint. That is, it defines the needs that the Viewpoint Context Viewpoint must address, together with relevant Stakeholder Roles.

The main need, taken from the Architectural Framework Context (see Figure 3) is to "Support definition of viewpoint needs"; the Viewpoint Context Viewpoint exists to capture the needs of a Viewpoint being defined. In order to do this, it is necessary to "Be consistent with needs of AF" and to be able to:

- "Identify viewpoint needs" - Identify the needs that the Viewpoint is being created to address.
- "Understand relationships between needs" - Understand any relationships between the needs that the Viewpoint is being created to address.
- "Identify viewpoint stakeholder roles" - Identify the Stakeholder Roles involved in the definition of the Viewpoint and that have an interest in or are affected by the identified needs.

As identified on Figure 3, the two key Stakeholder Roles involved are (through the «include» relationships from "Define an architectural framework for creating..."
architectural frameworks”): the Architectural Framework Sponsor and the Architectural Framework Modeller.

**Viewpoint Definition View**
The definition of the Viewpoint Context Viewpoint (VCV) is shown in the following diagram, a CAFF Viewpoint Definition View.

![Figure 36 - Viewpoint Definition View for the Viewpoint Context Viewpoint](image)

Figure 36 addresses the needs defined on Figure 35.

It defines the content of the Viewpoint Context Viewpoint, which defines the Context for a Viewpoint. It is made up of a Boundary, one or more Stakeholder Roles, that are outside the Boundary, and one or more Use Cases that are inside the Boundary.

Each Use Case represents a Viewpoint Concern (a type of Need) for a Viewpoint, that yields observable results to one or more Stakeholder Roles. Each Use Case may interact with a number of other Use Cases.

**Example View**
For an example of a Viewpoint Context View, see Figure 35. This is an example of a VCV that defines the context for itself. For an example without self-referentiality, see Figure 33.
A.6. Viewpoint Definition Viewpoint (VDV)

This section defines the Viewpoint Definition Viewpoint (VDV).

**Viewpoint Context View**

The needs that the Viewpoint Definition Viewpoint (VDV) is intended to address are shown in the following diagram, a CAFF Viewpoint Context View.

Figure 37 shows the Viewpoint Context View for the Viewpoint Definition Viewpoint. That is, it defines the needs that the Viewpoint Definition Viewpoint must address, together with relevant Stakeholder Roles.

The main need, taken from the Architectural Framework Context (see Figure 3) is to "Support definition of viewpoint content"; the Viewpoint Definition Viewpoint exists to define the contents of a Viewpoint. In order to do this, it is necessary to "Be consistent with needs of viewpoint". That is, the Viewpoint must be defined in such a way that it meets its needs, described on its associated Viewpoint Context Viewpoint.

The main needs that must be addressed are to:

- "Identify viewpoint elements" - Identify the Viewpoint Elements that will appear on the Viewpoint.
- "Identify viewpoint relationships" - Identify any relationships *between* the Viewpoint Elements that appear on the Viewpoint.

Both of these needs are constrained by the need to:

- "Conform to ontology" - Every Viewpoint Element (and relationship) that can appear on a Viewpoint *must* correspond to an Ontology Element (or..."
relationship) from the Ontology. Nothing can appear on a Viewpoint that does not exist on the Ontology.

As identified on Figure 3, the two key Stakeholder Roles involved in the main need to "Support definition of viewpoint content" are (through the «include» relationships from "Define an architectural framework for creating architectural frameworks"): the Architectural Framework Sponsor and the Architectural Framework Modeller. The Domain Expert role is also involved, given the constraint imposed by the need to "Conform to ontology".

**Viewpoint Definition View**
The definition of the Viewpoint Definition Viewpoint (VDV) is shown in the following diagram, itself a CAFF Viewpoint Definition View.

![Diagram of Viewpoint Definition View](image)

Figure 38 addresses the needs defined on Figure 37. It defines the content of the Viewpoint Definition Viewpoint.

The Viewpoint Definition Viewpoint defines a single Viewpoint, giving it a Name, an ID and a Description, and showing the Viewpoint Elements that make up the Viewpoint and the relationships between them. In most cases there will be a one-to-one mapping between Viewpoint Elements and the Ontology Elements that they correspond to and in such cases the same name is usually used so that a Viewpoint can, for all practical purposes, be considered to be made up of those Ontology Elements. However, this is not always the case; in some Viewpoints a single Ontology Element is represented on the Viewpoint by multiple Viewpoint Element. In these cases, the Viewpoint Definition Viewpoint shows the relationships between Viewpoint Elements and Ontology Elements.
This diagram has a subtlety that, at first viewing, may be missed; the Viewpoint Definition Viewpoint is its own View. To understand this consider the following: to define a Viewpoint, it is necessary to create an instance (i.e. a View) of the Viewpoint Definition Viewpoint for the Viewpoint being defined. That is, one creates a Viewpoint Definition View, that conforms to the Viewpoint Definition Viewpoint, for the Viewpoint being defined. Thus the Viewpoint Definition View created identifies the Viewpoint and the Viewpoint Elements that make it up.

- For example, Figure 32 - Viewpoint Definition View for the Ontology Definition Viewpoint, is a Viewpoint Definition View (a View), conforming to the Viewpoint Definition Viewpoint (a Viewpoint), that defines what can appear on the Ontology Definition Viewpoint.
- In a similar way, Figure 38 - Viewpoint Definition View for the Viewpoint Definition Viewpoint, is a Viewpoint Definition View (a View), conforming to the Viewpoint Definition Viewpoint (a Viewpoint), that defines what can appear on the Viewpoint Definition Viewpoint. Thus the View and the Viewpoint for the Viewpoint Definition Viewpoint are essentially one and the same; the Viewpoint Definition Viewpoint is self-referential. It defines itself and hence an instance (a View) is its own Viewpoint and vice versa.

**Example View**
For an example of a Viewpoint Definition View, see Figure 38. This is an example of a VDV that defines its own content. For an example without self-referentiality, see Figure 36.

**A.7. Rules Definition Viewpoint (RDV)**
This section defines the Rules Definition Viewpoint (RDV).

**Viewpoint Context View**
The needs that the Rules Definition Viewpoint (RDV) is intended to address are shown in the following diagram, a CAFF Viewpoint Context View.

Figure 39 shows the Viewpoint Context View for the Rules Definition Viewpoint. That is, it defines the needs that the Rules Definition Viewpoint must address, together with relevant Stakeholder Roles.
The main need, taken from the Architectural Framework Context (see Figure 3) is to "Support definition of architectural framework rules"; the Rules Definition Viewpoint exists to define any Rules that constrain the Architectural Framework. Note that such Rules can constrain any aspect of the Architectural Framework. The main needs that must be addressed are to:

- "Define rules" - Define any Rules which constrain the Architectural Framework.
- "Define relationships between rules" - Define any relationships between the Rules. This allows complex Rules to be built up.

As identified on Figure 3, the key Stakeholder Role involved in the main need to "Support definition of architectural framework rules" is (through the «include» relationships from "Define an architectural framework for creating architectural frameworks"): the Architectural Framework Sponsor.

**Viewpoint Definition View**

The definition of the Rules Definition Viewpoint (RDV) is shown in the following diagram, a CAFF Viewpoint Definition View.
Figure 40 addresses the needs defined on the Figure 39.

It defines the Rules that constrain the Architectural Framework. Each Rule has an ID and a Description. A Rule may be related to zero or more other Rules. Note that Rules constrain the entire Architectural Framework and hence can constrain any of the Viewpoints that make up an Architectural Framework.

**Example View**
For an example of a Rules Definition View see Figure 7.
Appendix B  Processes for AF Definition and Construction

This appendix presents the model for the architectural framework definition processes. These processes have been defined according to the ‘seven views’ approach to process modelling.

B.1 Requirements Context View

This section presents the requirements context view (RCV) for the architectural framework definition processes.

![Diagram of Requirements Context View](Figure 41 - Requirements context view (RCV) for the architectural framework definition processes)

The main need that must be fulfilled is to "Define an architectural framework for creating architectural frameworks", constrained by the need to "Comply with best practice" such as Architectural Framework Standards (for example, ISO42010). In order to "Define an architectural framework for creating architectural frameworks" it is necessary to:

- "Allow needs that the AF is to address to be captured" - When defining an Architectural Framework, it is important that the needs that the Architectural Framework is to address can be captured, in order to ensure that the Architectural Framework is fit for purpose.
- "Support definition of ontology for AF domain" - When defining an Architectural Framework, it is essential that the concepts, and the relationships between them, are defined for the domain in which the Architectural Framework is to be used. This is the Ontology that forms the foundational
basis of the definition of the Architectural Framework’s Viewpoints. Such an Ontology ensures the consistency of the Architectural Framework. The Architectural Framework must support such a definition of an Ontology.

- "Support identification of required viewpoints" - The Viewpoints that make up the Architectural Framework need to be identified. As well as supporting such an identification, the Architectural Framework must also "Support identification of relationships between viewpoints" and "Support identification of grouping of viewpoints into perspectives".

- "Support definition of viewpoint needs" - In order to define the Viewpoints that make up an Architectural Framework, it is essential that the needs of each Viewpoint be clearly understood in order to ensure each Viewpoint is fit for purpose and that the Viewpoints defined meet the overall needs for the Architectural Framework.

- "Support definition of viewpoint content" - An Architectural Framework is essentially a number of Viewpoints that conform to an Ontology. Therefore, when defining an Architectural Framework it is essential that each Viewpoint can be defined in a consistent fashion that ensures its conformance to the Ontology.

- "Support definition of architectural framework rules" - Often, when defining an Architectural Framework, it is often necessary to constrain aspects of the Architectural Framework through the definition of a number of constraining Rules. It is therefore essential that an AF Framework supports the definition of such Rules.

B.2 Stakeholder View

This section presents the stakeholder view (SV) for the architectural definition processes.

![Figure 42 - Stakeholder view (SV) for the architectural definition processes](image-url)
The key Stakeholder Roles involved are:

- 'Standard’ - the role of any appropriate standard for Architectural Frameworks. An example of a standard that could fill this role would be ISO42010.
- 'Reviewer' - this role is essential for all aspects of MBSE that covers both mechanical review (straightforward verification review that does not require any real human input but simply executes a pre-defined rule) and human reviews (that require reasoning and will tend to be qualitative and are typically very difficult, if not impossible to automate using a tool).
- ‘Configuration Manager’ - this role is responsible for ensuring that the model is correctly controlled, managed and configured.

B.3 Process Content View

This section presents the process content view (PCV) for the architectural framework definition processes.

The PCV for the architectural framework definition processes consistent of four processes, that are detailed as follows.

‘AF Definition Process’. The aim of this process is to understand the underlying need for the AF.

This process is made up of the following activities:

- 'Identify context’ - This activity identifies the context of the architectural framework that is under development. For example, some architecture frameworks, such as MODAF, DoDAF, etc. have been developed in an acquisition context, whereas others are more aimed at development, such as Zachman, etc.
- 'Identify source standard’ – This activity is concerned with identifying any source standards that the AF under development may need to comply with.
• 'Define AF context' - The context for the architecture framework is now created by invoking the 'Context Process' that will return the AF Context Viewpoint

• 'Define AF ontology' - Based on the AF Context Viewpoint, the ontology for the architecture framework under development is now defined by invoking the 'Ontology Definition Process' that returns the Ontology Definition Viewpoint

• 'Identify viewpoints' - Based on the Ontology and the AF Context, a number of Viewpoints are now identified along with the relationships between them. This results in the Viewpoint Definition Viewpoint.

• 'Review' - The review activity is concerned with the following:
  o AF context viewpoint, where the context is reviewed for consistency in terms of its associated source standards, and that the Use Cases pass a sanity check.
  o Ontology definition viewpoint, where the ontology and its associated elements are reviewed to ensure that each ontology element has provenance, that all ontology elements have at least one relationship to another elements, and that the ontology can pass a sanity check.
  o Viewpoint definition viewpoint, where each viewpoint that has been identified is checked against the underlying ontology and that each has relationships to at least one other viewpoint.
  o If the review is passed, then control is passed onto the 'baseline' activity, whereas if the review is failed, then control reverts back to the 'define AF context' activity

• 'Baseline' - All process artefacts: are recorded, are held under configuration control, form part of an overall baseline and are stored on the project repository

'Viewpoint Definition Process'. The main aim of this process is to identify the key viewpoints and to classify them into perspectives.

This process consists of the following activities:
• 'Select viewpoint' - This activity selects a single Viewpoint from the Viewpoint Definition Viewpoint that is to be used as basis for this process

• 'Define context' - The context for the selected viewpoint is now created by invoking the 'Context Process' that will return the Viewpoint Context Viewpoint

• 'Refine ontology elements' - based on the ontology definition viewpoint, a subset of the ontology is now identified that is judged to be relevant from the selected viewpoint

• 'Define viewpoint definition' - Based on the subset of the ontology that was identified in the previous process, the ontology for the viewpoint under development is now defined by invoking the 'Ontology Definition Process' that returns the Viewpoint Definition Viewpoint
• ‘Establish relationships’ - Any relationships that exist between the selected viewpoints and any others are identified at this point in the form of the viewpoint relationship viewpoint. These will be used for both a sanity check against the original viewpoint definition viewpoint and as an input for defining rules
• ‘Define rules’ - A set of rules for ensuring consistency of the viewpoints is defined
• ‘Review’ - The review activity is concerned with the following:
  o Viewpoint Context Viewpoint, where the context is reviewed for consistency with the AF context definition view, internal consistency and a sanity check is performed.
  o Viewpoint Definition Viewpoint, where the view is checked against the AF ontology for consistency and a sanity check is performed.
  o Viewpoint relationship viewpoint, where consistency against the AF viewpoint definition view is checked.
  o Rules Definition Viewpoint, where the rules are checked for consistency against the AF ontology and the viewpoint relationships view.
  o If the review is passed and there are no more viewpoints to be defined, then control passes to the 'baseline activity'. If the review is passed and there are more viewpoints to be defined, then control reverts back to the ‘select viewpoint’ activity. If the review fails, then control reverts back to the ‘define context’ activity
• ‘Baseline’ - All process artefacts: are recorded, are held under configuration control, form part of an overall baseline and are stored on the project repository

‘Ontology Definition Process’. The main aim of this process is to identify and define the main concepts and terms used for the AF in the form of an ontology.

This process consists of the following activities:
• ‘Identify concepts’ - This activity identifies a concept that is relevant to the AF under development and that needs to be defined
• ‘Define concepts’ - The concept that has been identified in the previous activity is now defined and its provenance is established against a source elements. This concept now becomes an ontology element
• ‘Define relationships with other concepts’ - The concept that has been identified and defined in the previous steps is now related to other concepts. These concepts and relationships now become part of the ontology
• ‘Create ontology definition viewpoint’ – The ontology is collected together to form the Ontology Definition Viewpoint.
• ‘Review’ - The review activity is concerned with the ontology, where each element is checked for provenance, that it relates to at least one other ontology element and that it is consistent with the overall AF ontology. If the review is passed, then control continues on to the ‘baseline’ activity. If the review is failed, then control reverts back to the ‘identify concept’ activity
• ‘Baseline’ - All process artefacts: are recorded, are held under configuration control, form part of an overall baseline and are stored on the project repository

‘Context Process’. The main aim of this process is to create a context that can be used to create either an ‘AF Context View’ or a ‘Viewpoint Context View’.

• ‘Select context’ - Based on the elements in the views, this activity identifies the source of the context. The context represents the needs of the system from a particular point of view and, therefore, must have a context source defined

• ‘Define context’ - This activity will produce a context that: identifies Use Cases based on the Needs, identifies the system boundary, identifies stakeholders that are external to this system boundary, identifies relationships between the Use Cases and the external stakeholders and identifies relationships between Use Cases

• ‘Analyse context’ – This activity considers the context and ensures that all Use Cases, stakeholders and their relationships are understood.

• ‘Resolve problems’ – This activity resolves any problems that may have been identified during the context analysis.

• ‘Review’ - The review activity is concerned with the Concern and Context that are generated by the process.

• ‘Baseline’ - All process artefacts: are recorded, are held under configuration control, form part of an overall baseline and are stored on the project repository

B.4 Process Behaviour Views

This section presents the process behaviour views (PBVs) for the architectural framework definition processes.

B.4.1 PBV – AF Definition Process
B.4.2 PBV – Viewpoint Definition Process

Figure 45 - PBV – Viewpoint Definition Process
B.4.3 PBV – Ontology Definition Process

Figure 46 - PBV – Ontology Definition Process

B.4.4 PBV – Context Process

Figure 47 - PBV – Context Process
B.5 Information Views

This section presents the information views for the architectural framework definition processes. For a full description of each artefact, see Section 2 and Appendix A of this document.

B.5.1 IV – AF Definition process

![Diagram](image)

Figure 48 - IV – AF Definition process

B.5.2 IV – Viewpoint Definition Process

![Diagram](image)

Figure 49 - IV – Viewpoint Definition Process
B.5.3 IV – Ontology Definition Process

Figure 50 - IV – Ontology Definition Process

B.5.4 IV – Context Process

Figure 51 - IV – Context Process