FP7-ICT-2013.3.4

Project acronym:	**AMADEOS**
Project full title:	Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems
Grant Agreement no.:	610535

Partners:

[1] Università degli Studi di Firenze

[2] Technische Universitaet Wien

[3] Université Grenoble Alpes

[4] ResilTech S.r.l.

[5] Thales Nederland Bv

[6] European Network For Cyber Security Cooperatief Ua

D2.3 – AMADEOS CONCEPTUAL MODEL – REVISED

(01.04.2015-30.09.2016)

Revision of the document:	1.1 (final)
Responsible partner:	TUW
Contributing partner(s):	all
Reviewing partner(s):	UJF, RES
Document date:	30/09/2016
Dissemination level:	PU

Change Records

Revision	Date	Changes	Authors	Status[1]
	08/06/2015	Content import from D2.2	TUW	NV
	07/10/2015	Minor additions to conceptual model concerning addressing the 2nd review recommendations.	TUW	NV
0.1	12/02/2016	Revision of conceptual model concerning interfaces and emergence.	TUW	NV
0.2	08/03/2016	Revision of Section 2.13.	TUW	NV
0.3	04/07/2016	Revision of Section 2.6 and added Annex A regarding emergence workshop. Revised Section 1.3 to limit scope of conceptual model to AMADEOS key perspectives.	TUW	NV
0.4	07/07/2016	Updated Section 4 for being aligned with the basic concepts of Section 2.	UNIFI	NV
0.5	25/07/2016	Updated Sections 2, 3, and 4.	TUW, UNIFI, UGA	NV
0.6	11/08/2016	Updated Section 2.13, Section 3.2, Section 4.	TUW	NV
0.7	16/08/2016	Minor updates to Sections 2.1.3, 2.2.3, 2.9, 2.11, 2.13.	TUW	NV
0.8	31/08/2016	Updated Section 4, Annex D. Finalized introduction, conclusion, and executive summary.	UNIFI, TUW	NV
	27/09/2016	Final refinements of a few interface, dynamicity and evolution concepts. Updated Section 4.	UNIFI, TUW	NV
1.0	28/09/2016	Finalized D2.3 for delivery to EC.	TUW	V
1.1	30/09/2016	Final editing for D2.3 submission	UNIFI	V

[1] V - Valid, NV - Not Valid, R - Revision

TABLE OF CONTENTS

CHANGES 2
1 INTRODUCTION 12
1.1 On the Nature of Concepts 13
1.2 Conceptual Modelling 14
1.3 Aim and Scope of this Document 15
1.4 Structure of the document 15
2 BASIC SOS CONCEPTS 17
2.1 Fundamental System Concepts 17
2.1.1 Basic Concepts 17
2.1.2 Systems 19
2.1.3 System-of-Systems 21
2.2 Time 22
2.2.1 Basic Concepts 23
2.2.2 Clocks 25
2.2.3 Time in an SoS 27
2.3 Data and State 29
2.3.1 Data and Information 29
2.3.2 State 31
2.4 Actions and Behavior 32
2.4.1 Actions 33
2.4.2 Behavior 35
2.5 Communication 35
2.5.1 Messages 36
2.5.2 Basic Transport Service 37
2.5.3 High-Level Protocols 40
2.5.4 Stigmergy 40
2.6 Emergence 41
2.6.1 Definition of Emergence 41
2.6.2 Explanation of Emergence 43
2.6.3 Supervenience 44
2.6.4 Downward Causation 45
2.6.5 Classification of Emergence in an SoS 45
2.6.6 Causes of Emergence In an SoS 46
2.7 Problem Solving 46
2.7.1 Basic Concepts 46
2.7.2 Problem Types 47

2.8 Dependability, Security and Criticality 48
2.8.1 Threats: Faults, Errors, and Failures 49
2.8.2 Dependability, Attributes, and Attaining Dependability 50
2.8.3 Security 51
2.8.4 Criticality 53
2.9 Evolution and Dynamicity 54
2.9.1 Basic concepts 54
2.9.2 Scenario-based Reasoning 55
2.10 System Design and Tools 57
2.10.1 Architecture 57
2.11 Governance 59
2.12 Quality Metrics and Attributes 60
2.12.1 Examples 61
2.13 Interfaces 62
2.13.1 Cyber-Physical Interfaces 64
2.13.2 Itom Interfaces 66
2.13.3 Service-based Interfaces 67
2.13.4 Interfaces of Constituent Systems integrated in System-of-Systems 68
3 CONCEPTUAL MODEL OF AN SOS 71
3.1 Model Overview 71
3.1.1 Introduction 71
3.1.2 Viewpoint of Structure 71
3.1.3 Viewpoint of Dynamicity 73
3.1.4 Viewpoint of Evolution 73
3.1.5 Viewpoint of Dependability and Security 75
3.1.6 Viewpoint of Time 77
3.1.7 Viewpoint of Multi-criticality 78
3.1.8 Viewpoint of Emergence 79
3.2 Relied Upon Interfaces 80
3.2.1 Information Transfer over RUIs 80
3.2.2 Cyber Space and Cyber Channels 81
3.2.3 Common Physical Environment and Stigmergic Channels 82
3.2.4 Dynamicity of RUIs 83
3.2.5 Evolution of RUIs 84
3.2.6 Monitoring and Dependability Considerations 84
4 SYSML REPRESENTATION 86
4.1 Introduction 86
4.1.1 A review on existing profiles for Systems of Systems 86

4.2 Mapping Rationale 88
4.3 Profile Definition 90
4.3.1 Structural components 90
4.3.2 Evolution and Dynamicity components 97
4.3.3 Dependability and Security components 102
4.3.4 Time component 106
4.3.5 Multi-criticality component 107
4.3.6 Emergence component 108
4.4 Example of profile application in MDE 111
4.5 Example of profile application in multi-agent based SoS design 111
4.5.1 Viewpoint of Dynamicity in multi-agent based SoS domain 112
4.5.2 Viewpoint of Emergence in multi-agent based SoS domain 114
5 CONCLUSION 116
REFERENCES 117
ANNEX A - EMERGENCE WORKSHOP 121
A.1 Introduction 122
A.2 Examples of Emergence 122
A.2.1 Emergence in Computer Systems 122
A.2.2 Emergence in Biological Systems 124
A.2.3 Emergence in Air Traffic Management (ATM) 125
A.3 Definition of Emergence 126
A.3.1 Philosophical View on Emergence 126
A.3.2 Definition for CPSoSs 128
A.4 Guidelines, Modeling and Detection Techniques 134
A.4.1 Detection Techniques in Air Traffic Management (ATM) 134
A.4.2 Modeling Emergence with Roles and Contexts 134
A.4.3 Engineering of Emergent Properties in Synthetic Biology 134
A.4.4 Component-based Representation of Emergent Behavior 135
A.4.5 Guidelines for CPSoS Design 136
A.5 Challenges in the Context of Emergence 138
A.6 Conclusion 138
A.7 References within Annex A 139
ANNEX B - SYSML PROFILE APPLICATION – ETHERNET CAPTURE EFFECT OF A LAN WITH CSMA-CD CONTENTION PROTOCOL 140
ANNEX C - TRACEABILITY MATRIX 144
ANNEX D - COMPARISON OF CONCEPTUAL MODEL WITH RELATED EU SOS CLUSTER PROJECTS 152
D.1 References within Annex D 154
ANNEX E - DEFINITION DIFFERENCES WRT D2.2 155

LIST OF FIGURES

Figure 1: The chain of threats: a fault in component A activates and generates an error; errors propagate until component A fails; the failure of A appears as an external fault to B [4]. 49
Figure 2: Interfaces of a Constituent System (CS). 69
Figure 3: Semantic relationships. Viewpoint of structure. 72
Figure 4: Semantic relationships. Viewpoint of Dynamicity. 73
Figure 5: Semi-formal model. Viewpoint of Evolution. 74
Figure 6: Semantic relationships. Viewpoint of Dependability. 75
Figure 7: Semantic relationships. Viewpoint of Security. 76
Figure 8: Semantic relationships. Viewpoint of Time. 77
Figure 9: Semantic relationships. Viewpoint of Multi-Criticality. 78
Figure 10: Semantic relationships. Viewpoint of Emergence. 79
Figure 11: Information Transfer over Relied Upon Interfaces in Systems-of-Systems. 81
Figure 12: Smart Grid case study 89
Figure 13: SoS Architecture package 92
Figure 14: Smart Grid Household Block Definition Diagram 93
Figure 15: SoS Communication package 95
Figure 16: Message exchange between constituent systems of a Smart Grid 96
Figure 17: SoS Interface Package 97
Figure 18: SoS Evolution package 98
Figure 19: Example of system evolution in a Smart Grid Household. 99
Figure 20: Smart Grid evolution modeling 99
Figure 21: SoS Dynamicity package 100
Figure 22: Dynamicity behavior of a Smart Grid 101
Figure 23: SoS Scenario-Based Reasoning (SBR) package 102
Figure 24: Dependability conceptual model. 104
Figure 25: SoS Security package. 105
Figure 26: SoS Time package. 107
Figure 27: Multi-criticality package. 108
Figure 28: SoS Emergence package. 109
Figure 29: Smart Grid Household – Emergent behavior. 110
Figure 30: Smart Grid Household SysML Model – Emergent Behavior description. 110
Figure 31: Internal representation of dynamicity viewpoint in CS agent model. 113
Figure 32: Sharing of Dynamicity viewpoint information between agents representing CSs. 114
Figure 33: Applying emergence models and observations to adapt local QoS negotiation behavior. 115
Figure 34: Ethernet capture effect 140
Figure 35: Ethernet Capture Effect SysML Model – Topology description 141

Figure 36: Ethernet Capture Effect – Emergent behavior 142
Figure 37: Ethernet Capture Effect SysML Model – Emergent Behavior description.....143

LIST OF TABLES

Table 1: Comparison of an SoS compared to a monolithic system 12
Table 2: Characteristics of Basic Transport Services.....40
Table 3: Classification of Emergent Behavior 46
Table 4: Viewpoint-based Literature Analysis of SysML Approaches to SoSs Design 87
Table 5: Mapping between the main concepts of Section 2 and the SoS profile's elements 144

Definitions and Acronyms

Acronym	Definition
BDD	Block Definition Diagram
CP	Consume/Produce
CPS	Cyber-Physical System
CS	Constituent System
DoW	Description of Work
ET	Event-Triggered
FMEA	Failure Mode and Effect Analysis
FTA	Fault Tree Analysis
GLONASS	Globalnaja nawigazionnaja sputnikowaja Sistema – Global Satellite Navigation System
GPS	Global Positioning System
GPSDO	GPS Disciplined Oscillator
HA	Hazard Analysis
IoD	Interval of Discourse
IoT	Internet of Things
LNAP	Local Network Access Point
MBSE	Model-Based System Engineering
MCDA	Multi-Criteria Decision Analysis
MDA	Model-Driven Architecture
NNAP	Neighbourhood Network
OODA	Observe, Orient, Decide and Act
PAR	Positive Acknowledge or Retransmission
PIM	Platform Independent Model
PSM	Platform Specific Model
RT	Real-Time
RUI	Relied Upon Interface
RUMI	Relied Upon Message Interface
RUPI	Relied Upon Physical Interface

RW	Read/Write
SBR	Scenario-Based Reasoning
SLA	Service Level Agreement
SLO	Service Level Objective
SOA	Service-Oriented Architecture
SoC	Sphere of Control
SoS	System-of-Systems
SysML	System Modelling Language
TAI	Temps Atomique International
TT	Time-Triggered
UoD	Universe of Discourse
UTC	Universal Time Coordinated
USAF	United States Air Force
WCET	Worst Case Execution Time
WP	Work Package

Executive Summary

This document contains the refined and final AMADEOS conceptual model that resulted from the joint work in WP2 achieved during M19-M36. A conceptual model establishes a domain specific ontology. Consequently it outlines a *domain specific vocabulary* and defines an *implicit theory* about a domain of discourse. The AMADEOS conceptual model is an explicit attempt to conceptualize the domain of Systems-of-Systems (SoSs). Such a conceptualization provides a language that enables the collaboration among SoS experts to discuss, analyse, design, and evolve SoSs. It is the intention of this deliverable to present the AMADEOS conceptual model such that it is available – as a whole or in parts – for SoS communities to better understand SoSs and further advance their research and exploitation.

In the previous versions of the conceptual model (cf. D2.1 and D2.2) concepts and their relationships have been defined and grouped into 13 distinct categories (e.g., fundamental SoS concepts, time, emergence, dependability and security) which resulted from the achievements of WP1 on SoS SOTA analysis and SoS cross-domain findings, and the first results of WP3 regarding the AMADEOS architecture.

Conceptual modelling is an iterative process that has been conducted in parallel to the other AMADEOS activities, particularly related to the definition of the AMADEOS architecture in WP3, the interactions with scientific communities and standardization efforts in WP5, the evaluation efforts in WP4, and taking EU and advisory board recommendations into account. While D2.2 is in full alignment with the results of WP3 and WP4, D2.3 even advances over D2.2 and contains a considerably more elaborated view on interfaces, emergence and evolution in SoSs, as well as minor refinements of concepts in order to reduce cognitive complexity across the whole conceptual model. These latest advancements are compatible extensions over D2.2, such that a consistent use of the now final AMADEOS conceptual model with the AMADEOS architecture, and the overall evaluation efforts remained possible.

The document starts by discussing the rationale for conceptual modeling of System-of-Systems (SoSs). Next, it contains a revision of the concepts and relationships from deliverables D2.1 and D2.2, including also a more detailed description on interfaces and emergence.

Starting from the former definitions, the document presents a graphical representation of concepts and their relationships organized in 7 different views namely *Structure*, *Dynamicity*, *Evolution*, *Dependability and Security*, *Time*, *Emergence* and *Multi-criticality*. By organizing the conceptual model through these views we focus on core AMADEOS issues by also leveraging the main expertise of AMADEOS partners. Following the graphical representation, the document connects all of the introduced viewpoints with a detailed, conceptual discussion on the *interface* between a Constituent System (CS) and its environment.

Further, the document discusses a SysML representation of the relationships of AMADEOS basic concepts. In particular, the document defines a SysML profile to represent the conceptual model of SoSs according to the 7 previously identified views. Modelling an SoS by using such a profile supports the understanding of critical SoS aspects (related to the views), and enables further analyses starting from a Platform Independent Model (PIM), as also highlighted by two applications of the profile.

Finally, the document contains a set of annexes presenting the results of the AMADEOS workshop on emergence, discussing a further SysML profile application regarding emergence, showing a traceability matrix among basic SoS concepts and the SysML profile, and a detailed comparison of AMADEOS concepts with related EU SoS cluster projects. In the annexes we also report on the differences of the conceptual model presented in this document and its previous version (i.e., deliverable D2.2).

1 INTRODUCTION

The report on *Software for Dependable Systems: Sufficient Evidence?* by the US National Academies [12] contains as one of its central recommendations*:*

One key to achieving dependability at reasonable cost is a serious and sustained commitment to simplicity, including simplicity of critical functions and simplicity in system interactions. ***This commitment is often the mark of true expertise.***

We feel that the first important step of achieving simplicity is the development of an understandable conceptual model of a domain. If the language used to talk about a domain contains terms with clearly defined meanings that are shared by a wide part of the community working in the domain, then the communication of ideas among experts is simplified. We hope that this AMADEOS conceptual model of an SoS contributes towards such a clarification.

An SoS comes about by the integration of existing systems (legacy systems), normally operated by different organizations and new systems that have been designed to take advantage of this integration. The following table compares some of the characteristics of an SoS compared to those of a monolithic system.

Table 1: Comparison of an SoS compared to a monolithic system

Characteristic	Monolithic System	System-of-System
Scope of the System	Fixed (known)	Not known
Clock Synchronization	Internal	External (e.g., GPS)
Structure	Hierarchical	Networked
Requirements and Spec.	Fixed	Changing
Evolution	Version Control	Uncoordinated
Testing	Test Phases	Continuous
Implementation Technology	Given and Fixed	Unknown
Faults (Physical, Design)	Exceptional	Normal
Control	Central	Autonomous
Emergence	Insignificant	Important
System Development	Process Model	???

If we look at this table we realize that many of the established assumptions in classical system design, such as e.g., the *scope of the system is known*, that *the design phase of a system is terminated by an acceptance test* or *that faults are exceptional events,* are not justified in an SoS environment.

SoS engineering is closely related to many other IT domains by looking at the *glue* that is needed to integrate the diverse systems:

- *Cyber-Physical Systems (CPSs)* and *embedded systems* that deal with the integration of systems that consist of a physical subsystem and a cyber subsystem.
- *Internet of Things (IoT)* that investigates the integration of an enormous variety of small smart sensors and large CPSs via the Internet.
- *Big Data* that looks at the analysis of the enormous, often heterogeneous data streams that are captured by the *IoT* and *SoSs*.

The concepts introduced in AMADEOS to describe the properties of an SoS can thus form a foundation of a conceptual model in these other domains as well.

The objective of the AMADEOS research is to bring time awareness and evolution into the design of System-of-Systems (SoS). The notion of time takes thus a prominent position in our understanding of an SoS and has influenced many of the presented concepts.

We start our work on the conceptual model of an SoS by looking at the nature of concepts and try to establish an agreement about the meaning of the term conceptual model.

1.1 ON THE NATURE OF CONCEPTS

In a changing world, *knowledge* about essential and permanent properties of objects and situations must be acquired and maintained since such knowledge is of critical importance for human survival. The continuous and immense flow of direct and indirect sensory data (eyes, ears, touch, smell, ...) to the human mind requires effective mechanisms to filter out those impressions that are considered relevant and must be stored in the brain to construct a useful *image* of the environment [13]. The most important of these mechanisms is the *mechanism of abstraction*. Abstraction is a process where the *characteristic particulars of an impression* are recognized and remembered to establish and identify a new category that is of later use in the construction of a mental model of reality.

Take the example of face recognition: Even a small child can remember, after a brief look at a person, the characteristic features of a face such that a person can be recognized again at a later time, even if the environmental conditions (lightning, line of sight) have changed. This very effective process of abstraction is carried out in the *intuitive experiential subsystem* of the human mind without any guidance by the *rational subsystem* [2, p.30].

Abstraction forms categories that are, considered in isolation, neutral. In order to establish a *utility* of a category, it must be linked with other categories. In the previous example, the *recognized person* can be linked with the category *friend* or *enemy*.

> ***Concept: A category that is augmented by a set of beliefs about its relations to other categories, i.e., existing knowledge, is called a concept.***

- The set of beliefs relates a *new concept* to already *existing concepts* and it provides an *implicit theory* (a mental model) of the domain.
- The theory explains how the individual interconnects the diverse concepts of the domain and *understands* their interrelationships.
- As a new domain is penetrated, new concepts are formed and linked to already existing concepts.

A new concept establishes a new *unit of thought* that lifts the mental processes to a *higher level of effectiveness. A new concept emerges and takes shape in the course of a complex operation aimed at the solution of some problem [17, p.54].* Problem solution and concept formation are thus closely related. In most cases, the formation of a new concept leads to an *abrupt simplification of a problem scenario*. In the history of the sciences, the formation and introduction of appropriate new concepts, such as *mass*, *acceleration,* etc., have been a necessary prerequisite for the formulation of novel natural laws and thus the advancement of science.

Normally a *name* (sound, string) or a *language phrase* —in scientific jargon— an *abbreviation,* is assigned to a new concept. The name can only be understood (i.e., it has a meaningful content to a user) *after* the concept has been formed.

A concept requires for its formation a number of experiences that have something in common:

- abstracting from familiar examples (*most important*) — starting with primary concepts (*ostensive* definition).

- prototype — an entity that depicts the typicality of the concept.
- feature specification (features may not always be *necessary* nor *sufficient*).
- establishing a set of *beliefs* and *relations* with already existing concepts (*concepts as theories*).
- precise definition in a language (*essential* definition).

Frequently a *precise definition* of a concept is not possible, since many concepts become fuzzy at their boundaries.

There is a *natural level of categorization*, neither too specific nor too general, that is widely used in thinking and conversation. This categorization leads to *basic-level concepts*. Basic level concepts are usually represented in the language by a *single* word. Take the example of the basic level concept *chair*, which is more concrete than *furniture*, but more abstract than *arm-chair*. Studies with children have shown that basic-level concepts are *acquired earlier* than *sub-concepts* or *encompassing* concepts.

Another classification of concepts distinguishes between *primary concepts* that are those directly derived from sensory experience (e.g., warm, cold) *and secondary concepts* that are those abstracted from other concepts, e.g., the example *furniture* from above. *Basic-Level* concepts are not necessarily *primary* concepts. By forming more and more abstract concepts on top of lower level concepts, a hierarchy of concepts can be constructed, leading to what [18, p.103] calls an *abstraction ladder*.

A new domain can be explained most effectively if the issues are treated at differing levels of the abstraction ladder. The starting point is a set of examples and observations based on already acquired concepts. The introduction of new, *more abstract concepts* must be based on generalization and abstraction from those already familiar existing concepts. Abstract analysis and concrete interpretation and explanation must be intertwined frequently. If one remains only at a *given low-level of abstraction,* then the amount of non-essential detail is overwhelming. If one remains only at a *given high-level of abstraction*, then relationships to the world as it is experienced by the senses are difficult to form.

The *knowledge base* of personal concepts that has been built up and is maintained by an individual in the *experiential* and *rational* subsystem of the mind over the lifetime of a human is called the *conceptual landscape (Weltbild)* [2, p.35]. This conceptual landscape is the result of genetic traits, sensory experience and communication. *Understanding a new concept* means that the properties of the *new concept* can be linked with already existing concepts in the *conceptual landscape* of the observing human. The firmer these links are, the better the understanding.

1.2 CONCEPTUAL MODELLING

According to [17], a conceptual model is characterized as follows:

The aim of a conceptual model is to express ***the meaning of terms and concepts*** *used by domain experts to discuss the problem, and to find the correct relationships between different concepts. The conceptual model attempts to clarify the meaning of various, usually ambiguous terms, and ensure that problems with different interpretations of the terms and concepts cannot occur. Such differing interpretations could easily cause confusion amongst stakeholders, especially those responsible for designing and implementing a solution, where the conceptual model provides a key artifact of business understanding and clarity. Once the domain concepts have been modeled, the model becomes a stable basis for subsequent development of applications in the domain.*

We discussed this characterization of a conceptual model at length within the AMADEOS project and came to the conclusion to support it fully. This characterization of a conceptual model, which is in agreement with wide opinions within our community, is thus forming a reference for the AMADEOS work on developing a conceptual model for an SoS.

A conceptual model establishes a domain specific ontology, since ontology *is a specification of the conceptualization of a domain of discourse.* It outlines a *domain specific vocabulary* and defines an *implicit theory* about a domain of discourse. An *ontological* commitment is the agreement to use the *shared vocabulary* in a coherent and consistent manner.

1.3 AIM AND SCOPE OF THIS DOCUMENT

This document proposes the conceptual model for AMADEOS serving as the basis for the other project related activities, in particular the definition of the AMADEOS Architectural Framework, its architectural building blocks and methodologies related to SoS dynamicity and evolution in WP3.

The AMADEOS conceptual model will focus on technical aspects of SoSs, in line with the AMADEOS key perspectives. Consequently, the concepts will only marginally relate to other important, but non-technical, issues regarding business, legal, or societal perspectives. However, the AMADEOS conceptual model allows the extension towards and the integration with these non-technical issues.

Starting from the definition of basic concepts and their relationships we aim at providing a two-fold formalization of the conceptual model. We will represent a non-formal graphical representation of the basic concepts and relationships grouped in seven different views which represent the key perspectives of AMADEOS, namely *Structure*, *Dynamicity*, *Evolution*, *Dependability and Security*, *Time*, *Emergence* and *Multi-criticality.* These views have been chosen in order to address the reviewers' comments of focusing on the key assets of AMADEOS. Beyond a non-formal graphical representation of the mentioned views, we will present a semi-formal representation of the conceptual model as a SysML profile composed of different components aiming at supporting the understanding and further analysis activities which can be carried out over instances of SoSs modeled through such a profile.

1.4 STRUCTURE OF THE DOCUMENT

Section 2 presents the basic concepts relevant for the description of an SoS. Definitions are structured along a set of high-level topics and they have been listed at the end of the document in alphabetical order along with a reference where they appear in the document. The definition of a concept is presented in ***italics bold*** while the discussion of the definitions is expressed in standard text.

As mentioned before, some circularity in the definitions is unavoidable, since some agreement on the meaning of words and phrases in the natural language must be assumed *a priori*. We have tried to keep this circularity to a minimum. We have also tried not to introduce formalisms whenever possible. In our opinion, formalism is only of value after a clear understanding of a concept has been gained by understanding the concept in its natural language presentation. Replacing fuzzy concepts by Greek letters does not improve the understanding.

Starting from the preliminary version of the conceptual model (D2.1) and its first full version (D2.2), the basic SoS concepts have been revised and in some cases added in order to align the conceptual model with the current activities of the other WPs of AMADEOS, and an additional study of the literature and discussions with experts. Further, we have considered the received EU recommendations and advisory board feedback by:

- including the physical part of the system into the conceptual model
- focusing on core AMADEOS issues, i.e., including multi-criticality definitions
- including design intents and goals (i.e., service)
- advancing and extending our concepts regarding interfaces, emergence and SoS evolution
- extending our research regarding emergence, to better understand emergent misbehavior and how to control it

- conducting an AMADEOS workshop on emergence in Cyber-Physical Systems-of-Systems (CPSoSs) and present its results
- clarifying definitions and use of the concepts law, constraint, and requirement
- presenting a comparison of main AMADEOS concepts with other FP7 Call 10 SoS cluster projects
- motivating the definition of our own SysML profile

The rest of the document is structured as follows. Section 3 builds upon the basic SoS concepts of Section 2 provides a high-level graphical representation of the conceptual model (divided according to the 7 different viewpoints) which explicitly shows the relationships among concepts. The section continues with a discussion of interfaces in SoSs which conceptually connect all relevant viewpoints. Section 4 presents a SysML profile for SoSs organized in packages whereas each package represent one of the 7 viewpoints. The applicability of the SysML profile is supported by several small examples. Section 5 concludes the document. ANNEX A contains a detailed report about the AMADEOS workshop on emergence in Cyber-Physical Systems of Systems (CPSoSs). ANNEX B gives another SysML profile application concerning emergence. ANNEX C contains a traceability matrix among basic SoS concepts and elements of the proposed SysML profile. ANNEX D contains a comparison of main AMADEOS concepts with other FP7 Call 10 SoS cluster projects. Details about changes in the conceptual definitions of the previous version of this deliverable (D2.2) can be found in ANNEX E

2 BASIC SOS CONCEPTS

2.1 FUNDAMENTAL SYSTEM CONCEPTS

Our philosophical view—termed *scientific realism* [18] — is committed to a *mind-independent world* that can be investigated by the sciences.

2.1.1 Basic Concepts

We start by delineating the universe and time of discourse of an *SoS*.

Universe of Discourse (UoD): The Universe of Discourse comprises the set of entities and the relations among the entities that are of interest when modeling the selected view of the world.

The word *domain* is often used as synonym to the notion *Universe of Discourse.*

Interval of Discourse (IoD): The Interval of Discourse specifies the time interval that is of interest when dealing with the selected view of the world.

In order to structure the *UoD* during the *IoD,* we must identify objects that have a distinct and self-contained existence.

Entity: Something that exists as a distinct and self-contained unit.

We distinguish between two very different kinds of entities, *things* and *constructs*.

Thing: A physical entity that has an identifiable existence in the physical world.

Referring to the *Three World Model of Popper* [19] there is another class of entities, we call them *constructs* that have no physical existence on their own but are products of the human mind.

Construct: A non-physical entity, a product of the human mind.

Examples of constructs are *an idea*, *a goal* or any other non-physical entity that is used in communication among humans to express a thought. In the *Three World Mode*l of Popper constructs are, as products of the human mind, in world three.

Humans can create entities—physical or non-physical—which are sometimes called artifacts.

Artifact: An entity that has been intentionally produced by a human for a certain purpose.

An entity can have one or more characteristic qualities that distinguish an entity from some other entities. We call such a characteristic quality an *attribute.*

Attribute: A characteristic quality of an entity.

Attributes are concepts that have been introduced in the history of science in order to be able to systematically characterize and investigate things. For example, a *thing*, e.g., a *stone*, can have the attribute of *mass*, *temperature*, *texture*, etc. The different characteristics that are of relevance when elaborating about an entity can be compiled in an attribute set.

The name of an entity can be viewed as a relation between an entity and its attribute set.

It is often useful to refine an attribute further and to introduce a value for an attribute.

Value: An element of the admissible value set of an attribute.

In many areas of science a *metric* has been established that makes it possible to assign a *numerical value* of *measurement units* to an attribute. For example, we can express the *mass of a stone* by a *numerical value* denoting kilograms or the *temperature of a stone* by a *numerical value* denoting degrees Celsius.

In other cases, such as *texture* or *colour,* a simple measurement unit to express this attribute on a linear numerical scale is either not available or not practical/common to be used. In these cases we can use words from a list of natural language expressions to describe the value of an attribute.

We call a valued attribute a property.

Property: A valued attribute.

If we say that a thing is *blue*, it means that the attribute *colour* of the thing has the value *blue*. This definition of a property follows the reasoning of Mealy [[48], p.526].

For example, the value set of the attribute *safeness* can be {*safe*, *unsafe*}. *Property checking* is an activity that determines if a property of a system, e.g., *safe*, is maintained during the operation of the system during the IoD.

A property can be viewed as a named-relation (the name is the name of the applicable attribute) between an entity and the value set of the property.

While an entity is a self-contained unit that persists over time, a property can change as time progresses (we will discuss the concept of *time* in more detail in Section 2.2).

Static Property: A property that does not change its value during the IoD.

Dynamic Property: A property that can change its value during the IoD.

When investigating SoSs we are primarily interested in dynamic properties. Since a dynamic property, e.g., *the traffic light has the colour red*, changes as time progresses, a proposition about the value of a dynamic attribute is only complete if the instant of the observation, i.e., a *timestamp,* is an atomic part of the proposition (the notion of a timestamp is discussed in Section 2.2).

Observation of an Entity: An atomic structure consisting of the name of the entity, the name and the value of the attribute (i.e., the property), and the timestamp denoting the instant of observation.

An observation is generated by a *sensor system* (cf. Section 2.3).

In *computer parlance*, an observation is often expressed in the form of a statement. Let us analyse the following statement:

Engine.temperature = 90

The variable name, *Engine.temperature* designates an attribute (temperature) of the identified entity *engine*. The value *90* specifies the value of the engine temperature. There are some elements missing from this observation: the specification of the unit for measuring the temperature and the timestamp of the observation. These missing elements are normally taken from the *context*.

Context: The set of cultural circumstances, conventions or facts, and the time that surround and have a possible influence on a particular thing, construct, event, situation, system, etc. in the UoD.

Contextual information, as it is sometimes called, is problematic in a large system which covers diverse cultural environments, since the contexts in different parts of the large systems may not be the same. Take the above example of *Engine.temperature* from Europe to the US. In Europe it is assumed that the temperature is measured in degrees *Celsius*, while in the US it is assumed that the temperature is measured in degrees *Fahrenheit*. The statement *Engine.temperature = 90* will thus have a different meaning, depending on the context.

Sphere of Control (SoC): The sphere of control of a system during an IoD is defined by the set of entities that are under the control of the system.

The sphere of control of a cyber system can extend beyond the boundaries of the cyber system. For example, in a cyber system that controls the traffic lights, the state of the traffic light is in the SoC of the controlling computer system.

> ***Sphere of Influence (SoI): The sphere of influence of a system during an IoD is defined by the set of entities that are under the influence of the system.***

2.1.2 Systems

We use the definition of the term *system* introduced in the EU Project DSOS (Dependable System-of-systems IST-1999-11585 [22, p.16]:

> ***System: An entity that is capable of interacting with its environment and may be sensitive to the progression of time.***

By 'sensitive to the progression of time' we mean the system may react differently, at different points in time, to the same pattern of input activity, and this difference is due to the progression of time. A simple example is a time-controlled heating system, where the temperature set-point depends on the current time [22, p.16]. The role of humans in a system is discussed at length below in this section.

> ***Environment of a System: The entities and their actions in the UoD that are not part of a system but have the capability to interact with the system.***

In *classical system engineering*, the first step in the analysis of a system is the establishment of an *exact boundary* between the system under investigation and its environment.

> ***System Boundary: A dividing line between two systems or between a system and its environment.***

In SoS Engineering, such an approach can be problematic, because in many SoS the system boundary is dynamic. Consider, e.g., a *car-to-car SoS* that consists of a *plurality of cars* cruising in an area. Where is the boundary of such an SoS? A good concept should be stable, i.e., its important properties, such as size, should remain fairly constant during the IoD. The *boundary* of the *car-to-car SoS* does not satisfy this requirement and is thus a *poor concept*. Our analysis of many other existing SoSs [21], e.g., the worldwide ATM system or a smart grid system came to a similar conclusion: It is hardly possible to define a stable boundary of an SoS.

In the above example of a *car-to-car SoS* each individual car in the system (consisting of the *mechanics of the car*, the *control system within the car* and the *driver*) can be considered as an *autonomous system* that tries to achieve its given objective without any control by another system.

> ***Autonomous System: A system that can provide its services without guidance by another system.***

Before starting with the detailed design of a large system an overall blueprint that establishes the framework of the evolving artifact should be developed.

> ***System Architecture: The blueprint of a design that establishes the overall structure, the major building blocks and the interactions among these major building blocks and the environment.***

Every organization that develops a system follows a set of explicit or implicit rules and conventions, e.g., naming conventions, representation of data (e.g, *endianness* of data), protocols etc. when designing the system.

> ***Architectural Style: The set of explicit or implicit rules and conventions that determine the structure and representation of the internals of a system, its data and protocols.***

Many of the existing legacy systems have been designed in the context of a single organization that follows its often ill-documented idiosyncratic architectural style. For example, undocumented implicit assumptions about the explanation of data can lead to mismatches when data is sent from one subsystem to another subsystem in an SoS.

> ***Monolithic System: A system is called monolithic if distinguishable services are not clearly separated in the implementation but are interwoven.***

Many systems are not monolithic wholes without any internal structure, but *are composed of interrelated parts, each of the latter being in turn hierarchic in structure until we reach some lowest level of elementary subsystem [24, p.184]*.

> ***Subsystem: A subordinate system that is a part of an encompassing system.***

We call the subsystems of a System of Systems (SoS) *Constituent Systems (CSs).*

> ***Constituent System (CS): An autonomous subsystem of an SoS, consisting of computer systems and possibly of controlled objects and/or human role players that interact to provide a given service.***

We introduce the concept of *service* in Section 2.4.

While the environment of a CS includes all entities within the UoD that may interact with the CS, we are still able to model the environment of a CS as another CS. This model of the environment can be used in simulations to, for example, test a CS with respect to its designed behavior, or determine calibration parameters.

The decomposition of a system into subsystems can be carried out until the internal structure of a subsystem is of no further interest.

The systems that form the lowest level of a considered hierarchy are called components.

> ***Component: A subsystem of a system, the internal structure of which is of no interest.***
>
> ***Legacy System: An existing operational system within an organization that provides an indispensable service to the organization.***
>
> ***Homogenous System: A system where all sub-systems adhere to the same architectural style.***

Many systems can be decomposed without loss into well-understood parts, so that the functions at the system level can be derived from the functions of the parts [50].

> ***Reducible System: A system where the sum of the parts makes the whole.***

If a system is designed by a single organization, then all subsystems will follow the same architectural style.

> ***Cyber-Physical System (CPS): A system consisting of a computer system (the cyber system), a controlled object (a physical system) and possibly of interacting humans.***

An *interacting human* can be a prime mover or role player.

> ***Prime mover: A human that interacts with the system according to his/her own goal.***

An example for a prime mover could be a legitimate user or a malicious user that uses the system for his/her own advantage. A human who is a prime mover can be considered to be a Constituent System (CS).

> ***Role player: A human that acts according to a given script during the execution of a system and could be replaced in principle by a cyber-physical system.***

A CPS is composed not only of the computer system, i.e., the cyber system, but also of a controlled object and possibly a human role player.

Entourage of a CPS: The entourage is composed of those entities of a CPS (e.g., the role playing human, controlled object) that are external to the cyber system of the CPS but are considered an integral part of the CPS.

In some CPSs, the boundary between the entourage and the environment of the CPS is dynamic, making it difficult to precisely define the scope of a CPS.

Many CPSs are Real-Time (RT) systems.

Real-time system: A computer system for which the correct results must be produced within time constraints.

Deadline: An instant when a computational action or a communication action must be completed.

We can classify deadlines according to the consequence of missing the deadline.

Hard Deadline: A deadline for a result is hard if a catastrophe can occur in case the deadline is missed.

Hard deadlines occur in hard real-time systems.

Firm Deadline: A deadline for a result is firm if the result has no utility after the deadline has passed.

Firm deadlines occur in Multimedia systems. The point of interaction of a system with another system or its environment is called an interface.

Interface: A point of interaction of a system with another system or with the system environment.

The services (see Section 2.4) that are provided by a system at an interface are described in the *interface service specification* (see Section 2.5).

In some models of physics, e.g., thermodynamics, the notion of a *closed system* is introduced in order to simplify the analysis of the system.

Closed System: A system that is not interacting with its environment during a given IoD.

A closed system is a concept that does not exist in the physical world. To contrast the notion of a *closed system*, the term *open system* has been coined.

Open System: A system that is interacting with its environment during the given IoD.

Evolutionary System: A system where the* interface *is dynamic (i.e., the service specification changes during the IoD).

Our AMADEOS analysis of the state of the art in SoS engineering has shown that many fielded SoSs are *evolutionary systems*. In an evolutionary system the external system boundary and the interface service specification are changing during the IoD.

2.1.3 System-of-Systems

We decided to select the following definition of Jamishidi as the starting point of our work [23].

System-of-Systems (SoS): An SoS is an integration of a finite number of constituent systems (CS) which are independent and operable, and which are networked together for a period of time to achieve a certain higher goal.

We consider the phrase *that are networked together for a period of time* an important part of this definition, since it denotes that a static scope of an SoS may not exist and the boundary between an SoS and its environment can be dynamic.

Human-Machine Interface (HMI) Component: A component of the CS that realizes the human-machine interface of a CS.

A browser to access services from the Internet is an example of an HMI Component.

Dahmann and Baldwin have introduced the following four categories of SoSs [24]:

Directed SoS: An SoS with a central managed purpose and central ownership of all CSs.

An example would be the set of control systems in an unmanned rocket.

Acknowledged SoS: Independent ownership of the CSs, but cooperative agreements among the owners to an aligned purpose.

Collaborative SoS: Voluntary interactions of independent CSs to achieve a goal that is beneficial to the individual CS.

Virtual SoS: Lack of central purpose and central alignment.

While a *directed SoS,* e.g., the CSs in an automobile that are under strict central management and ownership of a car company, comes close to a homogenous system, the other extreme, a *virtual CS*, lacks the elements of homogeneity and is formed by heterogeneous subsystems belonging to very different organizations.

We call an interface of a CS where the services of a CS are offered to other CSs a *Relied Upon Interface (RUI). It is ''relied upon'' w.r.t. the SoS,* since the service of the SoS as a whole relies on the services provided by the respective CSs across the RUIs.

Relied upon Interface (RUI): An interface of a CS where the services of the CS are offered to other CSs.

In addition to a *Relied upon Message Interface (RUMI)* where messages containing information are exchanged among the CSs, a Relied upon *Physical Interface (RUPI)* where things or energy are exchanged among the CSs can exist.

Relied upon Message Interface (RUMI): A message interface where the services of a CS are offered to the other CSs of an SoS.

Relied upon Physical Interface (RUPI): A physical interface where things or energy are exchanged among the CSs of an SoS.

Relied upon Service (RUS): (Part of) a Constituent System (CS) service that is offered at the Relied Upon Interface (RUI) of a service providing CS under a Service Level Agreement (SLA).

There may be other interfaces to systems external to a CS which we investigate in Section 2.5. The issues of *information exchange* and *interface specification* are the core topics of further sections of this report.

2.2 TIME

The focus of the previous Section was on the *static structure* of systems and their parts. In this Section we start being concerned with *change*. The concept of *change* depends on the progression of *time* that is one of the core topics that is investigated in AMADEOS. In an SoS a *global notion of time* is required in order to

- Enable the interpretation of timestamps in the different CSs.
- Limit the validity of real-time control data.
- Synchronize input and output actions across nodes.
- Provide conflict-free resource allocation.

- Perform prompt error detection.
- Strengthen security protocols.

We base our model of time on *Newtonian physics* and consider time as an independent variable that progresses on a dense time-line from the past into the future. For a deep discussion on the issues of time we refer to the excellent book by Withrow, *The Natural Philosophy of Time* [25] that considers also the revision to the Newtonian model by the theory of relativity. From the perspective of an SoS the *relativistic model of time* does not bring any new insights above those of the Newtonian model of time.

Time in biology is extensively treated in the book by Winfree, *The Geometry of Biological Time* [26].

2.2.1 Basic Concepts

In this Section we introduce some of the basic concepts of time that are needed when discussing the temporal properties of SoSs.

Time: A continuous measureable physical quantity in which events occur in a sequence proceeding from the past to the present to the future.

This definition of *time,* which denotes *physical time,* has been adapted from *Dictionary.com* and uses a number of fundamental concepts that cannot be defined without circularity.

Timeline: A dense line denoting the independent progression of time from the past to the future.

The directed time-line is often called the *arrow of time*. According to Newton, time progresses in dense (infinitesimal) fractions along the arrow of time from the past to the future.

Instant: A cut of the timeline.

There is a special instant on the timeline, the instant *now* that separates the past from the future.

Now: The instant that separates the past from the future.

Event: A happening at an instant.

An event is a happening that reports about some change of state at an instant.

Signal: An event that is used to convey information typically by prearrangement between the parties concerned.

Instants are totally ordered, while events are only partially ordered. More than one event can happen at the same instant.

Time code: A system of digital or analog symbols used in a specified format to convey time information i.e, date, time of day or time interval [49].

Temporal order: The temporal order of events is the order of events on the timeline.

Time Scale: A family of time codes for a particular timeline that provide an unambiguous time ordering (temporal order of events (adapted from [49]).

Causal order: A causal order among a set of events is an order that reflects the cause-effect relationships among the events.

Temporal order and causal order are related, but not identical. Temporal order of a cause event followed by an effect event is a necessary prerequisite of causal order, but causal order is more than temporal order [Kop11, p.53].

Interval: A section of the timeline between two instants.

While an interval denotes a section of the timeline between two instants, the duration informs about the length only, but not about the position of such a section.

Duration: The length of an interval.

The *length of an interval* can only be measured if a standard for the duration is available. The *physical SI second* is such an international standard (the *International System of Units* is abbreviated by *SI*).

Second: An internationally standardized time measurement unit where the duration of a second is defined as 9 192 631 770 periods of oscillation of a specified transition of the Cesium 133 atom.

The physical second is the same in all three important universal time standards, UTC, TAI and GPS time. UTC (*Universal Time Coordinated*) is an astronomical time standard that is aligned with the rotation of the earth. Since the rotational speed of the earth is not constant, it has been decided to base the SI second on atomic processes establishing the International Atomic Time *TAI (Temps Atomique International).* On January 1, 1958 at 00:00:00 TAI and UTC had the same value. The TAI standard is chronoscopic and maintained as the weighted average of the time kept by over 200 atomic clocks in over 50 national laboratories. TAI is distributed world-wide by the satellite navigation system GPS (*Global Positioning System*).

Offset of events: The offset of two events denotes the duration between two events and the position of the second event with respect to the first event on the timeline.

The position of an instant on a standardized timeline can only be specified if a starting point, the origin, for measuring the progression of time (in seconds) has been established.

Epoch: An instant on the timeline chosen as the origin for time-measurement.

GPS represents the progression of TAI time in weeks and full seconds within a week. The week count is restarted every 1024 weeks, i.e, after 19.6 years. The Epoch of the GPS signal started at 00:00:19 TAI on January 6, 1980 and again, after 1024 weeks at 00:00:19 TAI on August 22, 1999.

Cycle: A temporal sequence of significant events that, upon completion, arrives at a final state that is related to the initial state, from which the temporal sequence of significant events can be started again.

An example for a cycle is the rotation of a crankshaft in an automotive engine. Although the duration of the cycle changes, the sequence of the significant events during a cycle is always the same.

Period: A cycle marked by a constant duration between the related states at the start and the end of the cycle.

Periodic Systems are of utmost relevance in control applications

Periodic System: A system where the temporal behavior is structured into a sequence of periods.

Periodicity is not mandatory, but often assumed as it leads to simpler algorithms and more stable and secure systems [29, p.19-4].

Note that the difference between *cycle* and *period* is the constant duration of the period during the IoD.

Phase: A measure that increases linearly in each period from 0 degrees at the start until 360 degrees at the end of the period.

The phase is an important concept in periodic systems that exhibit a regular behavior and interact by the exchange of messages.

Phase alignment: The alignment of the phases between two periodic systems exhibiting the same period, such that a constant offset between the phases of the two systems is maintained.

A tight phase alignment between the computational and communication actions executed in different CSs minimizes the overall time of a transaction between a stimulus from the environment and a response to the environment.

Phase shift: An intentional or unintentional change in phase from a reference [49].

Phase jump: A sudden phase shift [49].

2.2.2 Clocks

Time is measured with clocks. In the cyber-domain, digital clocks are used.

Clock: A (digital) clock is an autonomous system that consists of an oscillator and a register. Whenever the oscillator completes a period, an event is generated that increments the register.

Oscillators and digital clocks are closely related. When looking at an oscillator, the form of the wave over the full cycle is of importance. When looking at a clock, only the distance between the events that denote the completion of cycles of the oscillator is of importance.

Syntonization: The process of setting multiple oscillators to the same frequency.

Although syntonized oscillators swing with the same frequency, they can differ in their phases. The syntonization of oscillators is important in telecommunication systems in order to properly extract the data from an incoming wave.

Nominal Frequency: The desired frequency of an oscillator [49].

Frequency drift: A systematic undesired change in frequency of an oscillator over time[49].

Frequency drift is due to ageing plus changes in the environment and other factors external to the oscillator.

Frequency offset: The frequency difference between a frequency value and the reference frequency value [49].

Stability: The stability of a clock is a measure that denotes the constancy of the oscillator frequency during the IoD.

The state of the register of a clock is often called the *state of clock*. The state of a clock remains constant during a complete period of the oscillator.

Wander: The long-term phase variations of the significant instants of a timing signal from their ideal position on the time-line (where long-term implies here that these variation of frequency are less than 10 Hz). (see also jitter) [49].

Jitter: The short-term phase variations of the significant instants of a timing signal from their ideal position on the time-line (where long-term implies here that these variation of frequency are greater than or equal to 10 Hz). (see also wander) [49].

The term timing signal can refer to a signal of a clock or of any other periodic event.

There exist other clocks, e.g., a *sun dial*, which is not digital in nature. The time resolution of every digital clock is limited by the duration of the period of the oscillator.

Tick: The event that increments the register is called the tick of the clock.

Granularity/Granule of a clock: The duration between two successive ticks of a clock is called the granularity of the clock or a granule of time.

The granularity of a clock can only be measured if another clock with a finer granularity is available.

We introduce a *reference clock* as a *working hypothesis* for measuring the instant of occurrence of an event of interest (such as, e.g., a clock tick) and make the following three hypothetical assumptions: (i) the reference clock has such a small granularity, e.g., a *femto second* (10^{-15} s), that digitalization errors can be neglected as second order effects, (ii) the reference clock can observe every event of interest without any delay and (iii) the state of the reference clock is always in perfect agreement with TAI time.

Reference clock: A hypothetical clock of a granularity smaller than any duration of interest and whose state is in agreement with TAI.

Coordinated Clock: A clock synchronized within stated limits to a reference clock that is spatially separated [49].

Every *good (fault-free) free-running clock* has an individual granularity that can deviate from the specified *nominal granularity* by an amount that is contained in the specification document of the physical clock under investigation.

Drift: The drift of a physical clock is a quality measure describing the frequency ratio between the physical clock and the reference clock.

Since the drift of a good clock is a number close to 1, it is conducive to introduce a drift rate by

Drift Rate = $|Drift - 1|$

Typical clocks have a drift rate of 10^{-4} to 10^{-8}. There exists no perfect clock with a drift rate of 0. The drift rate of a good clock will always stay in the interval contained in the specification document of the clock. If the drift rate of a clock leaves this specified interval, we say that the clock has *failed*.

Timestamp (of an event): The timestamp of an event is the state of a selected clock at the instant of event occurrence.

Note that a timestamp is always associated with a selected clock. If we use the reference clock for time-stamping, we call the time-stamp *absolute*.

Absolute Timestamp: An absolute timestamp of an event is the timestamp of this event that is generated by the reference clock.

If events are occurring close to each other, closer than the granularity of a digital clock, then an existing temporal order of the events cannot be established on the basis of the timestamps of the events.

If two events are *timestamped* by two different clocks, the temporal order of the events can be established on the basis of their timestamps only if the two clocks are synchronized.

Path Delay: The delay in the propagation of a signal between a source (input point) and a destination (output point).

In our model we assume, that the path delay between a clock and the reference clock has been compensated and can therefore be neglected.

Clock Ensemble: A collection of clocks, not necessary in the same physical location, operated together in a coordinated way either for mutual control of their individual properties or to maximize the performance (time accuracy and frequency stability) and availability of a time-scale derived from the ensemble [49].

Clock synchronization establishes a *global notion of time* in a clock ensemble. A global notion of time is required in an SoS if the timestamps generated in one CS must be interpreted in another

CS. *Global time* is an abstraction of physical time in a distributed computer system. It is approximated by a properly selected subset *of the ticks* of each synchronized local clock of an ensemble. A *selected tick* of a local *clock* is called a *tick* of the *global time*. For more information on the global notion of time see [2, pp. 58-64].

> ***Precision: The precision of an ensemble of synchronized clocks denotes the maximum offset of respective ticks of the global time of any two clocks of the ensemble over the IoD. The precision is expressed in the number of ticks of the reference clock.***

The precision of an ensemble of clocks is determined by the quality of the oscillators, by the frequency of synchronization, by the type of synchronization algorithm and by the jitter of the synchronization messages. Once the precision of the ensemble has been established, the granularity of the global time follows by applying the *reasonableness condition*.

> ***Reasonableness Condition: The reasonableness condition of clock synchronization states that the granularity of the global time must be larger than the precision of the ensemble of clocks.***

We distinguish between two types of clock synchronization, internal clock synchronization and external clock synchronization.

> ***Internal Clock Synchronization: The process of mutual synchronization of an ensemble of clocks in order to establish a global time with a bounded precision.***

There are a number of different internal synchronization algorithms, both non-fault tolerant or fault-tolerant, published in the literature (see, e.g, [28], and many others). These algorithms require the cooperation of all involved clocks.

> ***External Clock Synchronization: The synchronization of a clock with an external time base such as GPS.***

> ***Primary Clock: A clock whose rate corresponds to the adopted definition of the second. The primary clock achieves its specified accuracy independently of calibration.***

The term *master clock* is often used synonymously to the term *primary clock.*

If the clocks of an ensemble are externally synchronized, they are also internally synchronized with a precision of $|2A|$, where A is the *accuracy*.

> ***Accuracy: The accuracy of a clock denotes the maximum offset of a given clock from the external time reference during the IoD, measured by the reference clock.***

The external time reference can be a primary clock or the GPS time.

2.2.3 Time in an SoS

In an SoS, *external clock synchronization* is the preferred alternative to establish a global time, since the scope of an SoS is often ill defined and it is not possible to identify *a priori* all CSs that must be involved in the (internal) clock synchronization. A CS that does not share the global time established by a subset of the CSs cannot interpret the timestamps that are produced by this subset.

The preferred means of clock synchronization in an SoS is the external synchronization of the local clocks of the CSs with the standardized time signal distributed worldwide by satellite navigation systems, such as GPS, Galileo or GLONASS.

The GPS system, consisting at least of 24 active satellites transmit periodic time signals worldwide that are derived from satellite-local atomic clocks and seldom differ from each other by more than 20 ns [29]. A GPS receiver decodes the signals and calculates, based on the offset among the

signals, the position and time at the location of the GPS receiver. The *accuracy* of the GPS time is better than 100 ns.

In a recent report from the GAO to the US Congress [30] on *GPS-Disruption—Efforts to Assess Risks to Critical Infrastructure and Coordinate Agency Action Should be Enhanced* it is pointed out that many of the large infrastructure SoSs in the US are already *using GPS time synchronization* on a wide scale and a disruption of the GPS signals could have a catastrophic effect on the infrastructure. In this report it is noted that *a global notion of time is required in nearly all infrastructure SoSs,* such as telecommunication, transportation, energy, etc. and this essential requirement has been met by gradually using more and more often the time signals provided by GPS, not considering what consequences a disruption of the time distribution, either accidental or intentional, has on the overall availability and function of the infrastructure. The report proposes to systematically monitor the risks associated with a GPS failure and to plan mitigating actions for such a case.

The periodic time signal, generated by a GPS receiver, can be used to discipline a quartz oscillator.

> ***GPSDO (Global Positioning System Disciplined Oscillator): The GPSDO synchronizes its time signals with the information received from a GPS receiver.***

With a well-synchronized GPSDO a drift rate in the order 10^{-10} can be achieved.

> ***Holdover: The duration during which the local clock can maintain the required precision of the time without any input from the GPS.***

According to [31, p.62] a good GPSDO has deviated from GPS time by less than 100 µsec during the loss of GPS input of one week. As long as the GPS is operating and its input is available, a GPSDO can provide an accuracy of the global time of better than 100 nsec. If there is the requirement that, the free running global time must not deviate by more than 1µsec, a holdover of up to one hour is achievable using a good GPSDO.

The measurement of the position of an event on the timeline or of the duration between two events by a *digital global time* must take account of two types of unavoidable errors, the *synchronization error* caused by the finite precision of the global time and the *digitalization error* caused by the discrete time base. If the *reasonableness condition* is respected, the sum of these errors will be less than *2g,* where *g* is the granularity of the global time. It follows that the true duration between two events d_{true} lies in the following interval around the observed value d_{obs}.

$$(d_{obs}-2g) < d_{true} < (d_{obs}+2g)$$

The duration between events that are temporally ordered can be smaller than the granularity of a single clock. This situation is even worse if two different globally synchronized clocks observe the two different events. It is therefore impossible to establish the true temporal order of events in case the events are closer together than *2g*. This impossibility result can give rise to *inconsistencies* about the perceived temporal order of two events in distributed system.

These inconsistencies can be avoided, if a minimum distance between events is maintained, such that the temporal order of the events, derived from the timestamps of the events that have been generated by different clocks of a system with properly synchronized clocks is always the same.

> ***Sparse Time: A time-base in a distributed computer system where the physical time is partitioned into an infinite sequence of active and passive intervals.***

The active intervals can be enumerated by the sequence of natural numbers and this number can be assigned as the timestamp of an event occurring in an active interval. In order to establish consistency all events that occur in the same active interval of a sparse time are considered to have occurred simultaneously. This procedure establishes *consistency* at the price of *faithfulness*, since the temporal order of events that are closer together than the distance between sparse events is lost.

Sparse Events: Events that occur in the active interval of the sparse time.

Events that are in the SoC of a computer system with access to a global time, e.g., the start of sending a message, can be delayed until the next active interval and thus can be forced to be sparse events.

Non-Sparse Events: Events that occur in the passive interval of the sparse time.

Events that are outside the SoC of the computer system and are in the SoC of the environment cannot be forced to occur in the active intervals of the sparse time base, and can therefore be *non-sparse events*.

If all observers of a non-sparse event agree, by the execution of an agreement protocol, that the observed non-sparse event should be moved to the same nearest active interval of the sparse time base, then the consistency of these events can be established at the price of a further reduced faithfulness.

Time-aware SoS: A SoS is time-aware if its Constituent Systems (CSs) can use a global timebase in order to timely conduct output actions and consistently—within the whole SoS – establish the temporal order of observed events.

2.3 DATA AND STATE

Systems-of-Systems (SoSs) come about by the transfer of *information* of one Constituent System (CS) to another CS. But what is *information*? How is *information* related to *data*? After a thorough investigation of the literature about the fundamental concepts *data* and *information* it is concluded, that these terms are not well-defined in the domain of information science—see also the paper by C. Zins who asked a number of computer scientists about their meaning associated with the terms *data—information—knowledge* and published the divergent views reported to him in [31]. In this Section we will elaborate on the concepts of *data* and *information* along the lines of reasoning expressed in [32].

2.3.1 Data and Information

Let us start by defining the fundamental concepts of *data* and *information* first [32]:

Data: A data item is an artifact, a pattern, created for a specified purpose.

In cyber space, data is represented by a *bit-pattern*. In order to arrive at the meaning of the bit pattern, i.e., the *information* expressed by the bit pattern, we need an *explanation* that tells us how to interpret the given bit pattern.

Information: A proposition about the state of or an action in the world.

A proposition can be about *factual circumstances* or *plans*, i.e., schemes for action in the future. In any case, information belongs to the category of *constructs*, i.e., non-physical entities in world three of Popper [19]. Sometimes the phrase *semantic content* is used as a synonym for information.

Explanation: The explanation of the data establishes the links between data and already existing concepts in the mind of a human receiver or the rules for handling the data by a machine.

Since only the combination of *data* and an associated *explanation* can convey information, we form the new concept of an *Itom* that we consider the smallest unit that can carry *information*.

Itom: An Itom (Information Atom) is a tuple consisting of data and the associated explanation of the data.

The concept of an *Itom* is related to the concept of an *infon* introduced by Floridi [33]. However the properties we assign to an *Itom* are different from the properties Floridi assigns to an *infon*. The

concept of an Itom does not make any assumptions about the truthfulness of the semantic content, the information, in the Itom. We thus can attribute factual information as true information (*correspondence theory of truth* [34]), misinformation (accidentally false) or disinformation (intentionally false), or just call it information if we do not know yet if it is true or false. It is often the case that only some time after data has been acquired it can be decided whether the information conveyed by the data is true or false (e.g., consider the case of a value error of a sensor)

When data is intended for a *human receiver* then the explanation must describe the data using concepts that are *familiar* to the intended human receiver. When data is intended for processing by a *machine*, the *explanation* consists of two parts, we call them *computer instructions* and *explanation of purpose [32]*.

The *computer instructions* tell the computer system how the *data bit-string* is partitioned into syntactic chunks and how the syntactic chunks have to be stored, retrieved, and processed by the computer. This part of the *explanation* can thus be considered as a *machine program* for a (virtual) computer. Such a machine program is also represented by a bit-string. We call the data bit-string *object data* and the instruction bit-string that *explains* the object data, *meta data*.

Object Data: Data that is the object of description by meta data.

Meta Data: Data that describes the meaning of object data.

A computer Itom thus contains digital *object data* and digital *meta data*. The recursion stops when the *meta data* is a sequence of well-defined machine instructions for the *destined* computer. In this case, the *design of the computer* serves as an *explanation for the meaning of the data*.

The second part of the explanation of an Itom, the *explanation of purpose,* is directed to humans who are involved in the design and operation of the computer system, since the *notion of purpose is alien to a computer system*. The *explanation of purpose* is part of the documentation of the cyber system and must be expressed in a form that is *understandable* to the human user/designer.

To facilitate the exchange of information among heterogeneous computer systems in the Internet, markup languages, such as the Extensible Markup Language XML [35], that help to explain the meaning of data have been developed. Since in XML the explanation is separated from the data, the explanation can be adopted to the context of use of the data. Markup languages provide a mechanism to support an explanation of data. In many situations the explanation of the data is taken implicitly from the context.

Variable: A tuple consisting of data and a name, where the name points to the explanation of the data.

A variable can thus be considered a basic form of an Itom.

Depending whether the data is used as an input to a system or as an output from a system we distinguish between input data and output data.

Input data: Data that is used as an input to a system.

Output data: Data that is produced by a system.

Output data is published at an interface of the system to the environment.

Raw data: The bit pattern that is produced by a sensor system.

The meaning of raw data—the primary bit pattern—depends on the *elementary discriminatory capabilities of the sensor system.* Take the example of the *eye* or a *camera* that produce as primary data an image consisting of a given number of pixels of varying color and intensity. The characteristics of the image depend not only on the properties of the observed entity, but also on the *purpose* that determines the position and view of the camera and the instant of taking the snapshot. The purpose has thus an effect on the characteristics of the image i.e., the *acquired afferent data.*

Refined data: Data that has been created by a purposeful process from the raw data to simplify the explanation of the data in a given context.

Refined data abstracts from those properties of the raw data that are not relevant for the given purpose (but maybe relevant for another purpose). Refined data is the result of the process of *feature extraction* taking the raw date as input and looking at those features in the raw data that are relevant for the given purpose. By drastically reducing the size of (raw) data, refined data, sometimes called *meaningful data*, only retains those attributes of an observation that are considered relevant from the perspective of the user purpose. Those features that are not considered relevant in the process of feature extraction are not retained in the refined data (although they are present in the *raw data*).

Whereas the representation of the *raw data* is determined by the design of the sensor system, the representation of *refined data* depends on conventions that are determined by the context where the data is used.

When data is moved from one CS to another CS of an SoS, the context may change, implying that an explanation that is context-dependent changes as well. Take the example of *temperature* expressed as a *number*. In one context (e.g., Europe) the number is interpreted as degrees Celsius, while in another context (e.g, the US) the number is interpreted as degrees Fahrenheit. If we do not change the number (the data) then the meaning of the Itom is changed when moving the data from one context to another context. The neglected context sensitivity of data has caused accidents in SoSs [36].

An observation of a dynamic entity is only complete if the instant, the *timestamp* of making the observation, is recorded as part of the explanation.

The timestamp of input data is determined by the termination instant of the sensing process.

No timestamp is needed in the explanation when the properties of the observed entity are static. In the context of our work, we are mostly interested in dynamic properties of systems.

A periodic system often produces a stream of related data elements.

Data Frame: A valued data structure produced by a periodic system.

In a periodic system, the data contained in a given data frame is closely related to the data contained in the following data frame.

Data stream: A sequence of data frames produced by a periodic system.

The concept of data streams is important in most control systems. In many cases it is possible to mitigate the loss of a data frame of a data stream by replacing it by the contents of the previous frame.

2.3.2 State

Many systems store information about their interactions with the environment (since the start of a system with a clean memory) and use this information to influence their future behavior.

State: The state of a system at a given instant is the totality of the information from the past that can have an influence on the future behavior of a system.

A state is thus a valued data structure that characterizes the condition of a system at an instant. The concept of state is meaningless without a concept of time, since the distinction between past and future is only possible if the system is time-aware.

Stateless System: A system that does not contain state at a considered level of abstraction.

Statefull System: A system that contains state at a considered level of abstraction.

The variables that hold the stored state in a statefull system are called *state variables*.

State Variable: A variable that holds information about the state.

State Space: The state space of a system is formed by the totality of all possible values of the state variables during the IoD.

Instantaneous State Space: The state space of a system is formed by the totality of all possible values of the state variables at a given instant.

If we observe the progress of a system, we will recognize that the size of the instantaneous state space grows or shrinks as time passes. The instantaneous state space has a relative minimum at the start and end of an *atomic action* (see the Section 2.4).

The size of the instantaneous state space is important if we want to restart a system after a failure (e.g., a corruption of the state by a transient fault). We have to repair the corrupted instantaneous state before we can reuse the system. Generally, the smaller the instantaneous state space at the instant of reintegration, the easier it is to repair and restart a system.

Most control systems are cyclic or even periodic systems.

Ground State: At a given level of abstraction, the ground state of a cyclic system is a state at an instant when the size of the instantaneous state space is at a minimum relative to the sizes of the instantaneous state spaces at all other instants of the cycle.

We call the instant during the cycle of a cyclic system where the size of the instantaneous state has a minimum the *ground state instant*.

Ground State Instant: The instant of the ground state in a cyclic system.

At the ground state instant all information of the past that is considered relevant for the future behavior should be contained in a declared ground state data structure. At the ground state instant no *task* may be active and all communication channels are flushed. Ground state instants are ideal for reintegrating components that have failed.

Declared Ground State: A declared data structure that contains the relevant ground state of a given application at the ground state instant.

The *declared ground state* is essential for system recovery. The declared ground state contains only those ground state variables that are considered *relevant* by the designer for the future operation of the system in the given application. Other ground state variables are considered non-relevant because they have only a minor influence on the future operation of the system. The decision of whether an identified state variable is relevant or not relevant depends on a deep understanding of the dynamics of an application.

Concise State: The state of a system is considered concise if the size of the declared ground state is at most in the same order of magnitude as the size of the system's largest input message.

Many control systems have a concise state. There are other systems, such as data base systems that do not have a concise state—the size of the state of a data-base system can be Gigabytes.

In contrast to state variables that hold information about the state at an instant an event variable holds information about a *change* at an instant.

Event Variable: A variable that holds information about some change of state at an instant.

2.4 ACTIONS AND BEHAVIOR

We can observe the dynamics of a system that consists of discrete variables by an *event-based view* or by a *state-based view*.

In the *event-based view* we observe the state of relevant state variables at the beginning of the observation and then record all events (i.e. changes of the state variables) and the time of occurrence of the events in a trace. We can reconstruct the value of all state variables at any past instant of interest by the recorded trace. However, if the number of events that can happen is not bounded, the amount of data generated by the event-based view cannot be bounded.

In the *periodic state-based view* (called *sampling*), we observe the values of relevant state variables at selected *observation instants* (the sampling points) and record these values of the state variables in a trace. The duration between two observation instants puts a limit on the amount of data generated by the state-based view. However, the price for this limit is the loss of fidelity in the trace. Events that happen within a duration that is shorter than the duration between the equidistant observation instants may get lost.

Sampling: The observation of the value of relevant state variables at selected observation instants.

Most control systems use *sampling* to acquire information about the controlled object. The choice of the *duration between two observation instants*, called *the sampling interval,* is critical for acquiring a satisfying image of the controlled object. This issue is discussed extensively in the literature about control engineering [47].

2.4.1 Actions

In the following we introduce some concepts that allow us to describe the dynamics of a computer system.

Action: The execution of a program by a computer or a protocol by a communication system.

An action is started at a specified instant by a *start signal* and terminates at a specified instant with the production of an *end signal*.

Start signal: An event that causes the start of an action.

End signal: An event that is produced by the termination of an action.

Between the start signal and end signal an action is active.

Execution Time: The duration it takes to execute a specific action on a given computer.

The execution time is data dependent.

Worst Case Execution Time (WCET): The worst-case data independent execution time required to execute an action on a given computer.

There are two possible sources for a start signal of an action.

Time-triggered (TT) Action: An action where the start signal is derived from the progression of time.

An action can also be started by the completion of the previous action or by some other event (e.g., the push of a start button).

Event-triggered (ET) Action: An action where the start signal is derived from an event other than the progression of time.

We distinguish also between computational actions and communication actions.

Computational Action: An action that is characterized by the execution of a program by a machine.

Communication Action: An action that is characterized by the execution of a communication protocol by a communication system.

Communication actions will be discussed in more detail in Section 2.5.

In our model of an action we assume that at the start event an action reads input data and state. At the end event an action produces output data and a new state.

An action *reads* input data if the input data is still available after the action. An action *consumes* input data if the input data record is unavailable after the consumption by an action.

An action *writes* output data, if an old version of the output data is *overwritten* by the output data generated by the action. An action *produces* output data if a *new unit* of output data is generated by the action.

Input Action: An action that reads or consumes input data at an interface.

Output Action: An action that writes or produces output data at an interface.

We distinguish between actions that access the state of a system and those that do not.

Stateless Action: An action that produces output on the basis of input only and does not read, consume, write or produce state.

Statefull action: An action that reads, consumes, writes or produces state.

An action starts at the *start signal* and terminates by producing an *end signal*. In the interval <*start signal, end signal*> an action is *active*. While an action is active, the notion of state is undefined.

We can compose actions to form action sequences.

Action Sequence: A sequence of actions, where the end-signal of a preceding action acts as the start signal of a following action.

An *action sequence* is often called a *process*.

Activity Interval: The interval between the start signal and the end signal of an action or a sequence of related actions.

An action at a given level of abstraction, e.g., the execution of a program, can be decomposed into sub-actions. The decomposition ends when the internal behavior of a sub-action is of no concern.

Atomic Action: An atomic action is an action that has the all-or-nothing property. It either completes and delivers the intended result or does not have any effect on its environment.

Atomic actions are related to the notion of a component introduced above: neither the internals of components (from the point of view of structure) nor the internals of an atomic action (from the point of view of behavior) are of interest.

Irrevocable Action: An action that cannot be undone.

An irrevocable action has a lasting effect on the environment of a system. For example consider an output action that triggers an airbag in a car.

Idempotent Action: An action is idempotent if the effect of executing it more than once has the same effect as of executing it only once.

For example, the action *move the door to 45 degrees* is idempotent, while the action *move the door by five degrees* is not idempotent. Idempotent actions are of importance in the process of recovery after a failure.

We can combine computational action and communication actions to form a transaction.

Transaction: A related sequence of computational actions and communication actions.

In a control system the duration of the transaction that starts with the observation of the controlled object and terminates with the output of the result to an actuating device has an effect on the quality of control.

Transaction Activity Interval: The interval between the start signal and the end signal of a transaction.

Real-Time (RT) Transaction: A transaction that must complete before a specified deadline.

2.4.2 Behavior

The behavior of a system—the observable traces of activity at the system interfaces—is of utmost interest to a user.

Function: A function is a mapping of input data to output data.

Behavior: The timed sequence of the effects of input and output actions that can be observed at an interface of a system.

The effect of a *consuming input action* is the consumption of the input data record, while the effect of a reading input action does not change the input data and therefore has no observable effect. This is not true at the output side. Both, a *writing output action* and a *producing output action* have an observable effect.

Deterministic Behavior: A system behaves deterministically if, given an initial state at a defined instant and a set of future timed inputs, the future states, the values and instants of all future outputs are entailed.

A system may exhibit an intended behavior or it may demonstrate a behavior that is unintended (e.g. erroneous behavior).

Service: The intended behavior of a system.

The service specification must specify the intended behavior of a system. In non real-time systems, the service specification focuses on the data aspect of the behavior. In real-time systems the *precise temporal specification* of the service is an integral part of the specification.

Capability: Ability to perform a service or function.

2.5 COMMUNICATION

It is the basic objective of a communication system to transport a message from a sender to one or more receivers *within a given duration* and with a *high dependability*. By *high dependability* we mean that by the end of a specified time window the message should have arrived at the receivers with a high probability, the message is not corrupted, either by unintentional or intentional means, and that the security of the message (confidentiality, integrity, etc.) has not been compromised. In some environments e.g., the Internet of Things (IoT), there are other constraints on the message transport, such as, e.g., minimal energy consumption.

In an SoS the communication among the CSs by the exchange of messages is the core mechanism that realizes the integration of the CSs. It is imperative to elaborate on the concepts related to the message exchange with great care.

Since communication requires that diverse senders and receivers agree on the rules of the game, all involved partners must share these rules and their interpretation.

Communication Protocol: The set of rules that govern a communication action.

In the past fifty years hundreds of different communication protocols that often only differ in minor aspects, have been developed. Many of them are still in use in legacy systems. This diversity of protocols hinders the realization of the economies of scale by the semiconductor industry.

We distinguish between two classes of protocols: basic transport protocols and higher-level protocols. The basic transport protocols are concerned with the transport of a message from a sender to one or more receivers. The higher-level protocols build on these basic transport protocols to provide more sophisticated services.

2.5.1 Messages

Message: A data structure that is formed for the purpose of the timely exchange of information among computer systems.

We have introduced the word *timely* in this definition to highlight that a message combines concerns of the value domain and of the temporal domain in a single unit.

In the temporal domain, two important instants must be considered.

Send Instant: The instant when the first bit of a message leaves the sender.

Arrival Instant: The instant when the first bit of a message arrives at the receiver.

Receive Instant: The instant when the last bit of a message arrives at the receiver.

Transport Duration: The duration between the send instant and the receive instant.

Messages can be classified by the strictness of the temporal requirements.

In real-time communication systems strict deadlines that limit the transport duration must be met by the communication system.

From the point of view of the value domain, a message normally consists of three fields: a *header*, a *data field,* and a *trailer*. The header contains transport information that is relevant for the transport of the message by the communication system, such as the delivery address, priority information etc. The data field contains the payload of a message that, from the point of view of transport, is an *unstructured bit vector*. The trailer contains redundant information that allows the receiver to check whether the bit vector in the message has been corrupted during transport. Since a corrupted message is discarded, we can assume (as the fault model on a higher level) that the communication system delivers either a correct message or no message at all.

Payload of a Message: The bit pattern carried in the data field of the message.

The payload of a message consists of *data* that require an *explanation* to retrieve the information that is conveyed by the message. In many cases, the explanation is influenced by the context, as detailed in Section 2.3.1. In an SoS, where the context of the sender and the receiver are different, care must be taken that the *data* and the *explanation* are consistent in order to ensure that the information carried by the message is not distorted.

Apart from the transport information that is contained in the header of a message, the explanation of a message consists of two parts: the syntactic specification and the semantic specification.

Syntactic Specification: The specification that explains how the data field of a message is structured into syntactic units and assigns names to these syntactic units.

The fact that syntactic units are positioned next to each other in a single message implies that these syntactic units are related to each other.

The names designate the *concepts* that explain the meaning of the data. An Interface description language (IDL), such as the OMG IDL [37] can be used to express the syntactic specification

Semantic Specification: The specification that explains the meaning of the named syntactic units.

There are different means to establish the link between a message and the *explanation of the message*:

- *Message name*: The payload contains an a *priori agreed* message name in a defined position. The message name points to the explanation of the message.
- *Port name*: The explanation is linked to the port where the message arrives.
- *Receive instant*: The explanation is linked to the receive instant when a periodic message arrives.
- *Self-contained message:* The explanation is at an a priori specified position of the payload of the message.

In order that errors in a message can be detected, *assertions* can be associated with a message

Message assertion: A message assertion is a predicate on the values of a message and relevant state variables that defines an application specific acceptance criterion.

Using such assertions messages can be classified according to the following concept definitions. This classification has been taken from the DSOS project ([20], p19).

Valid Message: A message is valid if its checksum and contents are in agreement.

Checked Message: A message is checked at the source (or, in short, checked) if it passes the output assertion.

Permitted Message: A message is permitted with respect to a receiver if it passes the input assertion of that receiver. The input assertion should verify, at least, that the message is valid.

Timely Message: A message is timely if it is in agreement with the temporal specification.

Value Correct Message: A message is value correct if it is in agreement with the value specification.

Correct Message: A message is correct if it is both timely and value correct.

Insidious Message: A message is insidious if it is permitted but incorrect.

2.5.2 Basic Transport Service

A basic transport service transfers a message from a sender to one or more receivers. We limit our discussion to three different transport protocol classes that are representative for a number of important protocols in each class:

- Datagram
- PAR Message
- TT Message

Datagram: A best effort message transport service for the transmission of sporadic messages from a sender to one or many receivers.

A datagram is a very simple transport service. A datagram is forwarded along the best available route from a sender to one or a number of receivers. Every datagram is considered independent from any other datagram. It follows that a sequence of datagrams can be reordered by the communication system. If a datagram is corrupted or lost, no error will be indicated.

PAR-Message: A PAR-Message (Positive Acknowledgment or Retransmission) is an error controlled transport service for the transmission of sporadic messages from a sender to a single receiver.

In *the positive acknowledgment-or-retransmission* (PAR) protocol a sender waits for a given time until it has received a positive acknowledgement message from the receiver indicating that the previous message has arrived correctly. In case the timeout elapses before the acknowledgement message arrives at the sender, the original message is retransmitted. This procedure is repeated n-times (protocol specific) before a permanent failure of the communication is reported to the high-level sender. The jitter of the PAR protocol is substantial, since in most cases the first try will be successful, while in a few cases the message will arrive after *n* times the timeout value plus the worst-case message transport latency. Since the timeout value must be longer than two worst-case message transport latencies (one for the original message and one for the acknowledgment) the jitter of PAR is longer than **(*2n*)** worst-case message-transport latencies [2, p.169]. In addition to this basic PAR protocol one can find many protocol variants that refine this basic PAR protocol.

TT-Message: A TT-Message (Time-Triggered) is an error controlled transport service for the transmission of periodic messages from a sender to many receivers.

A time-triggered message is a periodic message that is transported from one sender to one or more receivers according to a pre-planned schedule. Since it is known *a priori* at the sender, the receiver and the communication system when a time-triggered message is expected to arrive, it is possible to avoid conflicts and realize a tight phase alignment (see Section 2.2.1) between an incoming and an outgoing message in a communication system switch. The error detection of a TT-message is performed by the receiver on the basis of his a priori knowledge about the expected arrival time of a message. The error detection latency is determined by the precision of the global time.

We will investigate these three basic transport protocols from the following points of view

- Data/Control Flow and Error Detection
- Message Handling
- Transport Duration

and present a summary of our investigation at the end of this Section.

2.5.2.1 DATA/CONTROL FLOW AND ERROR DETECTION

A message is both, a container for delivering a payload and a carrier of control signals.

Data flow: The flow of the payload data of a message from a sender to the receivers.

The *data flow view* only looks at the flow of payload data and is not concerned with the control signals associated with a message transmission.

Control flow: The flow of control signals when executing a protocol.

In the datagram and TT messages, *data flow* and *control flow* are uni-directional. In a PAR message, the data flow is *uni-directional* while the control flow is *bi-directional*. The bi-directional control flow in the PAR message is needed to implement flow control and error detection.

Flow control: The control of the flow of messages from the sender to the receiver such that the sender does not outpace the receiver.

We distinguish between two types of flow control, explicit flow control and implicit flow control.

Explicit flow control: After having sent a message, the sender receives a control message from the receiver informing the sender that the receiver has processed the sent message.

In a PAR message, the sender uses the explicit flow control message also to perform err*or detection by the sender*. By associating a time-out with the instant of arrival of the explicit control

message the sender can detect a communication error. The error detection latency in PAR is relatively long, since an error will only be reported to a client after all retries to send the message have failed.

> ***Implicit flow control: The sender and receiver agree a priori on a maximum send rate. The sender commits to never send messages faster than the agreed send rate and the receiver commits to accept all messages that the sender has sent.***

The TT message uses implicit flow control based on the a priori knowledge of the send schedule by the sender and receiver. The *error detection* of a TT-message is performed by the *receiver* based on his a priori knowledge of the expected arrival times of the periodic message. The error detection latency of a TT message is determined by the precision of the global time.

2.5.2.2 MESSAGE HANDLING

Communication systems differ in the way they handle the messages at the sender and at the receiver. We distinguish between the *consume/produce* and *read/write* paradigm for handling messages at the sender and receiver.

> ***Consume/Produce (CP) paradigm: At the sender, the communication system consumes the message from a sender queue and at the receiver the communication system adds the received message to a receiver queue.***

The CP paradigm requires a queue management at the sender and at the receiver and must deal with all cases of *queue emptiness* and *queue overflow*. This is called back-pressure flow control.

> ***Back-pressure flow control: A message control schema, where the receiver exerts back pressure on the sender to ensure that the speed of the sender will not outpace the speed of the receiver and thus lead to queue overflow.***
>
> ***Read/Write (RW) paradigm: At the sender the communication system reads the contents of the message from a message variable and at the receiver the communication system writes the arriving message into a message variable, overwriting the old content of the message variable.***

In the RW paradigm no queue management and back-pressure flow control is required but a message that is not read by the receiving process before the next message arrives is lost. In real-time systems, the RW transport schema, that always presents the most recent version of a message to the receiving process, is widely used.

2.5.2.3 TRANSPORT DURATION AND JITTER

In real-time systems the duration and the jitter of a message from a sender to a receiver is of utmost importance.

> ***Jitter of a Message: The duration between the minimal transport duration and the maximum transport duration.***

While the transport duration of a datagram is a priori unknown, the transport duration of a PAR message is bounded, but the jitter of a PAR message might be significant.

A TT message has a bounded transport duration and a small jitter that is determined by the precision of the clock synchronization.

2.5.2.4 SUMMARY

Table 2 presents the summary of our analysis.

Table 2: Characteristics of Basic Transport Services

Characteristic	Datagram	PAR-Message	TT- Message
Send Instants	Sporadic	Sporadic	Periodic
Data/Control Flow	Uni-directional	Bi-directional	Uni-directional
Flow Control	None	Explicit	Implicit
Message Handling	R/W or C/P	C/P	R/W
Transport Duration	a priori unknown	Upper limit known	Tight limit known
Jitter of the Message	Unknown	large	small
Temporal Error Detection	None	At Sender	At Receiver
Example	UDP	TCP/IP	TT-Ethernet

Although the basic datagram service does not provide temporal error detection, a posteriori error detection of datagram messages can be achieved by putting the send timestamp in the message, given that synchronized clocks are available.

It is up to the application to decide which basic transport protocol is most appropriate to integrate the CSs into an SoS.

2.5.3 High-Level Protocols

The transport protocols form the basis for the design of higher-level protocols, such as protocols for *file transmission* and *file sharing*, *device detection* and numerous other tasks. It is beyond the scope of this conceptual model to discuss the diversity of higher-level protocols. In the latter part of the AMADEOS project we will look at some of the higher level protocols that are of particular relevance in SoSs.

However, it must be considered that the temporal properties of the basic transport protocol determine to a significant extent the temporal properties of the high-level protocols.

2.5.4 Stigmergy

Constituent systems (CSs) that form the autonomous subsystems of Systems-of-Systems (SoSs) can exchange information items via two different types of channels: the conventional communication channels for the transport of messages and the *stigmergic channels* that transport information via the change and observation of states in the environment. The characteristics of the stigmergic channels, which often close the missing link in a control loop can have a decisive influence on the system-level behavior of an SoS and the appearance of emergent phenomena [51].

> ***Stigmergy: Stigmergy is a mechanism of indirect coordination between agents or actions. The principle is that the trace left in the environment by an action stimulates the performance of a next action, by the same or a different agent.***

The concept of *stigmergy* has been first introduced in the field of biology to capture the indirect information flow among ants working together [53] [54]. Whenever an ant builds or follows a trail, it deposits a greater or lesser amount of pheromone on the trail, depending on whether it has successfully found a prey or not. Due to positive feedback, successful trails—i.e., trails that lead to an abundant supply of prey—end up with a high concentration of pheromone. The running speed of the ants on a trail is a non-linear function of the trail-pheromone concentration. Since the trail-pheromone evaporates—we call this process *environmental dynamics*—unused trails disappear autonomously as time progresses.

> ***Environmental Dynamics: Autonomous environmental processes that cause a change of state variables in the physical environment.***

The impact of environmental dynamics on the stigmergic information ensures that the captured itoms are well-aligned with the current state of the physical environment. No such alignment takes place if itoms are transported on cyber channels.

> ***Stigmergic Information Flow: The information flow between a sending CS and a receiving CS where the sending CS initiates a state change in the environment and the receiving CS observes the new state of the environment.***

If the output action of the sender and the input action of the receiver are closing a stigmergic link of a control loop, then the synchronization of the respective output and input actions and the transfer function of the physical object in the environment determine the delay of the stigmergic link and are thus of importance for the performance and stability of the control loop. The synchronization of the respective output and input actions requires the availability of a global time base of known precision.

2.6 EMERGENCE

Systems-of-Systems (SoSs) are built to realize novel services that go beyond the services that can be provided by any of the Constituent Systems (CSs) in isolation. We call these novel services *emergent services.* The concept of emergence is thus at the core of SoS engineering and warrants an in-depth analysis of the topic.

Take the example of the worldwide ATM system. A local ATM system gives a client access to his financial assets via an ATM terminal in the form of the local currency at the home base. The worldwide ATM system realizes a novel service: the access of financial assets of a client from any ATM terminal in the world in a currency that is determined by the local context of the foreign ATM terminal. This novel emergent service cannot be provided by any ATM system in isolation but comes about by the worldwide integration of all ATM terminals. Incidentally, this example is also a good illustration of the concept of an *itom* introduced in Section 2.3.1. The *same value* of a financial transaction is represented in different countries (contexts) by the local currency of the country, implying that the data (the amount of money in the local currency) and the representation (the local currency) are different, while the *semantic content*, the value of the transaction, remains the same.

The study of the topic of emergence, how novel phenomena emanate into the world by the interaction and integration of lower-level components, has been an important subject of research in the domains of the natural sciences and philosophy in the long history of the sciences. However, we could not find a generally accepted explanation of the concept of emergence, although there are some excellent publications devoted to this topic, e.g. [38][39][40][41][42]. We align the concepts of emergence introduced in the AMADEOS conceptual model of an SoS with the concepts published in the general literature about emergence.

In March 2016, the AMADEOS project organized an interdisciplinary workshop on emergence. The results of this workshop are documented in a separate technical report [72] and in ANNEX A .

2.6.1 Definition of Emergence

In his seminal paper *The Architecture of Complexity* H. Simon discusses the central role that hierarchies play in models of complex systems [22]. Related primitive elements at the bottom of a hierarchy are clustered together to form a *whole* for the next higher level. This process of agglomeration is repeated recursively until the top of the hierarchy the *apex*, is reached. The apex is determined by the purpose of the model.

In the following Sections we focus our attention on two adjacent levels of a multi-level hierarchy, a lower level, also called the *micro-level* and the adjacent upper level (the level above) also called the *macro-level*.

Simon distinguishes between two different types of hierarchies, a *formal hierarchy* and a *non-formal hierarchy* or, often called, a *holarchy* [43]. In a formal hierarchy each subsystem at the

micro-level is linked vertically to its controlling system at the macro-level by a *reporting and control relation*. *More exactly, in a hierarchic formal organization, each system consists of a "boss" and a set of subordinate subsystems* [22]. In a formal hierarchy there are no direct horizontal interactions among the subsystems at the micro level. An example for a formal hierarchy is the command structure of a military organization.

In a *non-formal hierarchy* or *holarchy,* there are *horizontal interactions* among the related sub-systems at the micro-level that lead to the formation of a *whole* with its own characteristic properties at the macro-level. Koestler [43] introduces the notion of a *holon*, a two-faced subsystem as the central concept in a non-formal hierarchy.

> ***Holon: A two-faced entity in a non-formal hierarchy that acts externally at the macro-level as a whole while it is established internally by the interactions of its parts at the micro-level.***

Viewed from the macro-level, a holon is thus an *autonomous unit* that can interact with other holons of that level, but viewed from the micro-level, a holon is an *isolated ensemble of parts* that interact among each other.

According to Koestler holons form the basic building blocks of holarchies.

> ***Holarchy: A structure where holons at one level interact horizontally to form a novel holon at the next higher level.***

An example of a holarchy is the hierarchic model of the universe, starting from subatomic elements up to the appearance of the human species, where higher-level subsystems emerge out of the interaction and organization of the primitive lower level physical entities.

It is generally agreed that a definition of emergence must focus on two adjacent levels of a hierarchic system. In an SoS context, the micro-level is the level of the CSs, while the macro-level is the level of the SoS. We introduce the following definition of emergence:

> ***Emergence: A phenomenon of a whole at the macro-level is emergent if and only if it is of a new kind with respect to the non-relational phenomena of any of its proper parts at the micro level.***

A phenomenon is of *a new kind* if the concepts required to communicate this phenomenon cannot be found in the world of the isolated parts. *Conceptual Novelty* is thus the landmark of our definition of emergence.

Note that, according to the above definition, the emergent phenomena *must only be of a new kind with respect to the non-relational phenomena of the parts*, not with respect to the knowledge of the observer. There are some publications that relate the appearance of emergence to a *surprise* by the observer. Such a view of emergence would mean that the concept is relative to the epistemic state of the observer. One observer, who knows about this phenomenon is *not surprised*, and another one, who sees the phenomenon for the first time is *surprised*. We do not feel that such a relative definition of the concept of emergence is useful.

If a phenomenon of a whole at the macro-level is *not of a new kind* with respect to the non-relational phenomena of any of its proper parts at the micro level then we call this phenomenon *resultant*.

> ***Resultant phenomenon: A phenomenon at the macro-level is resultant if it can be reduced to a sum of phenomena at the micro-level.***

The essence for the occurrence of emergent phenomena at the macro-level lies in the *organization of the parts*, i.e., in the spatial arrangement and/or the physical or informational interactions among the parts at the micro-level. Although this definition requires that the emergent phenomenon does not appear at any level below the macro (SoS) level, i.e., emergent phenomena are *systemic,* the definition is extensive and includes many phenomena that are familiar to us from our every-day experience.

An emergent phenomenon of a system that occurs at the macro-level can be related to *structure* or *behavior*. In an SoS context we are primarily interested in *emergent behavior*. In most cases the emergent behavior is *diachronic*, i.e., it develops not spontaneously, but over a time interval.

2.6.2 Explanation of Emergence

According to the above definition, a novel phenomenon is considered *emergent*, irrespective of whether it can be *explained* how the new phenomenon at the macro level has developed out of the properties of the parts at the micro-level.

Given the *present state of knowledge*, some of these emergent phenomena can be explained by existing theories while there are other emergent phenomena where at present no full explanation can be given as to how they developed.

But what constitutes a *proper explanation*?

Hempel and Oppenheim [73] outlined a general schema for a scientific explanation of a phenomenon as follows:

> **Given**
>
> *Statements of antecedent conditions*
>
> **and**
>
> *General Laws*
>
> **then a logical deduction of the**
>
> *Description of the empirical phenomenon to be explained*
>
> **is entailed.**

The *antecedent conditions* can be initial conditions or boundary conditions that are *unconstrained* by the general laws.

> ***General Law: An inevitable consequence that follows if a set of antecedent conditions applies.***

The *general laws* can be either universally valid *natural laws* that reign over the behavior of things or *logical laws* describing a valid judgment in the domain of constructs. Natural laws do not change in time or have a memory of the past. A natural law, such as a physical law, must hold everywhere, no matter what level of a multi-level hierarchy is the focus of the investigations.

A weaker form of explanation is provided if the *general laws* in the above schema are replaced by *established rules.*

> ***Established Rule: An observed consequence that often follows if a set of antecedent conditions applies.***

There are fundamental differences between general laws and established rules. General laws are *inexorable* and *universally* valid while established rules are *structure dependent* and *local* [All82, p. 42]. Rules about the behavior of things are based on more or less meticulous experimental observations. A special case is the introduction of *imposed rules*, e.g., the rules of an artificial game, such as chess. The degree of accuracy and rigor of various established rules differ substantially.

It thus follows that between the two extremes of *scientifically explained* and *not explained at all* there is a *continuum of explanations* that are more or less acceptable and are relative with respect to the general state of knowledge and the opinion of the observer at a given point in time.

It is often the case that novel concepts have to be formed to capture the emergent phenomena at the macro level and new laws, called *intra-ordinal laws* [44], have to be introduced to be able to describe the emerging phenomena at the macro level appropriately.

> ***Intra-ordinal Law: A new law that deals with the emerging phenomena at the macro level.***

These phenomena and laws are either absent at the level below or are simply meaningless at a lower level [47, p.36]. Intra-ordinal laws of an *axiomatic nature* have an explanatory power for phenomena at *all levels above* the macro level for which they have been introduced. Whether a newly formulated intra-ordinal law is of an axiomatic nature, i.e. is a universally accepted principle, is a topic of intense philosophical debates.

For example, if we consider a plurality of water molecules under appropriate environmental conditions *fluidity* and *wetness* are new concepts that are introduced to describe the emergent phenomena of the *compound physical entity* water. The *intra-ordinal Navier-Stokes law* explains and predicts the *fluid dynamics* of *water*—this law is meaningless if there is no state of liquidity. In contrast, the weight of water is *resultant*.

A second category of laws, called trans-ordinal laws [44], explains the appearance of the emergent phenomena.

> ***Trans-ordinal Law: A Law that explains the emergence of the whole and the new phenomena at the macro-level out of the properties and interactions of the parts at the lower adjacent micro-level.***

We distinguish between two types of emergence, *explained emergence* and *unexplained emergence*. This classification can only be accomplished after a thorough analysis of the emergent phenomenon has been carried out.

> ***Explained emergence: An emergent phenomenon that is observed at a macro level is explained emergent if a trans-ordinal law that explains the occurrence of the emergent phenomenon at the macro level out of the properties and interactions of the parts at the adjacent micro level is known (or has been formulated post facto).***
>
> ***Unexplained Emergence: An emergent phenomenon that is observed at the macro level is unexplained emergent if, after a careful analysis of the emergent phenomenon, no trans-ordinal law that explains the appearance of the emergent phenomenon at the macro level out of the properties and interactions of the parts at the adjacent micro level is known (at least at present).***

The distinction between *explained emergent* and *unexplained emergent* depends on the *state of knowledge in the scientific community.* If, after a phase of detailed observation and analysis, an emergent phenomenon cannot be explained on the basis of existing knowledge about the constituent systems and their interactions, then it is classified as *unexplained emergent.* Often the extension or modification of known laws at the micro-level will provide for some later *post facto* explanation of the emergent phenomenon. The emergent phenomenon will then be reclassified as *explained emergent*.

There are some unexplained emergent phenomena that up to now cannot be explained, based on the present state of knowledge, although an immense effort has been invested in order to gain an understanding of these phenomena. Examples for these phenomena are the *emergence of life* or the *emergence of the mind*. It is an ongoing philosophical debate whether *our limited knowledge* or *ontological novelty* is the reason for the difficulty in explaining these unexplained emergent phenomena.

2.6.3 Supervenience

The principle of *supervenience* [46] establishes an important dependence relation between the emerging phenomena at the macro-level and the interactions and arrangement of the parts at the micro-level.

> ***Supervenience: The principle of Supervenience states that (Sup i) a given emerging phenomenon at the macro level can emerge out of many different arrangements or interactions of the parts at the micro-level while (Sup ii) a difference in the emerging phenomena at the macro level requires a difference in the arrangements or the interactions of the parts at the micro level.***

Because of *Sup-i* one can abstract from the many different arrangements or interactions of the parts at the micro level that lead to the same emerging phenomena at the macro level. *Sup-i* entails a *significant simplification* of the higher-level models of a holarchy.

Because of *Sup-ii* any difference in the emerging phenomena at the macro level can be traced to some significant difference at the micro level. *Sup-ii* is important from the point of view of *failure analysis*.

Let us look at the example of a transistor. The *transistor effect* is an emergent effect caused by the proper arrangement of dopant atoms in a semiconducting crystal. The exact arrangement of the dopant atoms is of no significance as long as the provided behavioral specifications of a transistor are met. In a VLSI chip that contains millions of transistors, the detailed microstructure of every single transistor is probably unique, but the external behavior of the transistors (the holons) is considered the *same* if the behavioral parameters are within the given specifications. It is a tremendous simplification for the designer of an electronic circuit that she/he does not have to consider the unique microstructure of every single transistor.

2.6.4 Downward Causation

In the literature about emergence the topic of *downward causation* of the emergent phenomena is intensively discussed.

> ***Downward Causation: The phenomenon that some novel macro-level properties have causal powers to control the micro-level properties from which they emerge.***

A good example for downward causation is the establishment of a fault-tolerant global time base at the macro-level from an ensemble of local clocks at the micro-level by the execution of a distributed synchronization algorithm. The synchronization algorithm uses the states of the local clocks to establish a global time-base. The global time base causes the states of a local clock to be corrected in order to assure that the local clock remains in agreement with the progression of the global time.

2.6.5 Classification of Emergence in an SoS

An SoS is formed by the information interactions and physical interactions among *Constituent Systems (CSs)*, e.g,. new or existing monolithic systems (called *legacy systems*). When analyzing emergence in an SoS, the focus is on two distinct adjacent levels: the micro-level of the CSs that interact via messages to cause the emergent phenomena (behavior) of interest at the macro-level, the level of the SoS.

The type of emergence that is occurring in the cyber part of an SoS is *explained emergence,* even if we are surprised and cannot explain the occurrence of an unexpected emergent phenomenon at the moment of its first encounter. If we have made proper provisions to observe and document all interactions (messages) among the CSs in the domains of time and value we can replay and analyze the scenario after the fact. At the end, we will find the mechanisms that explain the occurrence of the emergent phenomenon. There is no ontological novelty in the interactions of the CSs in the cyber parts of an SoS.

Table 3 depicts four cases of emergent behavior that must be distinguished in an SoS. *Expected and beneficial* emergent behavior is the normal case. Un*expected and beneficial emergent* behavior is a positive surprise. *Expected detrimental emergent* behavior can be avoided by adhering to proper design rules. The problematic case, which we investigate in the following in more detail, is *unexpected detrimental emergent behavior*.

Table 3: Classification of Emergent Behavior

Contribution of the emergent behavior to the overall goal of a system **Emergent Behavior**	**Beneficial**	**Detrimental**
Expected	Normal case	Avoided by appropriate rules
Unexpected	Positive surprise	Problematic case

2.6.6 Causes of Emergence In an SoS

We identified two main causes that can lead to emergent behavior in an SoS, *causal loops* and *avalanche effects*.

> ***Causal Loop: A causal loop exists, if the emergent property at the macro-level causes a change of the state of the parts at the micro-level.***

A good example for a causal loop is the distributed clock synchronization discussed above. Causal loops require that there exists an information flow from the macro-level to the parts at the micro-level. In some CPSoS, this information flow is realized via stigmergic channels.

The detailed analysis of the information flows in an SoS can detect causal loops that have the potential to cause emergent phenomena. However in a CPSoS there is a fundamental limitation to detect all possible causal loops, because the scope of the CPSoS is undefined. There may be hidden interactions among constituent systems that are not considered in the available interface models.

Emergent phenomena can also be caused by a *Cascade effect* (Fisher 06):

> ***Cascade Effect: A cascade effect exists, if in a system with a multitude of parts at the micro level a state change of a part at the micro-level causes successive state changes of many other parts at the micro level such that the cumulative effect of the totality of these state changes results in a novel phenomenon.***

A good example for a cascade effect is an *epidemic*. Cascade effects are *diachronic*, since they develop over time.

2.7 PROBLEM SOLVING

2.7.1 Basic Concepts

Many systems are developed in order to provide a service that helps a user in the solution of some identified problem.

> ***Problem: A perceived need to transform an initial state to a goal state.***

The initial state, the starting state for a problem solving activity, can be classified as follows:

Initial State: (i) an existing deficient state of affairs that needs a solution or (ii) a recognized opportunity that should be exploited or (iii) a formal statement of a question (academic story problem).

In a real-life problem scenario, there are normally many states that can be considered a solution to a given problem.

Solution Set: The set of states that are considered a solution to the problem at hand.

The goal state is any element of the solution set.

Goal State: An element of the solution set.

In the first phase, a problem solver must develop a decision procedure to determine whether an element of the state space is a member of the solution set.

Problem Space: A state space that contains the initial state, the solution states and the identified constraints that the paths form the initial state to the solution state must satisfy.

Acceptance Test: A test that determines if a state in the problem space is a member of the solution set.

Normally, the size of the problem space is limited by a set of given constraints.

Constraint: A restriction in the problem space.

We call a possible path that leads from the initial state through the problem space to a goal state a solution path.

Solution Path/Plan: A path of intermediate states from the initial state to the goal state, considering the given constraints.

Some states of a solution path are important intermediate results.

Sub Goal: A significant intermediate state on the solution path.

To summarize, we consider the following concepts as the problem elements:

Problem Elements: The initial state, the solution states, the acceptance test, the sub goals and the constraints that must be considered on the solution path.

2.7.2 Problem Types

It is useful to distinguish between the structural and situational characteristics of a problem.

Structural Characteristics: Representation of the abstract structure of the problem, focusing on the problem elements and their interactions.

The situational characteristics focus on the context of the problem.

Situational Characteristics: Representation of the context of the problem that anchors the problem in the real world and elaborates the constraints.

Problems can be classified along a scale from well-structured to ill-structured.

Well-structured Problem: A problem, where the initial state, the solution states, the acceptance test, and the constraints are well defined.

Well-structured problems are dominated by the structural characteristics. Most textbook problems are well-structured problems.

Ill-structured Problem: A problem where either the initial state and/or the solution states, and/or the acceptance test and/or constraints, are not well defined.

Ill-structured problems are dominated by the situational characteristics. Most real-life problems are ill-structured problems.

Problem Framing: The process of describing and interpreting a problem within the problem domain (also the point of view, from which the problem is described).

We can distinguish between the following problem types (from *well-structured* to *ill-structured*):

Formal Problem: A problem in a well-defined problem space.

Example: Textbook problem, logic problems, algorithmic problems.

Decision Problem: A problem to select an alternative out of a set of given alternatives.

Example: Should I accept an offered job?

Diagnosis Problem: A problem to find the cause of observed symptoms.

Example: Medical Diagnosis, technical troubleshooting.

Strategy Problem: A problem to devise a strategy or conceive a design that satisfies an abstract goal state under a myriad of explicit and implicit constraints.

Example: devise a policy to reduce the deficit.

Problem Solving: The process of finding a satisficing goal state. Problem solving encompasses (i) Analyzing and specifying the initial state (ii) Exploring the constraints in the problem domain (iii) Finding one or a set of satisficing goal states and (iv) Finding a solution path from the initial state to a selected goal state that observes the given constraints.

2.8 DEPENDABILITY, SECURITY AND CRITICALITY

In this section we define the basic concepts related to dependability, security and multi-criticality. These are important properties for System-of-Systems (SoSs) since they impact availability/continuity of operations, reliability, maintainability, safety, data integrity, data privacy and confidentiality. Definitions for dependability and security concepts can be very subtle; slight changes in wording can change the entire meaning. Thus, we have chosen to refer to basic concepts and definitions that are widely used in the dependability and security community.

The reference taxonomy for the basic concepts of dependability applied to computer-based systems can be found in [3]. It is the result of a work originated in 1980, when a joint committee on "Fundamental Concepts and Terminology" was formed by the TC on Fault–Tolerant Computing of the IEEE CS1 and the IFIP WG 10.4 "Dependable Computing and Fault Tolerance" with the intent of merging the distinct but convergent paths of the dependability and security communities.

In addition to the work of Laprie [3] [4], we also refer to definitions from the *CNSS Instruction No. 4009: National Information Assurance (IA) Glossary* [1]. The CNSSI 4009 was created through a working group with the objective to create a standard glossary of security terms to be used across the U.S. Government, and this glossary is periodically updated with new terms. We also cite security terms as defined by Ross Anderson's "Security Engineering: A Guide to Building Dependable Distributed Systems" [7] and Bruce Schneier's "Applied Cryptography" [8].

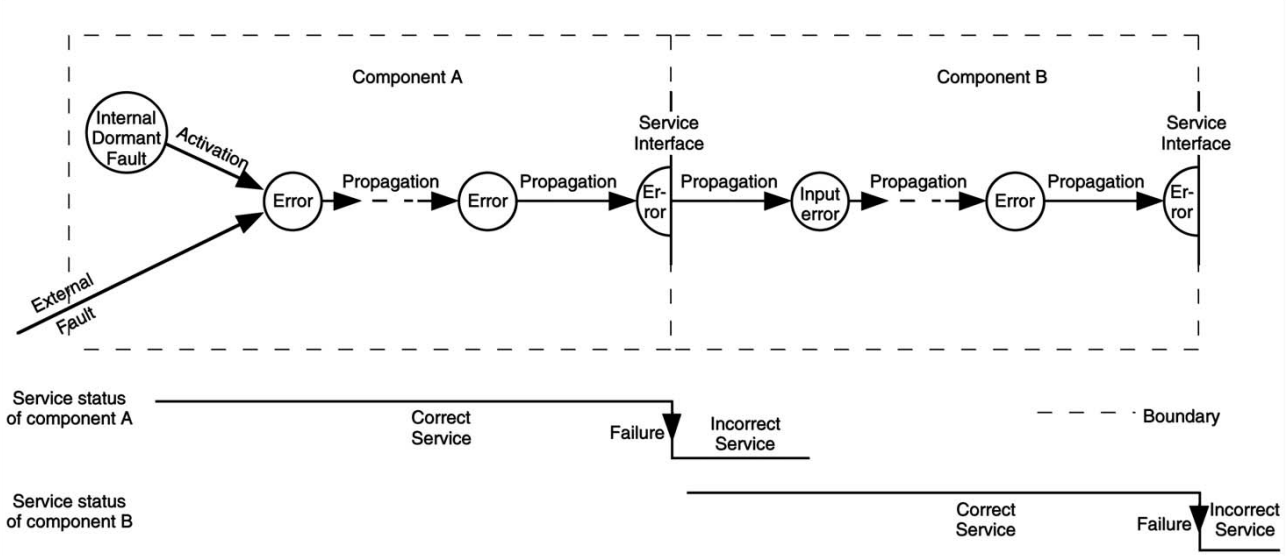

Figure 1: The chain of threats: a fault in component A activates and generates an error; errors propagate until component A fails; the failure of A appears as an external fault to B [4].

2.8.1 Threats: Faults, Errors, and Failures

The threats that may affect a system during its entire life from a dependability viewpoint are failures, errors and faults. Failures, errors and faults are defined in the following, together with other related concepts.

> ***Failure: The actual system behavior deviation from the intended system behavior.***

A system may fail in different forms, so the following concept of failure mode is introduced:

> ***Failure modes: The forms that the deviations from the system service may assume; failure modes are ranked according to failure severities (e.g. minor vs. catastrophic failures).***

At some point in time a system may fail, i.e., shows behavior that deviates from the intended behavior, and may return to the intended behavior at some later point in time:

> ***System outage: The interval during which the system has failed, i.e. where the behavior of the system deviated from its intended one.***
>
> ***System restoration: The transition from system failure to intended system behavior.***
>
> ***Error: Part of the system state that deviated from the intended system state and could lead to system failure.***

It is important to note that many errors do not reach the system's external interfaces, hence do not necessarily cause system failure.

> ***Fault: The adjudged or hypothesized cause of an error; a fault is active when it causes an error, otherwise it is dormant.***

The prior presence of vulnerability, i.e., a fault that enables the existence of an error to possibly influence the system behavior, is necessary such that a system fails.

The creation and manifestation mechanism of faults, errors, and failures, depicted in Figure 1, is called "chain of threats". The chain of threats summarizes the causality relationship between faults, errors and failures. A fault activates (fault activation) in component A and generates an error; this error is successively transformed into other errors (error propagation) within the component (internal propagation) because of the computation process. When some error reaches the service interface of component A, it generates a failure, so that the service delivered by A to component B

becomes incorrect. The ensuing service failure of component A appears as an external fault to component B.

2.8.2 Dependability, Attributes, and Attaining Dependability

Dependability (original definition): The ability to deliver service that can justifiably be trusted.

The above definition stresses the need for justification of "trust", so an alternate definition is given:

Dependability (new definition): The ability to avoid failures that are more frequent and more severe than is acceptable.

This last definition has a twofold role, because in addition to the definition itself it also provides the criterion for deciding whether the system is dependable or not.

Dependability is an integrating concept that encompasses the following dependability attributes:

Availability: Readiness for service.

Reliability: Continuity of service.

Maintainability: The ability to undergo modifications and repairs.

Safety: The absence of catastrophic consequences on the user(s) and on the environment.

Integrity: The absence of improper system state alterations.

A specialized secondary attribute of dependability is robustness.

Robustness: Dependability with respect to external faults (including malicious external actions).

The means to attain dependability (and security) are grouped into four major dependability categories:

Fault prevention: The means to prevent the occurrence or introduction of faults.

Fault tolerance: The means to avoid service failures in the presence of faults.

Fault removal: The means to reduce the number and severity of faults.

Fault forecasting: The means to estimate the present number, the future incidence, and the likely consequences of faults.

Fault prevention is part of general engineering and aims to prevent the introduction of faults during the development phase of the system, e.g. improving the development processes.

Fault tolerance aims to avoid the occurrence of failures by performing error detection (identification of the presence of errors) and system recovery (it transform a system state containing one or more errors into a state without detected errors and without faults that can be activated again) over time.

Fault removal can be performed both during the development phase, by performing verification, diagnosis and correction, and during the operational life, by performing corrective and preventive maintenance actions.

Fault forecasting is conducted by performing an evaluation of system behavior with respect to fault occurrence of activation, using either qualitative evaluations (identifying, classifying and ranking the failure modes) or quantitative ones (evaluating in terms of probabilities the extent to which some of the attributes are satisfied).

The relationship among the above mentioned means are the following: fault prevention and fault tolerance aim to provide the ability to deliver a service that can be trusted, while fault removal and

fault forecasting aim to reach confidence in that ability by justifying that the functional and the dependability and security specifications are adequate and that the system is likely to meet them.

> ***Fault Containment Region: A Fault Containment Region (FCR) is a collection of components that operates correctly regardless of any arbitrary fault outside the region.***

Fault effects might enter a FCR as erroneous input data and while this FCR might still operate by itself correctly its output actions are also erroneous. Error containment solves this problem.

> ***Error Containment: Error Containment prevents propagation of errors by employing error detection and a mitigation strategy.***

A possible error mitigation strategy would be to drop erroneous input.

> ***Error Containment Region: A set of at least two Fault Containment Regions (FCRs) that perform error containment.***

2.8.3 Security

Continuing with the concepts defined by Laprie [4], when addressing security an additional attribute needs to be considered: confidentiality.

> ***Confidentiality: The absence of unauthorized disclosure of information.***

Based on the above definitions, security is defined as follows:

> ***Security: The composition of confidentiality, integrity, and availability; security requires in effect the concurrent existence of availability for authorized actions only, confidentiality, and integrity (with "improper" meaning "unauthorized").***

We also consider the security definitions proposed by the CNSSI 4009 [1], which discusses security in terms of risk management. In this glossary, security is a condition that results from the establishment and maintenance of protective measures that enable an enterprise to perform its mission or critical functions despite risks posed by threats to its use of information systems.

> ***Threat: Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, or other organizations through a system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.***

A threat can be summarised as a failure, error or fault.

> ***Vulnerability: Weakness in a system, system security procedures, internal controls, or implementation that could be exploited by a threat.***

Vulnerabilities can be summarised as internal faults that enable an external activation to harm the system [4].

Risk is defined in terms of the impact and likelihood of a particular threat.

> ***Risk: A measure of the extent to which an organization is threatened by a potential circumstance or event, and typically a function of 1) the adverse impacts that would arise if the circumstance or event occurs; and 2) the likelihood of occurrence.***

Encryption using cryptography can be used to ensure confidentiality. The following definitions from Bruce Schneier's "Applied Cryptography" [8], have been modified to incorporate the definitions of *data* and *information* given in Section 2.3.1.

> ***Encryption: The process of disguising data in such a way as to hide the information it contains.***

Cryptography: The art and science of keeping data secure.

Data that has not been encrypted is referred to as plaintext or cleartext.

Plaintext: Unencrypted data.

Data that has been encrypted is called ciphertext.

Ciphertext: Data in its encrypted form.

The opposite of encryption is referred to as decryption.

Decryption: The process of turning ciphertext back into plaintext.

Encryption systems generally fall into two categories, asymmetric and symmetric, that are differentiated by the types of keys they use.

Key: A numerical value used to control cryptographic operations, such as decryption and encryption.

Symmetric Cryptography: Cryptography using the same key for both encryption and decryption.

Asymmetric cryptography is also known as public key cryptography.

Public Key Cryptography: Cryptography that uses a public-private key pair for encryption and decryption.

Symmetric cryptography uses the same key for encryption and decryption, called the symmetric key.

Symmetric Key: A cryptographic key that is used to perform both encryption and decryption.

Public key cryptography uses two kinds of keys, a public key and a private key.

Private Key: In an asymmetric cryptography scheme, the private or secret key of a key pair which must be kept confidential and is used to decrypt messages encrypted with the public key.

Generally, the private key is used for decryption and the public key is used for encryption. Depending on the cryptographic scheme, the public key can be used for authentication.

Public Key: A cryptographic key that may be widely published and is used to enable the operation of an asymmetric cryptography scheme. This key is mathematically linked with a corresponding private key.

Within cryptographic operations, Kerckhoff's principle states that only the cryptographic key (symmetric key for symmetric cryptography and the private key for public key cryptography) must remain secret in order for the system to be secure. This entails that the cryptographic algorithm and other settings can be public knowledge without compromising the secrecy of the encrypted information.

Controlling access to a system entails both authentication (verifying who is using the system) and authorization (verifying what they can do) for subjects and objects.

Subject: An active user, process, or device that causes information to flow among objects or changes the system state.

Object: Passive system-related devices, files, records, tables, processes, programs, or domain containing or receiving information. Access to an object implies access to the information it contains.

Authentication: The process of verifying the identity or other attributes claimed by or assumed of a subject, or to verify the source and integrity of data.

Authorization: Authorization is the mechanism of applying access rights to a subject. Authorising a subject is typically processed by granting access rights to them within the access control policy.

The terms “authorization” and “access control” are often used interchangeably. ***[6]***

Access Control: Access control is concerned with providing control over security critical actions that take place in a system. Providing control over actions consists of explicitly determining either the actions that are permitted by the system, or explicitly determining the actions that are not permitted by the system.

Authorization is typically managed using an access control model.

Access Control Model: An access control model captures the set of allowed actions as a policy within a system.

Controlling what a subject can do within the system is normally managed by assigning permissions to that subject.

Permission: Attributes that specify the access that subjects have to objects in the system.

The security policy states what permissions are held by the users [10].

Security Policy: Given identified subjects and objects, there must be a set of rules that are used by the system to determine whether a given subject can be permitted to gain access to a specific object. This is called the security policy.

In an access control system, the security policy is enforced by what is called the reference monitor.

Reference Monitor: A reference monitor represents the mechanism that implements the access control model. A reference monitor is defined as: An access control concept that refers to an abstract machine that mediates all accesses to objects by subjects.

The concept of trust is defined in terms of failures and the ability to break the security policy ***[7]***.

Trusted System: A trusted system or component is one whose failure can break the security policy.

Trusted Computing Base: The set of components (hardware, software, human,…) whose correct functioning is sufficient to ensure that the security policy is enforced, or, more vividly, whose failure could cause a breach of the security policy.

Anderson also distinguishes between *trusted* and *trustworthy* [7]. The failure of a trusted system could break the security policy, whereas a trustworthy system will not fail (and thus break the security policy).

Trustworthy System: A system or component that warrants trust because the system or component's behavior can be validated in some convincing way, such as through formal analysis or code review.

In the next phase of the AMADEOS project, we will investigate how to build a secure and dependable SoS architecture and develop best practices for such secure and dependable systems.

2.8.4 Criticality

Our definition of criticality is based on [66] where the criticality of a system component is determined by dependability requirements.

Criticality level: The criticality level is the level of assurance against failure.

Criticality: Criticality is a designation of the required criticality level for a system component.

Components of a system may have different levels of criticality; in this case we consider the system to be a multi-criticality system.

Multi-criticality system: A multi-criticality system has at least two components that have a different criticality.

"Up to five levels may be identified (see, for example, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards)" [66].

Critical service: A critical service is the service of a system that requires a specific criticality level.

For example, a railway system is a multi-criticality system given that it consists of components that deliver, e.g., a braking service and a heating service. These components usually adhere to different Safety Integrity Levels (SIL) resulting in a system exhibiting different levels of criticality.

In engineering, typically a top-down design approach is followed where the criticality of a system component is determined by the highest critical service the component is supposed to deliver.

2.9 EVOLUTION AND DYNAMICITY

2.9.1 Basic concepts

Large scale Systems-of-Systems (SoSs) tend to be designed for a long period of usage (10 years+). Over time, the demands and the constraints put on the system will usually change, as will the environment in which the system is to operate. The AMADEOS project studies the design of systems of systems that are not just robust to dynamicity (short term change), but to long term changes as well. This Section addresses a number of terms related to the evolution of SoSs.

Evolution: Process of gradual and progressive change or development, resulting from changes in its environment (primary) or in itself (secondary).

Although the term evolution in other contexts does not have a positive or negative direction, in the SoSs context, evolution refers to maintaining and optimizing the system - a positive direction, therefore.

Managed evolution: Evolution that is guided and supported to achieve a certain goal.

For SoSs, evolution is needed to cope with changes. Managed evolution refers to the evolution guidance. The goal can be anything like performance, efficiency, etc. The following two definitions further detail managed evolution for SoSs:

Managed SoS evolution: Process of modifying the SoS to keep it relevant in face of an ever-changing environment.

This is Primary evolution; examples of environmental changes include new available technology, new business cases / strategies, new business processes, changing user needs, new legal requirements, compliance rules and safety regulations, changing political issues, new standards, etc.

Unmanaged SoS evolution: Ongoing modification of the SoS that occurs as a result of ongoing changes in (some of) its CSs.

This is Secondary evolution; examples of such internal changes include changing circumstances, ongoing optimization, etc. This type of evolution may lead to unintended emergent behavior, e.g., due to some kind of "mismatch" between Constituent Systems (CSs) (see Section 2.1.3).

Managed SoS evolution is due to changes in the environment (see Section 2.1.2).

Local Evolution: Local evolution only affects the internals of a Constituent System (CS) which still provides its service according to the same and unmodified Relied Upon Interface (RUI) specification.

Global Evolution: Global evolution affects the SoS service and thus how CSs interact. Consequently, global evolution is realized by changes to the Relied Upon Interface (RUI) specifications.

Evolutionary Performance: A quality metric that quantifies the business value and the agility of a system.

Evolutionary Step: An evolutionary change of limited scope.

Minor Evolutionary Step: An evolutionary step that does not affect the Relied Upon Interface (RUI) Itom Specification (I-Spec) and consequently has no effects on SoS dynamicity or SoS emergence.

Major Evolutionary Step: An evolutionary step that affects the Relied Upon Interface (RUI) Itom specification and might need to be considered in the management of SoS dynamicity and SoS emergence.

The goal of managed evolution in AMADEOS is maximizing business value while maintaining high SoS agility:

Business value: Overarching concept to denote the performance, impact, usefulness, etc. of the functioning of the SoS.

Agility (of a system): Quality metric that represents the ability of a system to efficiently implement evolutionary changes.

Quantizing business value is difficult since it is a multi-criteria optimization problem. Aiming for Pareto optimality, various aspects (measured by utility functions) are weighted on a case-by-case basis.

System performance is a key term in the concept of business value:

System performance: The combination of system effectiveness and system efficiency.

System effectiveness: The system's behavior as compared to the desired behavior.

System efficiency: The amount of resources the system needs to act in its environment.

For the last definition, it is important to understand what system resources are in the area of SoS:

System resources: Renewable or consumable goods used to achieve a certain goal. E.g., a CPU, CPU-time, electricity.

Dynamicity of a system: The capability of a system to react promptly to changes in the environment.

Linked to dynamicity and to the control strategy shifting from a central to an autonomous paradigm, is the concept of *reconfigurability*.

Reconfigurability: The capability of a system to adapt its internal structure in order to mitigate internal failures or to improve the service quality.

2.9.2 Scenario-based Reasoning

The management of any activity entails a number of processes that are continuously executed. These processes can be summarised with reference to the Observe, Orient, Decide and Act

(OODA) loop[2]. In AMADEOS these steps are applied to the *managed evolution* of an SoS. Given the context of an SoS, the following assumptions can be reasonably made:

1. It is impossible to observe the *entire* state of the SoS. Therefore, observations about the SoS are partial (i.e., incomplete), and possibly uncertain and/or conflicting. This may lead to a number of possible interpretations or mental perspectives on the (current) state of the SoS and its possible developments.
2. It is impossible to *precisely* determine the *best* course of actions to take, given a certain interpretation or mental perspective on the state of the SoS, and to predict the effects of management actions. This may also lead to a number of possible outcomes of the decisions made and actions taken.

Given the predominant uncertainties in the processes above, a structured technique that can assist in the Orient and Decide phases of the managed SoS evolution and address those uncertainties is introduced, namely Scenario-Based Reasoning (SBR).

> ***Scenario-Based Reasoning (SBR): Systematic approach to generate, evaluate and manage different scenarios in a given context.***

Scenarios, in turn, are "what-if" stories that give alternative accounts of a situation and may include assumptions as well as real pieces of information.

> ***Scenario: A scenario is a projected or imagined sequence of events describing what could possibly happen in the future (or have happened in the past).***

As such, scenarios should be plausible (not go beyond what is possible), consistent (with no contradictions between parts) and coherent (with explicit logical connections between events).

Scenarios can be used to deal with uncertainties regarding a situation, supporting processes of situation assessment and decision making.

> ***Situation assessment: Situation assessment is the process of achieving, acquiring or maintaining situation awareness.***
>
> ***Decision making: Decision making is the process resulting in the selection of a course of action among several alternative possibilities.***

Situation awareness, on the other hand, is a state of knowledge on the situation.

> ***Situation awareness: The perception of the context in a given situation, the comprehension of their meaning and the projection of their status in the near future.***

In order to generate scenarios representing e.g. multiple states of SoS situation awareness or the possible effects of management actions on the SoS, SBR makes use of observations of some relevant states and domain models in combination with SoS inference processes.

> ***Domain models: Models that represent relations between different items of knowledge in a given domain.***
>
> ***SoS inference processes: Processes that map observations about the SoS to estimates about states that are relevant for decision making or planning but cannot be directly observed.***

The domain models represent the general knowledge of the domain, i.e. knowledge that is independent of the specific system that is reasoned about. An example of a domain model is a causal model.

[2] Attributed to United States Air Force (USAF) Colonel John Boyd, who gave numerous presentations on this topic. No original papers written by him. For more information, see http://en.wikipedia.org/wiki/OODA_loop

Causal model: Abstract model describing the causal dependencies between relevant variables in a given domain.

The causal model must express more than correlation relations, since correlation does not imply causation. The causal dependencies in causal models can be made explicit through the use of causal graphs.

Causal graphs: Directed graphs representing the cause-effect relations between the variables (nodes) in the graph.

In summary, the domain models represent the scenario structure whereas the (SoS) inference processes are used to derive values for the variables in the scenario.

The management of generated set scenarios for a given situation involves the processes of scenario pruning and scenario updating.

Scenario pruning: Scenario pruning is the process of discarding scenarios that include incorrect or unlikely information.

Scenario updating: Scenario updating addresses the problem of how to deal with newly available information, not present in the generated set of scenarios.

In order to support the managed SoS evolution, modifications to the SoS may be proposed in the form of decision alternatives.

Decision alternative: Alternative course-of-action that may be chosen in the process of decision making.

For the managed-evolution of SoSs each decision alternative is (likely to be) a plan of actions, such as (but not restricted to): adjustments to monitoring, configuration changes of the SoS or its CSs, functioning changes, behavior changes, add/remove/update CSs, change environment, etc.

The evaluation of each decision alternative is typically performed taking into account the decision problem's overall goal (e.g. maximizing business value of the SoS). Typically, the overall goal is broken down into (multiple) criteria, resulting in a multi-criteria optimization problem that may be addressed with Multi-Criteria Decision Analysis (MCDA).

Multi-Criteria Decision Analysis (MCDA): MCDA is a sub-discipline of operations research that explicitly considers multiple criteria in decision-making, allowing the evaluation of one or more decision alternatives in light of the multiple criteria.

With SBR it is possible, in a structured and traceable manner, to reason about uncertain and missing information, as well as to explore possible different futures, including futures in which given decision alternatives are effectuated. The MCDA approach allows the evaluation of these decision alternatives for a number of different scenarios.

2.10 SYSTEM DESIGN AND TOOLS

SoSs can have a very complex architecture. They are constituted by several CSs which interact with each other. Due to their complexity, well defined design methodologies should be used, in order to avoid that some SoS requirement is not fulfilled and to ease the maintainability of the SoS.

2.10.1 Architecture

The architecture (defined in Section 2.1.2) represents a more static view about how the components constitute the system. The architecture of a system can have some variants or even can vary during its operation.

Then this adaptability of the architecture is translated into three more concepts.

Evolvable architecture: An architecture that is adaptable and then is able to incorporate known and unknown changes in the environment or in itself.

Flexible architecture: Architecture that can be easily adapted to a variety of future possible developments.

Robust architecture: Architecture that performs sufficiently well under a variety of possible future developments.

The architecture then involves several components which interact with each other. The place where they interact is defined as interface (see Section 2.1.2).

During the development lifecycle of a system, we start from conceptual thoughts which are then translated into requirements, which are then mapped into an architecture. The process that brings designers to define a particular architecture of the system is called design.

Design: The process of defining an architecture, components, modules and interfaces of a system to satisfy specified requirement.

In the AMADEOS context, design is a verb, architecture is a noun. The people who perform the design are designers.

Designer: An entity that specifies the structural and behavioral properties of a design object.

There are several methodologies to design a system.

Hierarchical Design: A design methodology where the envisioned system is intended to form a holarchy or formal hierarchy.

Top Down Design: A hierarchical design methodology where the design starts at the top of the holarchy or formal hierarchy.

Bottom Up Design: A hierarchical design methodology where the design starts at the bottom of the holarchy or formal hierarchy.

Meet-in-the-Middle Design: A hierarchical design methodology where the top down design and the bottom up design are intermingled.

This methodology is useful to decrease the degree of the complexity of a system and to ease its maintainability. In particular, in the hierarchical design the system can be split in different subsystems that can be grouped in modules.

Module: A set of standardized parts or independent units that can be used to construct a more complex structure.

The modularity is the technique that combines these modules in order to build a more complex system.

Modularity: Engineering technique that builds larger systems by integrating modules.

In the context that an SoS shall deal with evolving environments, then also design methodologies focused on evolution shall be defined.

Design for evolution: Exploration of forward compatible system architectures, i.e. designing applications that can evolve with an ever-changing environment. Principles of evolvability include modularity, updateability and extensibility. Design for evolution aims to achieve robust and/or flexible architectures.

Examples for design for evolution are Internet applications that are forward compatible given changes in business processes and strategies as well as in technology (digital, genetic, information-based and wireless).

In the context of SoS, design for evolution can be reformulated in the following manner.

> ***Design for evolution in the context of SoS: Design for evolution means that we understand the user environment and design a large SoS in such a way that expected changes can be accommodated without any global impact on the architecture. 'Expected' refers to the fact that changes will happen, it does not mean that these changes themselves are foreseeable.***

In addition during the system development lifecycle some activities to verify if the architecture developed during the design process is compliant and if it fulfills the requirements of the system are foreseen. Verification of design is an important activity especially in critical systems and several methodologies are defined to perform it. In particular there is a methodology which has in mind verification since the design phase.

> ***Design for testability: The architectural and design decisions in order to enable to easily and effectively test our system.***

Then we have two methodologies to perform design verification:

> ***Design inspection: Examination of the design and determination of its conformity with specific requirements.***
>
> ***Design walkthrough: Quality practice where the design is validated through peer review.***

In order to perform all these activities some useful tools can be used to help the designers and verifiers job.

2.11 GOVERNANCE

The underlying purpose of SoS governance is to ensure that the interoperation among constituent systems will achieve SoS goals.

> ***Governance: Theoretical concept referring to the actions and processes by which stable practices and organizations arise and persist. These actions and processes may operate in formal and informal organizations of any size; and they may function for any purpose.***

In the context of SoS, governance can be implemented in a collaborative fashion, due to the independence of the constituent systems.

> ***Collaborative governance: Process and a form of governance in which participants (parties, agencies, stakeholders) representing different interests are collectively empowered to make a policy decision or make recommendations to a final decision-maker who will not substantially change consensus recommendations from the group.***

The governance actions and processes are initiated by an administrative domain part of the organization.

> ***Administrative domain: An organization that controls resources and constituent systems that are part of the SoS (or: going to be part of the SoS).***

Administrative domain has authority to make changes to configuration of resources & systems, and their connections to other resources & systems in other administrative domains.

> ***Authority: The relationship in which one party has the right to demand changes in the behavior or configuration of another party, which is obliged to conform to these demands.***

(Collaborative) SoS Authority: An organizational entity that has societal, legal, and/or business responsibilities to keep a collaborative SoS relevant to its stakeholders. To this end it has authority over RUI specifications and how changes to them are rolled out.

For the purpose of rolling out changes to RUI specifications, the SoS authority needs the capabilities to measure the state of the implemented changed RUI specifications, and to give incentives to motivate CSs to implement the RUI specification changes.

Incentive: Some motivation (e.g., reward, punishment) that induces action.

2.12 QUALITY METRICS AND ATTRIBUTES

Including quality metrics and attributes in the conceptual model for the design of System-of-Systems (SoS) is challenging. In fact, on the one hand, SoS are systems themselves, and so dependability, resilience, security levels, quality of service and performance attributes and metrics normally defined for a Constituent System (CS) can be in principle used for SoS functional correctness verification, as well. On the other hand, SoSs have peculiarities that call for specific quality attributes and metrics, such as the interaction of autonomous and independently evolving CSs and the previously defined emergence.

Quality: The standard of something as measured against other things; the degree of excellence of something.

Quality of Service: The ability of a system to meet certain requirements for different aspects of the system like performance, dependability, evolvability, security or cost; possibly expressed in terms of levels and quantitatively evaluated through metrics.

Metric: Indicator used to quantitatively describe an attribute of the system, like throughput for performance or availability for dependability.

Resilience: Persistence of service delivery that can justifiably be trusted, when facing changes. Changes here may refer to unexpected failures, attacks or accidents (e.g., disasters). Changes may also refer to increased loads. Resilience thus deals with conditions that are outside the design envelope whereas other dependability metrics deal with conditions within the design envelope [3].

Security Level: Specification of the level of security to be achieved through the establishment and maintenance of protective measures.

It is useful to look at Table 1, in order to single out a set of attributes that are specific for SoS and their peculiarities compared to old-monolithic systems.

As regards *testing*, the need for continuous testing vs. established test phases, and the frequency and nature of faults becoming *normal* rather than *exceptional* events, call for *testability* and (the already defined) *robustness* of SoS. Also, *repeatability* is more difficult to be granted.

Testability: Property of a system to support testing.

Repeatability: Condition of measurement, out of a set of conditions that includes the same measurement procedure, same operators, same measuring system, same operating conditions and same location, and replicate measurements on the same or similar objects over a short period of time [5].

Testing SoSs brings a set of relevant challenges, which lead to the necessity continuous testing. As a result, Constituent Systems need to be equipped with testing supports. Amongst these challenge, we report [87], [86]:
i) *lack of observability of the code and structure*, as Constituent Systems are most likely represented as simple interfaces, preventing white-box testing approaches,

ii) *Dynamicity, adaptiveness and in-service reconfiguration,* where the Constituent Systems composing the SoS may not know each other and change while providing SoS services,
iii) *Partial control or lack of control,* because Constituent Systems runs independently, possibly geographically distributed, and evolve without requiring notification to a global coordinator or to the users
iv) costs, that can vary depending on the kind of SoS and its dimension.

Repeatability is often not achievable when measurements are carried out on SoS. The same conditions can, in fact, hardly be guaranteed. There are many factors that may affect *repeatability*, but non-determinism deserves special attention. Especially, distributed, asynchronous SoS are inherently non-deterministic. Thus, two executions of the same experiment may produce different, but nevertheless valid, ordering of events. In field measurement or controlled experiments it may be difficult to reproduce results of experiments. For example, we can consider dynamic and adaptive systems where despite the geographic mobility of the nodes is very carefully described and followed during the course of the experiments, the results of the experiments may differ since the propagation conditions are strongly influenced by changing environment conditions [88].

Similarly, the fact that *requirements* and *specifications* are not fixed anymore, but subject to frequent changes poses dynamicity (cf. Section 2.9) as a relevant attribute.

> ***Requirement: A statement that identifies a necessary attribute, capability, characteristic, or quality of a system.***

According to the definition of SoS: "*An SoS is an integration of a finite number of constituent systems which are independent and operable, and which are networked together for a period of time to achieve a certain higher goal,"* the role of communication is central, as well as the notion of time. Consequently, information *integrity* and *consistency*, and communication *timeliness* are vital attributes of an SoS.

> ***Consistency: The property of a set of entities that see the same data at the same time.***

Due to the heterogeneity of SoS natures and scopes, it is difficult to generalize attributes and metrics, as relevant attributes of different SoSfs may differ from one another. So, we will specify quality attributes and metrics presented in this Section in the conceptual model and add domain-specific attributes and metrics, on a use-case basis (see WP4).

> ***Contract: Agreement between two or more parties, where one is the customer and the others are service providers. This can be a legally binding formal or an informal "contract". It can be expressed in terms of objectives.***
>
> ***Objective: Values for the quality metrics to be attained.***
>
> ***Quality of Experience/ Quality of Perception: A subjective measure of a user's experience/perception with a system and satisfaction for a system's quality.***

2.12.1 Examples

There are general quality metrics related to virtual interactions between Constituent Systems (CSs). These include, for instance, data throughput, response time, data link bandwidth, message recipients set, number of incoming/outgoing messages.

The concept of data throughput represents the rate at which data is received or sent by a Constituent System.

> ***Incoming data throughput: The amount of data, per unit of time, that a Constituent System (CS) receives from other CSs.***
>
> ***Outgoing data throughput: The amount of data, per unit of time, that a Constituent System (CS) sends to other CSs.***

In case of request-response interactions between Constituent Systems (CSs), a message can represent a request sent by a CS to another CS, or a message can represent a response received by a CS. The concepts of response time, and message throughput apply in this type of interactions.

Response time: The time elapsed between a request message is sent by a Constituent System (CS) to another CS, and a response message is received by the message CS.

Incoming message throughput: The amount of messages, per unit of time, that a Constituent System (CS) receives from other CSs.

Outgoing message throughput: The amount of messages, per unit of time, that a Constituent System (CS) sends to other CSs.

2.13 INTERFACES

Central to the integration of systems are their interfaces, i.e., their points of interaction with each other and the environment over time. A point of interaction allows for an exchange of information among connected entities.

Interaction: An interaction is an exchange of information items at connected interfaces.

The concept of a channel represents this exchange of information at connected interfaces.

Channel: A logical or physical link that transports information among systems at their connected interfaces.

A channel is implemented by a communication system (e.g., a computer network, or a physical transmission medium) which might affect the transported information, for example by introducing uncertainties in the value/time domains. In telecommunications a channel model describes all channel effects relevant to the transfer of information.

Channel Model: A model that describes effects of the channel on the transferred information.

An interaction at connected interfaces has the following fundamental attributes:

- **Transferred Information:** Every interaction involves the transfer of information among participating systems.
- **Temporal Properties:** An interaction takes time, i.e., for an interaction to occur it is initiated and completed according to system-specific temporal properties.
- **Dependability Requirements:** For example, interactions might require resilience w.r.t. perturbation or need to guarantee security properties like confidentiality. Usually, dependability requirements are inherited from the requirements of systems that participate in the interaction.

Interface Properties: The valued attributes associated with an interface.

Interface Layer: An abstraction level or modelling viewpoint under which interface properties can be discussed.

Cyber Space: Cyber space is an abstraction of the Universe of Discourse (UoD) that consists only of information processing systems and cyber channels to realize message-based interactions.

Environmental Model: A model that describes the behavior of the environment that is relevant for the interfacing entities at a suitable level of abstraction.

Note that abstraction is always associated with a given specified purpose.

Interface properties can be characterized at different levels of abstraction:

- **Cyber-Physical Layer:** At the cyber-physical layer information is represented by data items (e.g., a bit-pattern in cyber space, or properties of things/energy in the physical world) that are transferred among interacting systems during the Interval of Discourse (IoD). While in this layer there is a distinction between cyber- and physical channels, both share many properties, because cyber channels are implemented by physical channels. Consequently any interaction over cyber-physical channels is ruled by the progression of time and fundamentally constrained by the speed of light and distance among communicating systems. Time is an elemental property of cyber- and physical interfaces and must be considered at all interaction abstractions. Important properties of the cyber-physical layer are: signals (i.e., prearranged representation of information), transmission medium, characteristics of connectors, frequencies, bit rates, energy levels.
- **Information Layer:** This interface layer concerns the timely exchange of Itoms [32] by unidirectional channels across interfaces. It provides an abstraction over cyber-physical channels to context-independent, direct and indirect information flows among systems and their environment [51]. Itoms are information items, i.e., '*a timed proposition about some state or behavior in the world*' [32], in some representation together with the explanation of this representation. Interacting systems may adhere to different contexts, i.e., often implicit assumptions about the actual meaning and temporal properties of exchanged data. Conceptually, Itoms solve this context dependency problem and allow for a consistent interpretation of exchanged data. Itom channels are implemented by cyber-physical channels. An Itom channel is characterized by sender, recipients, temporal properties, and dependability properties.
- **Service Layer:** At the service layer, the interface exposes the system behavior structured as capabilities. In contrast to the informational layer, Itom channels are not individually described at the service layer, but only the interdependencies between the exchanged Itoms are specified. If a system with a need is matched with a system that offers the needed capability, the interdependencies must be resolved in the information interface layer with concrete Itom channels. Hence, at the service interface layer there is an instantiable collection of Itom channels per offered capability where generic properties of the Itom channels and their interaction pattern are described.

Each system's point of interaction is an interface that (1) together with all other system interfaces establishes a well-defined boundary of the system, and (2) makes system services to other systems or the environment available. Consequently, any possibly complex internal structure that is responsible for the observable system behavior can be reduced to the specification of the system interfaces [2].

> ***Interface Specification: The interface specification defines at all appropriate interface layers the interface properties, i.e., what type of, how, and for what purpose information is exchanged at that interface.***
>
> ***Interface Cyber-Physical Specification (CP-Spec): Part of the interface specification that concerns interface properties at the cyber-physical interface layer.***
>
> ***Interface Message Specification (M-Spec): Part of the CP-Spec that concerns the specification of messages exchanged with the cyber space environmental model.***
>
> ***Interface Physical Specification (P-Spec): Part of the CP-Spec that concerns the specification of exchanges with the physical environmental model.***
>
> ***Interface Itom Specification (I-Spec): Part of the interface specification that concerns interface properties at the informational interface layer.***

Interface Service Specification (S-Spec): Part of the interface specification that concerns interface properties at the service interface layer.

Depending on the system complexity and its underlying architecture the interface specification needs to be considered at appropriate levels of abstraction and modelling viewpoints. On the one hand, systems may require a specification at all of the introduced abstraction levels. For example, a temperature-controlled room where a temperature sensor, a heating element and a PID controller interact via message-based, physically explicitly specified interfaces. On the other hand, Service-based Applications (SBAs) like cloud-based file-sharing in the context of the Internet do not require details about the physical level of interfaces anymore. TCP/IP connectivity is assumed to be present regardless of the actual physical and logical underlying channels.

Interface Environment: The interface environment of an interface consists of the universe of all entities and actions that can have an impact on the afferent (input) data of that interface.

There are two distinct and different interface environments associated with every interface. E*fferent* (output) data is part of the interface environment if it has a — possibly indirect — influence on the *afferent* data.

Active Interface Environment: The interface environment during an activity interval.

Disjoint Interface Environment: Two interface environments are disjoint, if there is no interaction among these two interface environments.

Joint Interface Environment: Two interface environments are joint, if there are interactions among these two interface environments.

2.13.1 Cyber-Physical Interfaces

At the cyber-physical abstraction level we have cyber- and physical interfaces.

2.13.1.1 PHYSICAL INTERFACES

Physical interfaces of CSs are realized by energy transformers that are able to (1) take observations from the physical environment, and (2) set actions initiated from the computer system of the CS in the physical environment. An observation in time-aware CPSoSs is a time-stamped measurement of a physical state variable (a property of a thing). A sensor is an interface device that measures the physical environment and produces observations in the form of digital data (a bit pattern), whereas the sensor design determines which property of the physical environment is observed. Sensor-fusion and state estimation [55] are well researched techniques to improve the fidelity of sensor observations. An actuator is an interface device that accepts digital data and control-information (e.g., an actuation deadline) from an interface component, and realizes the intended effect in the physical environment (influences physical state variables).

Sensor: A sensor is an interface device that observes the system environment and produces data (a bit pattern) that can be explained by the design of the sensor and its placement in the physical environment.

Actuator: An actuator is an interface device that accepts data and control information from an interface component and realizes the intended physical effect at its placement in the physical environment.

Transducer: An interface device converting data to energy or vice versa. The device can either be a sensor or an actuator.

As physical interfaces enable the interaction with the time-sensitive physical environment, they are time-sensitive as well. Hence, sensor/actuator latency and jitter affect the timely accuracy of an observation or the timely effect in the physical environment.

2.13.1.2 CYBER INTERFACES

Cyber interfaces produce and/or consume messages, i.e., bit-patterns in cyber space (e.g., an email). Their interface specification consists of three parts: (1) the transport specification, (2) the syntactic specification, and (3) the semantic specification.

> ***Transport Specification: This part of the interface specification describes all properties of a message that are needed by the communication system to correctly transport a message from the sender to the receiver(s).***

A correctly transported message adheres to all temporal and dependability specifications.

> ***Message-based Interface Port: The message-based interface contains ports (i.e., channel endpoints) where message payloads can be placed for sending, or received message payloads can be read from.***

A port has the following properties:

- **Direction:** Each port has either the direction incoming, i.e., incoming messages can be read from the port, or the direction outgoing, i.e., outgoing messages can be written to the port.
- **Size:** The size of the data contained in the message determines the size of the port.
- **Type:** The port type specifies whether the message contains state data and should adhere to a read/write paradigm or event data and should adhere to the consume/produce paradigm.
- **Temporal Properties:** The temporal properties determine the temporal behavior of a message with respect to maximum/minimum delay, maximum jitter, periodicity, and bounds on send and receive instants.
- **Dependability Properties:** The dependability properties specify dependability parameters (e.g., reliability, security, availability, …) of the message transport.

The syntactic- and semantic specifications (see Section 2.5) establish the Itoms associated with a message. The named syntactic units of the message payload (syntactic specification) and their explanations (semantic specification) establish the message variables.

> ***Message Variable: A tuple consisting of a syntactic unit of a message and a name, where the name points to the explanation of the syntactic unit.***

The semantic interface specification can be given by an interface model which refers to the names of message variables and contains their explanation.

> ***Interface Model: The interface model contains the explanation of the data sent or received over this interface and thus establishes the Itoms.***

If the interface meta-model provides a time aware execution semantics, the sum of all executable interface models of a system can be used to simulate the system service at the message level.

> ***Interface State: The state of the interface model at the ground state instant.***

The interface state can be partitioned as follows:

- db-state: data base state, i.e. the interface state that is stored in a self contained data base
- ei-state: external input interface state, i.e., interface state that can be retrieved from the external environment by input operations
- eo-state: external output interface state that is in the sphere of control of the interface model

- cr-state: critical interface state, i.e., an interface state that is considered relevant and is neither db-state, ei-state, or eo-state

Declared Interface State: At a ground state instant, the value assigned to a declared data structure of the interface model, comprising all data items of the cr-state.

We regard the point of interaction of two or more systems from two sides: the server side and the client side. Each of the sides is local to each system and requires consideration in the respective interface specifications. For a valid exchange of information the server and client(s) interface specifications need to be compatible, i.e., the specified exchange of information must match at server and client(s) sides.

Formal methods on formally defined interface specifications can be used to verify whether interfaces are compatible.

Context compatibility: the same data (bit pattern) is explained in the same way at the sender and at the receiver.

Context incompatibility: the same data (bit pattern) is explained differently at the sender and at the receiver.

Syntactic Compatibility: The syntactic chunks sent by the sender are received by the receiver without any modification.

Compatibility (full, Itom): The Itom that is sent by the sender is received by the receiver without modification.

In case of context compatibility, syntactic compatibility suffices to realize full compatibility.

In case of context incompatibility a gateway is required to translate the data representation of the sender to a data representation that is compatible with the context of the receiver.

Gateway: A transformation system in cyberspace.

In a stigmergic channel, the environment, including the actuators at the sender and the sensors at the receiver forms a transformation system.

2.13.2 Itom Interfaces

The abstraction over cyber-physical channels at the Informational layer removes any lower-level details of the interactions that is not relevant for describing the behavior of CSs. Itoms at this layer are maximally refined and explicitly specified, i.e., their meta data is available to the extent necessary for all CSs that are possibly involved with these Itoms. Their realization at the lower-level cyber-physical layer must adhere to the semantics specified at the informational layer, otherwise the abstraction is invalid and there is risk of *property mismatch* among interacting CSs. All for the CPSoS service relevant cyber-physical interactions must be taken into account at the informational layer. Otherwise there are hidden information flows at the informational layer which might compromise security, safety, or lead to detrimental emergence at the underlying cyber-physical layer.

Further, the information layer focuses on modeling the direct and indirect communication among CSs. There are cases where it is beneficial to not model every system involved in an interaction explicitly, but regard them as anonymous common environment of a smaller set of systems of interest which indirectly interact over this common environment. Indirect communication also allows for decentralized coordination of systems [Val04] and the description of cascading effects [71].

At the Itom level we discern between two fundamentally different communication channels:

- **Direct Communication:** Itoms are transferred unidirectionally from one sending to one or more receiving systems.

- **Indirect Communication:** The Itoms of a sending system affects the state of the common environment of one or more systems. Additionally the state of this common environment is possibly affected by environmental dynamics, i.e., time-sensitive processes that act autonomously and independently of the explicitly modeled systems. Finally, receiving systems read Itoms from the common environment by taking observations. The received Itoms represent the totality of all influences carried out by other systems or environmental dynamics. In contrast to direct communication, not all systems participating in an interaction need to be modeled explicitly, as long as their effects are appropriately considered in the model of the environmental dynamics.

The received Itoms represent the totality of all influences carried out by other systems or environmental dynamics. In contrast to direct communication, not all systems involved in an interaction need to be modelled explicitly, as long as their effects are considered in the model of the environmental dynamics.

2.13.3 Service-based Interfaces

Systems may provide many services through their interfaces given that their internal structure can rely on requested services. This concept is also a fundamental principle in the Service-Oriented-Architecture (SOA) [] where components in need of capabilities and components that offer capabilities are brought together by means of a service registry, service discovery, and service composition.

Service Provider: The component that provides a service.

Service Consumer: The component that requires a service.

Service Registry: The service registry is a repository of Interface Service Specifications (S-Specs) of service providers.

Service Discovery: Service discovery is the process where service consumers match their service requirements against the available Interface Service Specifications (S-Specs) in a service registry.

Service Composition: The integration of multiple services into a new service is called service composition.

The benefits of this service-based view are twofold:

- First, there is an immediate reduction of complexity, because we do not need to regard component relations on the basis of single message channels anymore. Service consumers can discover services they depend on and a scheduler can instantiate the necessary unidirectional message channels automatically. This scheduler ensures that there exist only channels among *compatible interfaces*.
- Second, the coupling of components (integrated in one system) is loose, because the actual locations of possibly composed services are unimportant background details in the service-based view. This location independence of services allows for self-organized system reconfiguration such that the system is able to perform optimally in case new services become available and previously active services become unavailable. For example, a service consumer does not depend on a single component to provide the required service, but the service consumer can lookup multiple suitable services from the service registry and choose the most optimal w.r.t. computational, communicational, or other costs.

These benefits have been originally observed in the context of free market economy where trading among buyers and sellers lead to efficiency in the production and distribution of products.

An S-Spec also includes a set of quality metrics that are available for an independent observer to determine the quality of a provided service. Based on this quality metrics, service providers can

offer their service under a Service Level Agreement (SLA) where prices of the service, the quality of the service, and compensation actions in case of failure to deliver a committed service are defined. Service providers can publish their SLA with a reference to the S-Spec of an offered service at the service registry, such that prospective service consumers can find and choose an appropriate service provider.

> ***Service Level Objective (SLO): A functional or non-functional objective that can be evaluated by observing the service provider to either achieved or not-achieved. Objectives are based on measurable quality metrics.***
>
> ***Service Level Agreement (SLA): A SLA defines a set of Service Level Objectives (SLOs), the price of the service, and compensation actions in case of failure to deliver a commited service.***

SLAs touch a wide range of issues related to service providers and consumers: e.g., "*services to be delivered, performance, tracking and reporting, problem management, legal compliance and resolution of disputes, customer duties and responsibilities, security, termination, […]"* [The Service Level Agreement Zone].

SLAs are expressed as a set of SLOs (Service Level Objectives) to achieve. The concept of SLO is defined in the following. The SLOs must be attainable, measurable, and controllable. Also, SLO / SLA violations might have economical and legal impacts, with consequences such as penalties, SLA renegotiation, etc. These aspects are out of the scope of the present AMADEOS conceptual model.

> ***Reservation: A commitment by a service provider that a resource that has been allocated to a service requester at the reservation allocation instant will remain allocated until the reservation end instant.***
>
> ***Reservation Request Instant: The instant when a resource is requested by a service requestor.***
>
> ***Reservation Allocation Instant: The instant when a resource reservation is allocated to a service requestor by a service provider.***
>
> ***Reservation End Instant: The instant until a reservation is allocated to a service provider.***

2.13.4 Interfaces of Constituent Systems integrated in System-of-Systems

Interfaces within Constituent Systems (CSs) that are not exposed to other CSs or the CS's environment are internal.

> ***Internal Interface: An interface among two or more subsystems of a Constituent System (CS).***
>
> ***External Interface: A Constituent System (CS) is embedded in the physical environment by its external interfaces.***

We distinguish three types of external CS interfaces: *Time-Synchronization Interface (TSI)*, *Relied Upon Interfaces (RUIs)* (see Section 2.1 and 3.2) and *utility interfaces*.

> ***Time-Synchronization Interface (TSI): The TSI enables external time-synchronization to establish a global timebase for time-aware CPSoSs.***
>
> ***Utility Interface: An interface of a CS that is used for the configuration, or the control, or the observation of the behavior of the CS.***

The purposes of the utility interfaces are to (1) configure and update the CS, (2) diagnose the CS, and (3) let the CS interact with its remaining local physical environment which is unrelated to the operative services of the SoS. In acknowledgement of these three purposes we introduce the utility

interfaces: Configuration Interface (C-Interface), Diagnostic Interface (D-Interface), and Local I/O Interface (L-Interface). Figure 2 gives an overview of the interfaces of a CS.

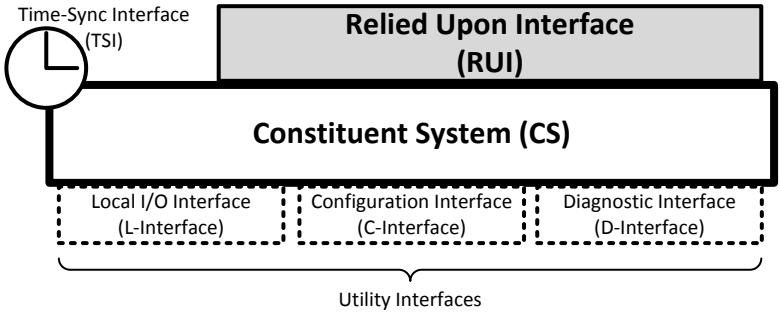

Figure 2: Interfaces of a Constituent System (CS).

> ***Configuration Interface (C-Interface): An interface of a CS that is used for the integration of the CS into an SoS and the reconfiguration of the CS's RUIs while integrated in a SoS.***

The C-Interface is able to modify the interface specification of RUIs. Concerning SoS evolution, we require in SoSs with no temporal guarantees and no sparse global time base that RUI specifications remain strictly backward-compatible, i.e., CSs must interact with and provide all legacy services, additionally to possibly new services. Otherwise, we have a disruption of legacy SoS service delivery at the affected CSs, because (even partly) non-compatible RUIs cannot (fully) interact and affected services on them remain inaccessible. However, if we can rely on a SoS where the CSs have access to a global time base, we can allow non-backward compatible updates (i.e., discontinuous evolution) and more importantly enforce a time-controlled SoS evolution. A predefined validity instant which is part of the interface specification determines when all affected CSs need to use the updated RUI specification and abandon the old RUI specification. This validity instant should be chosen appropriately far in the future (e.g., in the order of the update/maintenance cycle of all impacted CSs).

> ***Validity Instant: The instant up until an interface specification remains valid and a new, possibly changed interface specification becomes effective.***

Service providers guarantee that the old interface specification remains active until the validity instant such that service consumers can rely on them up to the reconfiguration instant.

> ***Diagnosis Interface (D-Interface): An interface that exposes the internals of a Constituent System (CS) for the purpose of diagnosis.***

The D-Interface is an aid during CS development and the diagnosis of CS internal faults.

> ***Monitoring Constituent System (CS): A CS of an SoS that monitors the information exchanges across the RUMIs of an SoS or the operation of selected CSs across the D-Interface.***

There are interfaces among the components of a CS which are hidden behind the RUI of the CS and which are not visible from the outside of a CS, e.g., the interface between a physical sensor and the system that captures the raw data, performs data conditioning and presents the refined data at the RUMI.

> ***Local I/O Interface (L-Interface): An interface that allows a Constituent System (CS) to interact with its surrounding physical reality that is not accessible over any other external interface.***

For example, a CS that controls the temperature of a room usually has at least the following L-Interfaces: a sensor to measure the temperature, an actuator that regulates the flow of energy to a

heater element, and a Human-Machine-Interface (HMI) that allows humans to enter a temperature set point.

Some external interfaces are always connected with respect to the currently active operational mode of a correct system.

> ***Connected Interface: An interface that is connected to at least one other interface by a channel.***

A disconnected external interface might be a fault (e.g., a loose cable that was supposed to connect a joystick to a flight control system) which causes possibly catastrophic system failure.

In a SoS the CSs may connect their RUIs according to a RUI connecting strategy that searches for and connects to RUIs of other CSs.

> ***RUI connecting strategy: Part of the interface specification of RUIs is the RUI connecting strategy which searches for desired, w.r.t. connections available, and compatible RUIs of other CSs and connects them until they either become undesirable, unavailable, or incompatible.***

For instance in the global ATM network, a cardholder together with a smartcard based payment card form a CS that is most of the time disconnected from any other CSs. The RUI connecting strategy of the cardholder/payment card CS is influenced by the cardholder's need for cash (desire), nearby located and operational ATM terminals (availability) and whether the ATM terminal accepts the payment card (compatibility).

One important class of faults that might occur at connected interfaces is related to compatibility.

> ***Property Mismatch: A disagreement among connected interfaces in one or more of their interface properties.***
>
> ***Connection System/Gateway Component/Wrapper: A new system with at least two interfaces that is introduced between interfaces of the connected component systems in order to resolve property mismatches among these systems (which will typically be legacy systems).***

3 CONCEPTUAL MODEL OF AN SOS

3.1 MODEL OVERVIEW

3.1.1 Introduction

The IEEE 42010 standard describes how to produce the architectural description of software-intensive systems. Such architectural description includes a conceptual model to support the description of architectures, and the required content of an architectural description. The conceptual model establishes the concepts, their definitions and relations which are then used in expressing the requirements.

Analogously to the conceptual model for the architecture of software intensive systems, in this deliverable we separate the description of basics SoS concepts into different perspectives. These perspectives are called views, each of which is focused on different concerns of the SoS.

In order to focus on the most relevant viewpoints, in alignment also with comments from reviewers received at the review meeting in Sophia Antipolis, in this deliverable we structured basic SoS concepts and their relationships in 7 different views thus focusing on core AMADEOS issues, namely:

- structure,
- dynamicity,
- evolution,
- dependability and security,
- time,
- multi-criticality,
- and emergence.

Structure represents architectural concerns of an SoS mainly concerning RUMIs and semantic of communications. *Dynamicity* and *evolution* represent short-term and long-term variations to the SoS, respectively. They should be achieved in order to react to external changes and possible failures. *Dependability and security* represent two major concerns of SoSs which become essential in critical environments. *Time* is fundamental since SoSs are sensitive to the progression of time. Managing emergence is essential to avoid un-safe unexpected situations generated from safe CSs interaction and to profit from positive emerging phenomena. Finally, *multi-criticality* allows to differently manage services with different criticalities.

In the following, we will describe for each different view its semantic relationships among concepts as they have been extracted from the definitions given in Section 2. We will adopt a graphical notation to express semantic relationships among concepts in each view: each entity represents a concept, while links stand for the semantic relationship extracted from the concept definitions. We will mainly focus on concepts related to design intents and goals without considering electronic aspects of SoSs, in alignment also with reviewers' comments.

3.1.2 Viewpoint of Structure

3.1.2.1 ROLE IN THE CONCEPTUAL MODEL.

The Viewpoint of Structure shows the structure of the SoS. It helps to understand how main CSs are associated with each other and how they communicate and they exchange data.

3.1.2.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Figure 3 depicts the semantic relationships for the concepts in the Structure viewpoint. An *SoS* integrates (from SoS to CS) a finite number of *CSs*. Constituent systems consist of (from CS to Human Controlled Object, Computer System) *computer systems*, and possibly *controlled objects* or possibly *humans* if they are considered as prime movers.

SoS are categorized into (from SoS to Directed SoS, Acknolwledged SoS, Collaborative SoS and Virtual SoS) *Directed SoS, Acknowledged SoS, Collaborative SoS, and Virtual SoS*. A CS has a (from CS to RUI) a *Relied Upon Interface (RUI)* that is composed of a Relied Upon Message Interface (RUMI) and a Relied Upon Physical Interface (RUPI). The RUMI allows the exchange of messages, and the RUPI allows the influence and observation of properties of things or force fields.

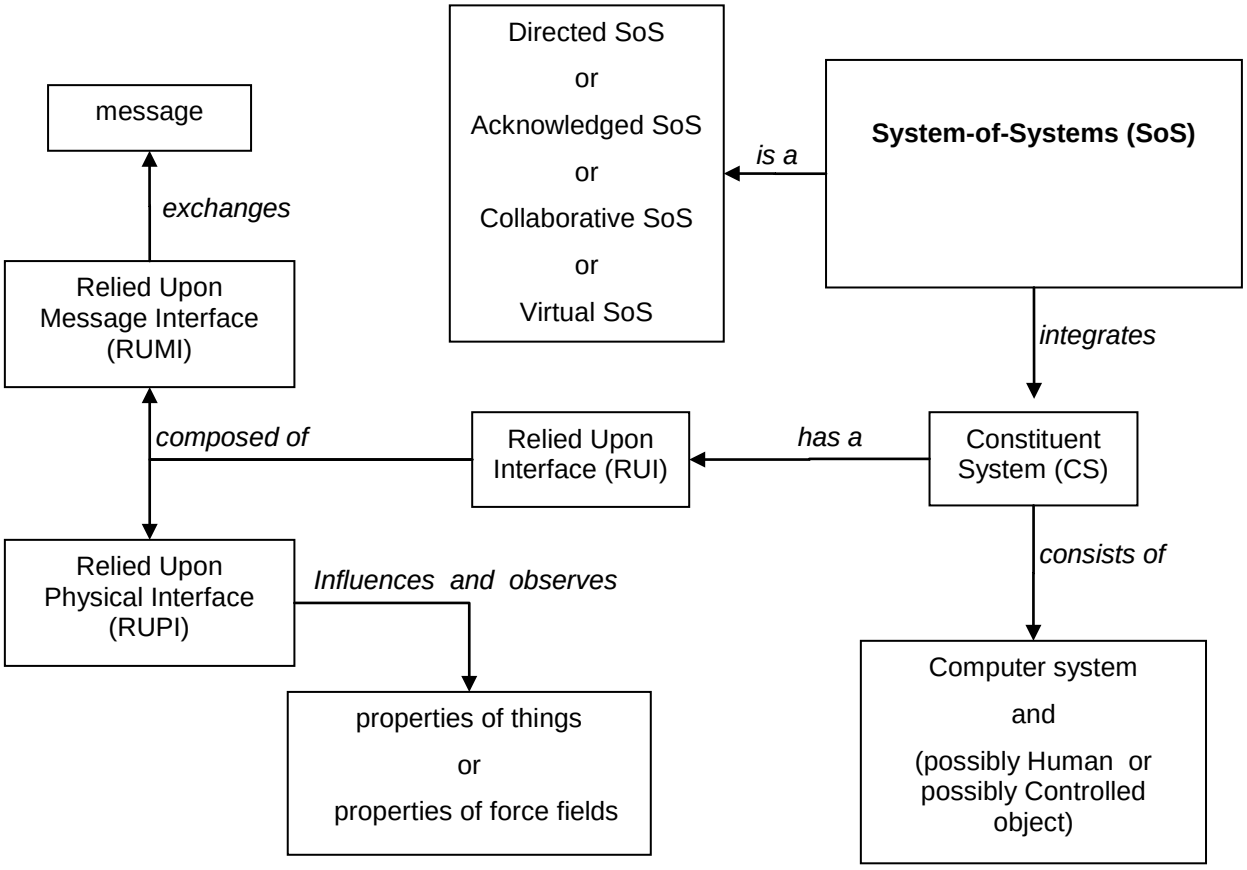

Figure 3: Semantic relationships. Viewpoint of structure.

3.1.3 Viewpoint of Dynamicity

3.1.3.1 ROLE IN THE CONCEPTUAL MODEL

This viewpoint describes the SoS from the dynamicity perspective. SoS dynamicity includes the adequate reconfiguration of SoS in specific situations, for example in case of an emergency management situation, such as a disaster or a failure of a CS after the occurrence of a fault.

3.1.3.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Semantic relationships related to dynamicity concepts are described in Figure 4. CSs exhibit (from CS to dynamicity) *dynamicity* in terms of (from dynamicity to services, structure and interactions) *services* they offer, their *structure* or the *interactions* with other entities. Dynamicity is managed by (from dynamicity to OODA loop) to OODA loop, which applies Decision Making and Scenario-Based Reasoning (SBR), that make use of SoS inference and Domain model such as Causal model. SBR involves scenario which deals with Decision making and Situation assessment and awareness.

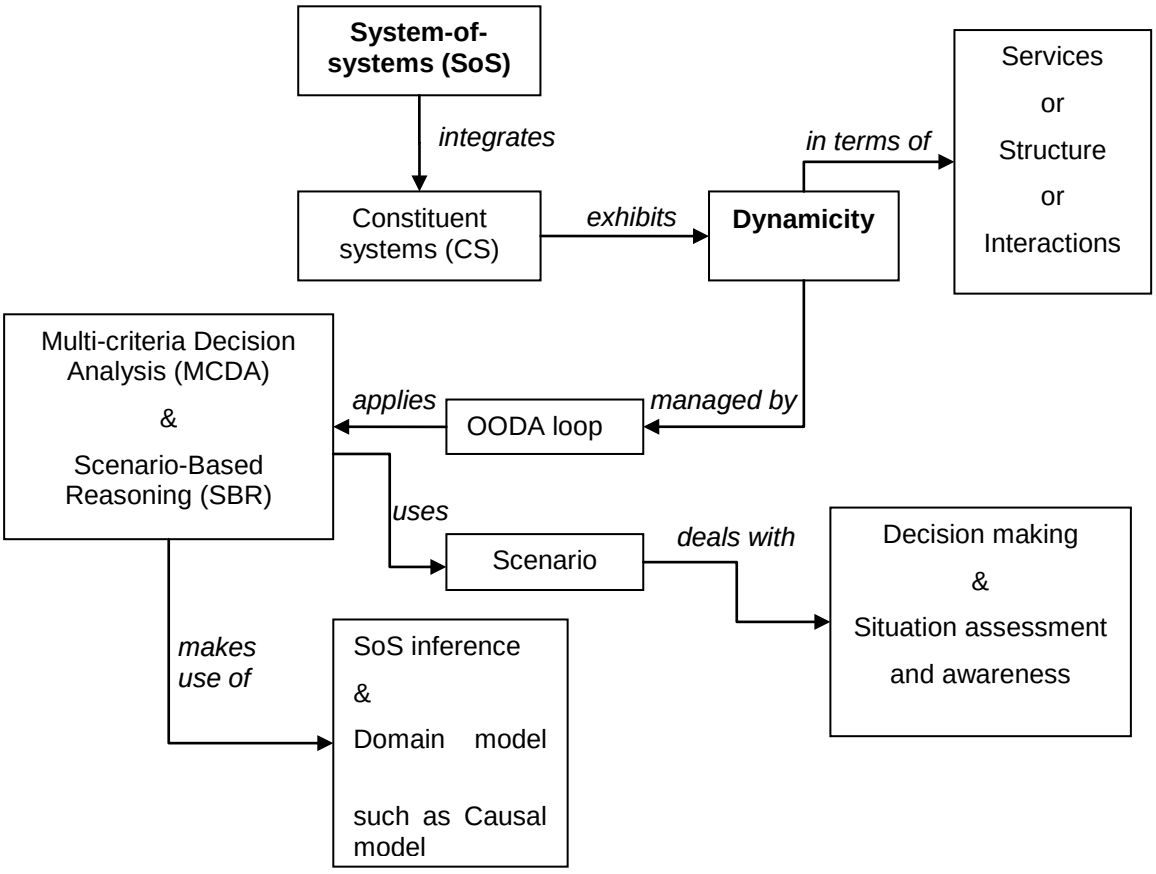

Figure 4: Semantic relationships. Viewpoint of Dynamicity.

3.1.4 Viewpoint of Evolution

3.1.4.1 ROLE IN THE CONCEPTUAL MODEL

The viewpoint of Evolution shows the SoS from the evolution perspective that deals with long-term adaptation. Evolution of an SoS is necessary for the adaptation to environmental changes such as

new business cases, legal requirements, compliance, changing safety regulations, evolving environmental protection rules, etc. SoS evolvability includes necessary modifications that are required to keep a system services relevant in the face of the ever-evolving society (e.g., new legal requirements, business cases, etc.).

3.1.4.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Figure 5 depicts semantic relationships for concepts in the evolution viewpoints. Evolution can be (from evolution to managed evolution and unmanaged evolution) managed or unmanaged. Environmental changes like new technologies, new legal requirements or changing user needs cause (from environment to managed evolution) managed evolution.

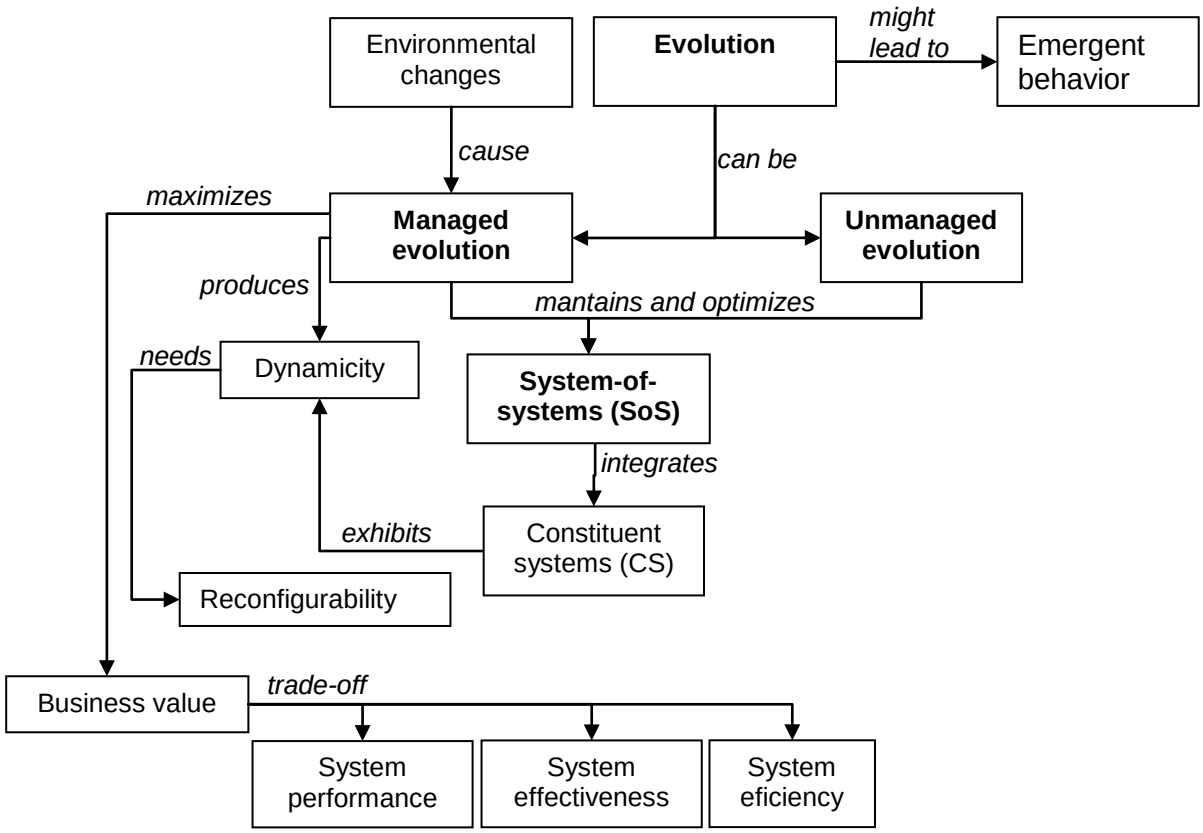

Figure 5: Semi-formal model. Viewpoint of Evolution.

Both managed evolution and unmanaged evolution maintain and optimize (from managed evolution and unmanaged evolution to SoS) the SoS. Unmanaged evolution may lead to (from unmanaged evolution to emergent behavior) unintended emergent behavior. Managed evolution produces (from managed evolution to dynamicity) dynamicity of CSs. In order to attain dynamicity, it is needed (from dynamicity to reconfigurability) reconfigurability. Managed evolution maximizes (from managed evolution to business value) the business value. The latter is a trade-off (from business value to System performance, System effectiveness and System efficiency) among System performance, System effectiveness and System efficiency.

3.1.5 Viewpoint of Dependability and Security

3.1.5.1 ROLE IN THE CONCEPTUAL MODEL.

The dependability and security viewpoint presents the SoS from the perspective of dependability and security properties. In any large system, faults and threats are normal and may impact on the availability, the continuity of operations, reliability, maintainability, safety, data integrity, data privacy and confidentiality. This insight has crucial consequences for the definition of the AMADEOS conceptual model and architecture framework.

3.1.5.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Semantic relationships among dependability concepts are shown in Figure 6. A fault causes (from fault to error) an error that might lead to (from error to failure) a failure. Failures cause (from failure to system outage) a system outage, which needs (from system outage to system restoration) system restoration.

Dependability is the means to avoid (from dependability to failure) failures. It integrates the attributes of availability, reliability, maintainability, safety, integrity and robustness. Dependability can be attained by means of fault prevention, fault tolerance, fault removal and fault forecast.

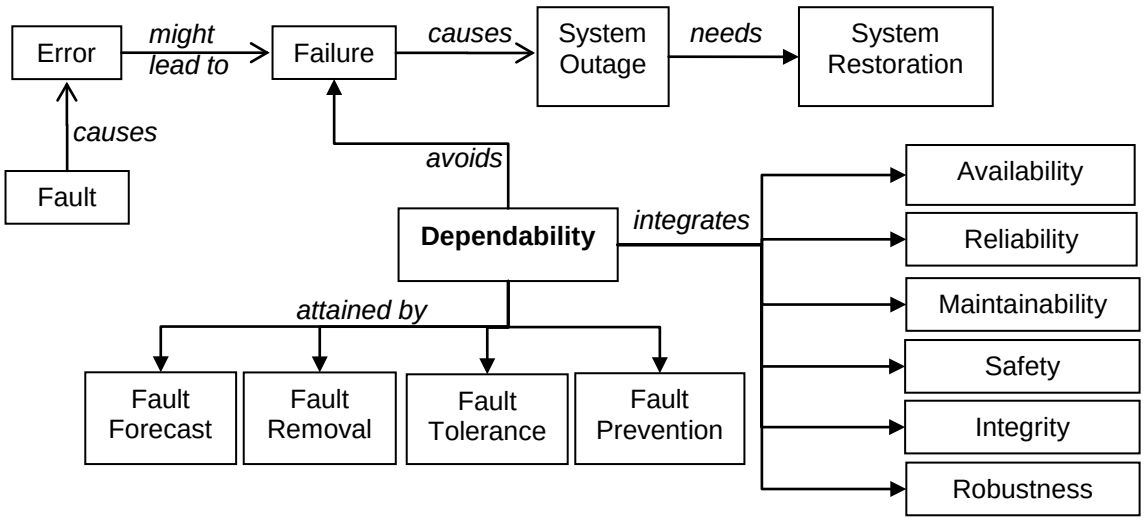

Figure 6: Semantic relationships. Viewpoint of Dependability.

Semantic relationships among security concepts are shown in Figure 7. Security integrates all the dependability attributes (see Figure 6) plus confidentiality. Security is impacted by threats that impose risks exploiting vulnerabilities that might be present in security. Confidentiality is ensured by means of encryption. Keys are used for encryption/decryption operations. Keys can be public or private.

Security needs authorization, which is provided by a reference monitor. Authorization assigns permissions, which are defined in a security policy. A security policy relies on trusted systems, which encompass hardware, software or human components.

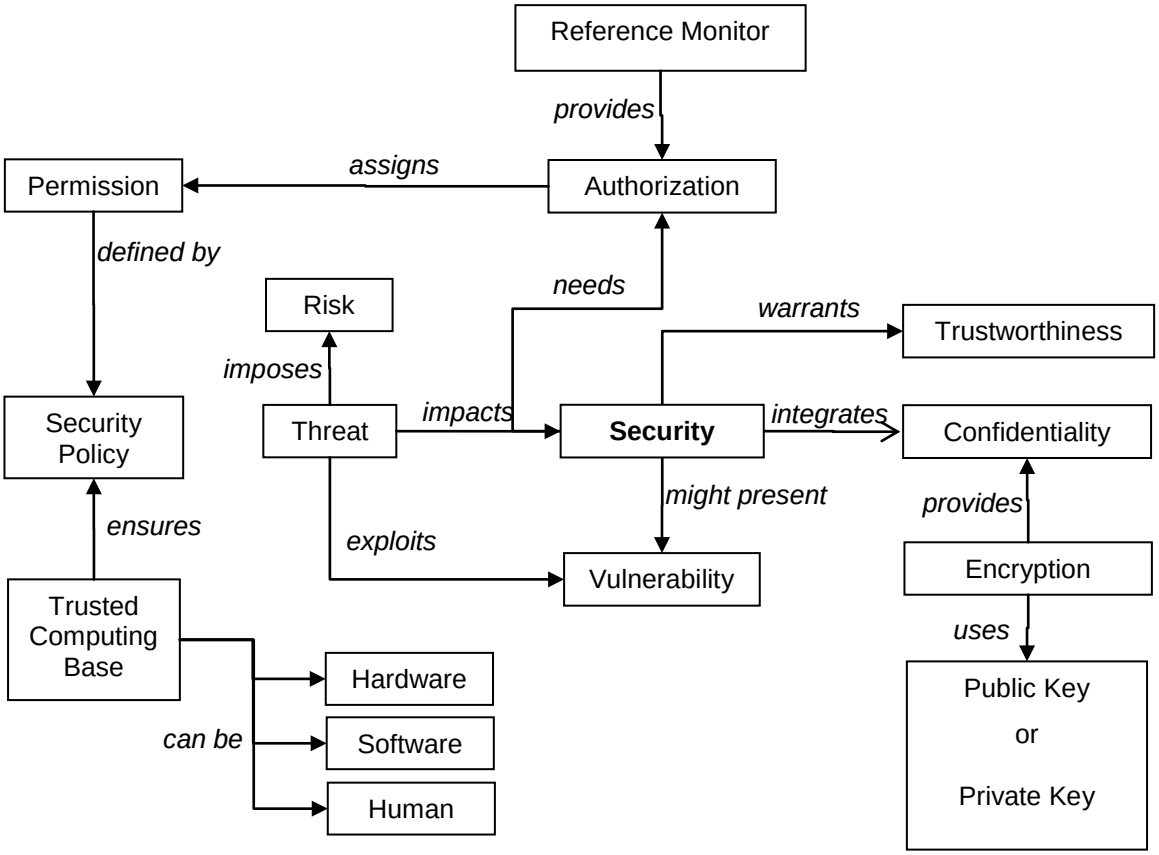

Figure 7: Semantic relationships. Viewpoint of Security.

3.1.6 Viewpoint of Time

3.1.6.1 ROLE IN THE CONCEPTUAL MODEL.

The viewpoint in this subsection describes the SoS from the perspective of time. Each constituent system, interface and element of an SoS has access to a precise, synchronized global time base.

3.1.6.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Figure 8 shows the concepts and their semantic relationships for the viewpoint of time.

SoS global time is an abstraction of physical time. SoS global time can be synchronized by *GPSDO,* which uses a *GPS* receiver. SoS Global time has (from SoS Global time to sparse time) a *sparse time* in which physical time is partitioned into *passive intervals* or *active intervals*. An event occurs in an interval.

An event has a *timestamp* associated w.r.t. a clock. The *absolute timestamp* is a timestamp generated by a *reference clock*.

A clock needs *synchronization*, which can be *internal and external*. The *accuracy* of a clock is measured by the reference clock. A *physical clock* has a *reference clock*.

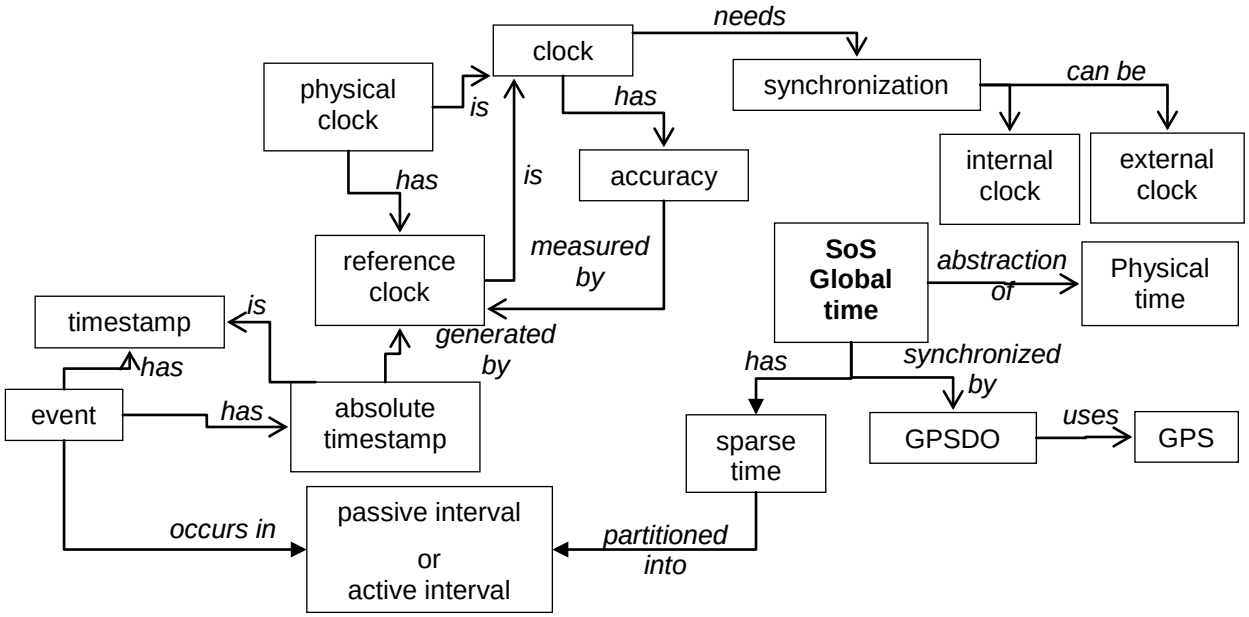

Figure 8: Semantic relationships. Viewpoint of Time.

3.1.7 Viewpoint of Multi-criticality

3.1.7.1 ROLE IN THE CONCEPTUAL MODEL.

In the multi-criticality viewpoint, we include all functionalities implemented by CSs that are critical for maintaining safety.

3.1.7.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

The semantic relationships among the concepts from the viewpoint of multi-criticality are depicted in Figure 9. Constituent systems (CS) implements functionalities that can be critical with a certain criticality policy.

In a System-of-Systems, different Constituent Systems may have different Criticality policies, which is denoted as multi-criticality. Critical functionalities may impact system safety.

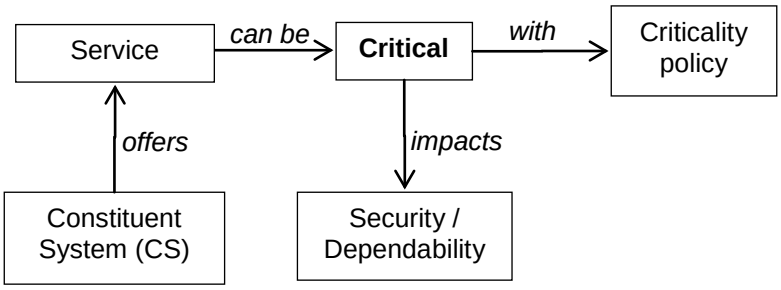

Figure 9: Semantic relationships. Viewpoint of Multi-Criticality.

3.1.8 Viewpoint of Emergence

3.1.8.1 ROLE IN THE CONCEPTUAL MODEL.

The SoS from the perspective of Emergence is described by the Emergence Viewpoint. Understanding emergence will help in the composition of CSs, especially in predicting the effects of composition on dependability, safety, security, availability.

3.1.8.2 SEMANTIC RELATIONSHIPS BETWEEN BASIC CONCEPTS

Figure 10 presents the semantic relationships between emergence-related concepts. Emergence can be expected or unexpected, beneficial or detrimental, and explained or unexplained.

Emergence produces a resultant phenomenon that can appear in the macro level (i.e. the SoS) or the micro level (i.e. the CS).

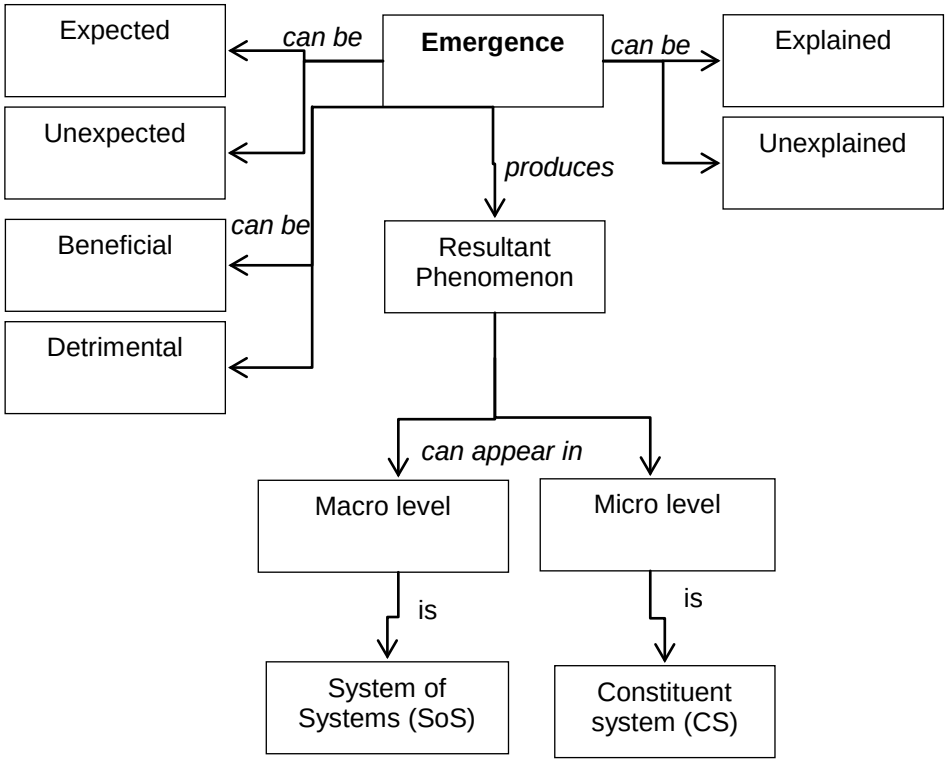

Figure 10: Semantic relationships. Viewpoint of Emergence.

3.2 RELIED UPON INTERFACES

For engineering and operating SoSs one of the most important concepts in the AMADEOS conceptual model is the *Relied Upon Interface (RUI)*, an interface where the SoS services relies upon. From a structural viewpoint, SoSs can be decomposed into *CSs* at their RUIs. They establish a *system boundary* of a CS by separating it from other CSs and the *physical environment*. By definition the CS *behavior* can be observed at its RUI, hence the *interface specification* of a CS's RUI hides the possibly complex internal structure from the overall SoS and vice versa. An interface specification can be regarded as a complexity firewall as it regulates all interactions taking place across the specified interface. Innate to RUIs, i.e., the points of interactions of CSs, is the transfer of *information* occurring over these interfaces. In our conceptual model this transfer of information among CSs is either accomplished by

- *message-based communication channels* in *cyber-space* at *Relied Upon Message Interfaces (RUMIs)*, or by
- *stigmergic communication channels* [51] in the *physical environment* of CSs at *Relied Upon Physical Interfaces (RUPIs)*.

In general, our identified purpose of SoSs is to realize *emergent services* that cannot be provided by a single CS in isolation. Consequently, one of the most crucial aspects of enabling such emergent services is the interaction of CSs and their environment. This interaction occurs over RUIs which again underlines their importance as a concept in the design of SoSs. Further, in many SoSs the number of CSs and their interaction capabilities are not static but change over time; sometimes quite suddenly. The effects of this and other dynamicity in SoSs have to be considered at RUIs and within their specification. Likewise the evolution of SoS – by changes in the behavior or the interaction of CSs to e.g., produce new emergent services – affects the CSs RUIs and determines how changes to their interface specification are carried out.

Therefore the concept RUI and its semantic relationships to other concepts defined in Section 2 are at the core of the AMADEOS conceptual model.

3.2.1 Information Transfer over RUIs

In our conceptual model, the fundamental interaction among CSs and their environment is the transfer of information by means of Itoms. This transfer of information is always accompanied by a transmission of physical quantities like energy or things. At the conceptual level, Itoms solve the context dependency problem of interpreting data as invariant information among CSs; even if these CSs adhere to different *architectural styles*. Itoms not only carry an information item in some representation (i.e., data), but also its explanation [32]. *Gateway components* that implement *transformation systems* can transform the data and explanation part of an Itom from one architectural style to another such that there is no information mismatch problem among CSs – provided they exchange Itoms over communication channels that do not invalidate the transported Itoms. Naturally the actual technical implementations of Itoms and gateway components are grand challenges by themselves, but are not within the scope of AMADEOS.

We consider the transfer of an Itom from one CS to another. The Itom originates at a CS, crosses its RUI to some communication channel, and either arrives in the common environment of the interacting CSs or directly arrives at one of the intended receiving CSs by crossing its RUI. We briefly discuss some of the properties of an Itom [32] that have implications on the design of RUIs:

- Name: The name of an Itom is a reference to its semantic content. For humans this name also serves as an explanation of the meaning of the Itom. The name of an Itom is context dependent and CSs require a shared ontology to correctly consume and produce Itoms.
- Purpose: There is some intention behind any information exchange which is ultimately responsible why information is emitted by a sender and received by receivers. In the context of RUIs the purpose of Itoms is rooted at the emergent SoS service and drives the interaction desires of CSs.

- Temporal Aspects: The utility of an Itom might be time sensitive. For example an Itom might contain the observation of a traffic light at an intersection. Clearly the utility of this Itom for a CS that is supposed to act on this information is limited by the progression of time.
- Physicalism: The storage and transfer of an Itom require a physical data carrier. At lower abstraction levels, data is observed via sensors (e.g., measuring voltage levels of an electrical wire) and produced via actuators. The physicalism property motivates a first refinement of RUIs.

We can separate communication that takes place among CSs over cyber channels, and communication that takes place indirectly among CSs over stigmergic channels in the common physical environment, i.e., in the overlapping *entourage*, of interacting CSs. Consequently we introduced the Relied Upon Message Interface (RUMI) for message-based communication and the Relied Upon Physical Interface (RUPI) for stigmergic communication. Figure 11 gives a structural overview of information transfer in SoSs at the *cyber-physical interface layer* by means of two interacting CSs: The CS A and the CS B are time-synchronized and have access to a sparse global time base such that transferred Itoms can be related to each other w.r.t. temporal order.

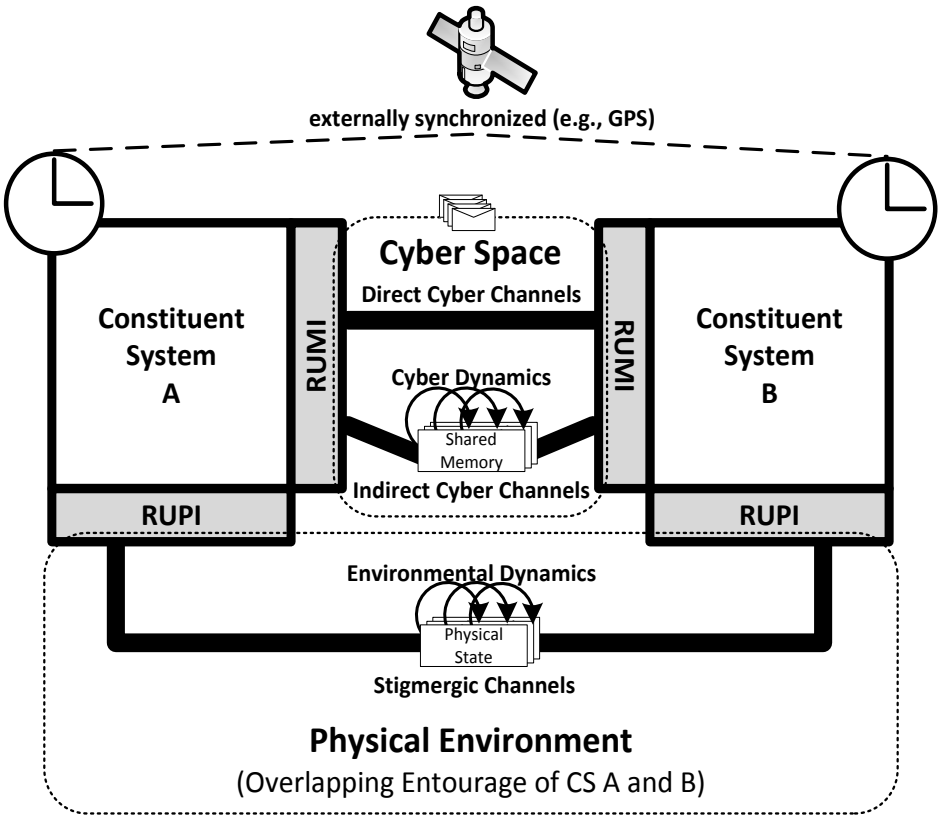

Figure 11: Information Transfer over Relied Upon Interfaces in Systems-of-Systems.

3.2.2 Cyber Space and Cyber Channels

A cyber channel describes direct or indirect CS interaction and is able to transport messages containing Itoms. We assume that cyber channels realized by computer networks (cyber space) have these properties:

- **Transport Any Type of Information:** In the most general case a message could contain information represented in natural language. There also is no restriction about the temporal relationship between Itom and the state of the present physical environment: a message can contain information about the past, present or future.

- **No Dynamics during Transportation:** Itoms carried by a cyber channel are 'frozen', i.e., the channel model does not realign the carried information to any physical environment. Naturally, a change of representation and explanation might occur (e.g., IP packets get encapsulated into an Ethernet frame), but the cyber channel itself has no influence on the carried Itoms.
- **Bounded Transport Duration:** In a time-aware SoS all cyber communication is bounded. If a cyber communication subsystem fails to transport a message within the given bound, the receiving CSs are still able to detect and tolerate the late or even missing message.
- **Multiple Contexts:** Sending and receiving CSs might adhere to different architectural styles such that Itom representations (data) need to be translated by gateway components.

Other requirements on cyber channels (for example maximum delay, the message size, or dependability) are derived from the Itoms that a cyber channel is supposed to transport.

We regard the cyber space as the set of mechanisms that realizes required cyber channels. For instance the IP-based Internet is a prominent example of a cyber space where any two systems with a unique IP address can establish channels.

3.2.3 Common Physical Environment and Stigmergic Channels

Stigmergic channels describe indirect interactions among CSs and exhibit different properties compared to cyber channels:

- **Modify and Observe Properties of Things:** A sender CS is restricted to influence properties of things in the environment by its actuator subsystem (e.g., increasing the temperature of a room by actuating a heating element). Environmental dynamics may act on the same properties of things (e.g., heat dissipation) and a receiver CS picks up Itoms by measuring this property. There is a tight temporal relationship between Itoms received or sent over a stigmergic channel and the present physical environment. Any actuation – when carried out after the actuator latency – affects the state of the environment immediately. Similarly, any measurement taken by a receiving CS is also about the present state of the environment minus the sensor latency.
- **Alignment by Environmental Dynamics:** Environmental dynamics both prevent that the same Itom placed in the environment can be measured by a receiver and also ensures that any Itom obtained by a receiver is closely aligned to the state of the physical environment. Stigmergic channels only allow for indirect means of CS interaction.
- **No Observation Delay:** The state of the physical environment can be immediately observed, regardless if it has already been influenced by another CS. Naturally the involved sensors have spatial and temporal limits w.r.t. transforming a measured property of a thing to data. These limits introduce uncertainties into the observation, which can be probabilistically quantified.
- **Unbounded Transport Duration:** Since the interaction is indirect the transport duration, i.e. the duration between the send instant and the receive instant of an Itom, cannot be bounded. For example, consider a single car, which is driving on a lane. Even though the car as a CS is sending Itoms related to movement over its RUPI into the environment, no other CS receives them. If eventually another car enters the lane, sensors of both cars might receive the other's car Itoms related to its respective changing position.
- **Single Context:** We assume that the physical environment offers only one universal context and all CSs that interact with the physical environment perceive it as the same for the interaction with intrinsic information, i.e., the information directly represented by the property of a thing. For example, the room temperature by itself (a specific environmental state) has universal meaning for all CSs that are able to measure or modify it. In case there is additional meaning attached (extrinsic information) to the property of a thing (e.g., warm

or cold room temperature), we have the same multiple context situation like in cyber channels.

Stigmergic channels do not directly transport an Item from one sending CS to a receiving CS. The sending CS merely applies a modification to the state of the physical environment where possibly many other physical processes including other CSs might constantly carry out other modifications. Consequently the receiving CSs are not able to measure exactly the state of the physical environment after the sending CS applied its modification. However, the received Item usually is related to a sent Item which justifies the use of the concept channel in the context of stigmergic CS interaction.

The design of sensors and actuators as well as their placement in the physical environment determines the semantics of the digital in-, and/or, output bit-pattern, i.e., effectively form a basic Item that contains the observation or actuation. In regard to information theory, we often have that some property of a thing (e.g., voltage level) has additional meaning to its receivers (e.g., digital zero and one), while the actual property of the thing becomes irrelevant, after the measured value has been properly abstracted. This refinement process takes place according to a receiver/sender shared conceptual context. It removes intrinsic information and produced a higher level Item that contains only the extrinsic information. In computer systems, such overlay-meaning needs to be added as meta data until an Item is formed which the target CSs can interpret and access correctly. The refinement process also removes unnecessary data that is not needed to convey the intended information, i.e., transforms raw object data to refined object data.

Take for example a speed limit sign at the side of the road which should be interpreted as such by an autonomous car CS. First, a camera sensor of the CS produces a bitmap Item where the road side including the speed limit is contained as a large array of pixels. Then, machine learning methods take this bitmap Item, segment the image, and finally extract an Item describing the speed limit sign. At this point the bitmap Item including its large object data becomes irrelevant and can be removed from the computer system memory. The new Item is then further contextualized with surrounding/implicit information (e.g., which metric or non-metric system the country where the road is located uses). Finally it is possible to construct the speed limit Item consisting of object data that represents a numeric value and meta data which explains (e.g., decimal, km/h, speed limit) the object data to be usable by the CS.

For engineering and testing RUPI interface specifications a model of the physical environment that can be simulated or evaluated with tools is desirable. There exist a large body of research [64][65] and many commercial products[3] to integrate the behavior of physical environment with cyber systems, which enables various simulation and verification methods.

3.2.4 Dynamicity of RUIs

Dynamicity (cf. Section 2.9.1) describes reaction or reconfiguration capabilities that have been already acknowledged during SoS architecture design time. Therefore, any supported dynamicity needs to be considered in the design of RUIs. We examine three prototypical dynamicity cases:

- dynamicity w.r.t. connecting two or more CSs at their RUIs,
- dynamicity in making (partial) CS or SoS services available to other CSs, and
- dynamicity to react to possibly sudden changes in the environment.

RUIs might be connected only for a finite duration within an Interval of Discourse (IoD). A disconnected RUI might be a normal, fault-free interface state and may result at most in system service degradation, but not in system failure. **This key aspect of RUIs is responsible for the impossibility to establish a static boundary of an SoS.** CSs may disconnect and reconnect and they might even be part of multiple SoSs (e.g., a modern day NFC enabled smartphone can be

[3] Mathworks Simulink, Maplesoft MapleSim, OpenModelica, Wolfram SystemModeler, xcos, …

part of the global telephone network, the Internet, and the global ATM network which are three large independently operating SoSs).

The RUI connection strategy is part of the interface specification of RUIs that regulates how CSs establish connections. All RUI connecting strategies are local to their respective CS. Still, they influence the dynamic global network topology of an SoS. Consequently, they are a crucial mechanism responsible for realizing self-organization and emergent phenomena.

Once two or more CSs are connected they can access services that are either available from a connected CS or one of the emergent SoS services. At the physical and/or message-based abstraction level of RUIs it is cognitively complex to describe the dynamic CS interaction that is necessary for accessing a service on the basis of single channels, because the specific client CSs are unknown before runtime. For example, take a CS that offers a database service. This database service requires a request and a response cyber channel per client CS. For n client CSs we would need to specify 2n dedicated channels at the message-based abstraction level. When we consider the same situation at the service abstraction level of interfaces, we only need to specify the database service together with the two required channels. The mechanisms of service discovery and service composition (cf. Section 2.13.3) allow a scheduler to automatically instantiate the request channel and the response channel for each client CS.

Finally, CSs might need to react to the changing environment and need to reconfigure the set of offered services in order to:

- accomplish overall SoS goals (e.g., limit total energy budget, enter a safe state, …), or
- tolerate faults (e.g., suddenly disconnected CSs or failing CSs).

At the service abstraction level of interfaces we can employ paradigms known from the Service-oriented Architecture (SoA) to replace services with a degraded version or to replace failed services altogether with services from other (redundant) CSs.

3.2.5 Evolution of RUIs

In case the SoS architecture design changes (e.g., refined requirements on SoS services), the affected RUI specifications and ergo the services of the involved CSs themselves require a modification as well. In contrast to dynamicity, evolution is a change in the SoS during runtime, which has not been planned for in the original SoS architecture design. Depending on the reconfigurability of the CS, the RUI modification is either carried out solely via the C-interface of the CS, which modifies the CS's configuration/software, or via a hardware update of the CS, or via a combination of both.

One insight gained from the conducted case studies in WP1 w.r.t. existing SoSs has been that changes in the SoS architecture designs have been carried out in a backward compatible way. CSs could coexist at different evolution versions, which allowed for a smooth and gradual update/replacement of CSs. Time-aware SoSs additionally permit non-backward compatible updates of RUI specifications at a predefined *validity instant*. Therefore temporal guarantees and an SoS available sparse global-time base enables sharper evolutionary steps which we consider useful for some SoSs. For example, in a forest fire monitoring SoS the CSs require an increased battery runtime to enable long maintenance cycles. One possible realization of this modified SoS goal is a transition to a more energy-efficient wireless communication protocol. We assume this can be done with a software update of the CSs via the C-interface. Then in an SoS with temporal guarantees and availability of a sparse global-time base we can switch at a specified validity instant to the new protocol without having to support the old protocol simultaneously. In fact supporting both protocols could even counteract to the new requirement of increased battery runtime.

3.2.6 Monitoring and Dependability Considerations

From the definition of SoSs we have to ensure that CSs are "*independent and operable"* (see Section 2.1.3). Hence, RUIs of a CS have to maintain a boundary for establishing a Fault

Containment Region (FCR) and an Error Detection Region (EDR) such that CSs fail independently from each other and errors do not propagate undetected from one CS to another.

4 SYSML REPRESENTATION

4.1 INTRODUCTION

According to the SoS conceptual model defined in Section 2 and Section 3, this section describes how the SoS concepts are formally translated using a semi-formal SysML language. The goals of this section are:

- describing why a semi-formal modeling language is useful to deal with the complexity of SoS development
- showing how a semi-formal language like SysML can be used to support description and analysis of an SoS.

First of all, a semi-formal modeling language is useful to improve the understandability of a problem. Using such language it is possible to abstract a problem thus focusing on particular points of interest by means of describing a system using independent views and levels of abstraction.

A semi-formal modeling language may also support the reduction of development risks and flaws by means of analysis and experimentation processes performed at early stages of the design cycle. In addition, by using a modeling language it is possible to improve the internal communication between the stakeholder, and to foster information sharing and reuse.

The semi-formal language we used to describe SoS concepts is SysML (System Modeling Language) [62], a general purpose visual modeling language for System Engineering applications and represents an enabling technology for Model-Based Systems Engineering (MBSE). SysML is defined as an extension of UML 2 by using the UML's profile mechanism. Through profiling it is possible to extend a reference metamodel (e.g. UML 2) by defining custom stereotypes, tagged values, and constraints. SysML provides an extension to UML in order to support modeling and analysis of system-level elements by means of specific stereotypes, (i.e., blocks) and their associations (e.g., generalization).

Different approaches using SysML have been proposed in order to describe and analyze particular aspects of SoSs. Nevertheless, a SysML profile that considers jointly our identified viewpoints has yet to be found in literature. Our goal in this section is to propose a SysML profile that, describing our main viewpoints of SoSs, could be used by system or software designer interested to describe and analyze the behavior of SoSs. By using our SoS profile and SysML language a designer could dynamically apply the SoSs concepts to its SysML model.

Furthermore, we have provided an example of profile application with the purpose of clarifying the usefulness of this profile. We have introduced an example that explains how a system designer can use our SoS profile for creating a preliminary interface analysis and discovering emergent behaviors of an SoS. In this case, our profile represents the starting point for an in-depth analysis or a very informative representation on which new system evaluations could arise. Finally, we emphasize that this is only one among the possible examples in which this profile can be used.

The following sections contain the description of SoS profile elements, their possible applications and the mapping with the concepts contained in Section 2 through a traceability matrix (see ANNEX C).

4.1.1 A review on existing profiles for Systems of Systems

Modelling of systems, or specific aspects of systems, has been carried out for many years in many different disciplines, and this has naturally spread to SoS engineering; as reported in [74], such models can be developed at different abstraction levels, depending on their purpose and the forms of analysis that are to be performed on them.

We present a viewpoint-driven analysis of related approaches presented in the literature of SoSs. This analysis, whose results are reported in Table 4, is not meant to be exhaustive but it is based

on the most representative related works on designing SoSs. Its objective is to determine at what extent viewpoints-based SoS concepts have been already captured in the literature.

Table 4: Viewpoint-based Literature Analysis of SysML Approaches to SoSs Design

	Structure	**Dynamicity**	**Evolution**	**Emergence**	**Time**	**Multi-criticality**	**Dep&Sec**
Huynh et al. [75]	✔	✖	✖	✖	✖	✔	✖
Lane et al. [76]	✔	✖	✔	✖	✖	✖	✖
Rao et al. [77]	✔	✔	✔	✖	✖	✖	✖
COMPASS [78][79][80]	✔	✖	✖	✔	✔	✖	✔
DANSE [81] [82]	✔	✔	✔	✔	✔	✖	✔

In [75] the authors propose the use of SysML in representing an SoS by adopting and in some cases extending canonical SysML diagrams in order to model different viewpoints of an SoS. In particular aspects related to the *structure* viewpoint have been deeply considered to identify the SoS internal structure, its boundaries with the environment through well-defined interfaces, SoS functionalities and how interactions occur by exchanging messages. Beyond *structure*, a specific support to the *multi-criticality* viewpoint is also provided by adopting the specific stereotypes aiming at grouping requirements according to qualitative and quantities metrics to support trade-off analysis. Nevertheless, in [75] a specific support to the *time* viewpoint has not been considered to assure the responsiveness of SoSs and *dependability/security* viewpoints have not specifically addressed. The authors did not consider viewpoints like *dynamicity*, *evolution* and *emergence*.

A partial answer to the above issues is given by the approach presented in [76] providing support to *structure* and e*volution* viewpoints of an SoS by exploiting several SysML models. The authors propose the adoption of diagrams to determine an evolving SoS and its environment and the interactions occurring between an SoS and the environment and among CSs themselves. Noteworthy the approach is still missing specific support to *dynamicity*, *emergence* and *multi-criticality* viewpoints and although the executable models presented are a first required step to assure *dependability/security* requirements and responsiveness (*time*), it still missing a specific support to those viewpoints.

In the SysML modeling approach presented in [77], the authors allow the definition of the SoS *structure* and how to support *dynamicity* and *evolution* viewpoints by means of understanding the dis-alignment of a simulated SoS with respect to its requirements. The approach makes use of different executable diagrams in order to simulate Net-centric SoS through the Petri Net formalism thus describing the dynamic behavior and assuring that end-user requirements are met. Noteworthy, the approach [77] is still missing a specific support to *emergence* and *multi-criticality*. Concerning *dependability* and *security* viewpoints as well as responsiveness (*time*), the approach [77], similarly to [76], does not provide any specific support.

The approach presented in [78], [79] in the context of COMPASS EU project [80], provides support to model the *structure* of an SoS and *emergence* by means of the extension to SysML diagrams. Analyses of the former models are conducted to evidence that requirements are fulfilled. COMPASS exploits tool's well established extension mechanisms to extend traditional systems modeling as needed to model and analyze SoS with the support of the formal COMPASS Modeling Language (CML). The latter has been exploited to support fault-handling (*dependability* viewpoint) and responsiveness (*time* viewpoint) of an SoS. Nevertheless, the approaches in [78][79] provide no specific support to *dynamicity*, *evolution* and *multi-criticality*.

The approach in [81], within the context of the DANSE EU project [82], supports the definition of an SoS *structure*, *dynamicity* and *evolution* (by means of Graph Grammars), *emergence,* etc., with the only exception of *multi-criticality.* DANSE presented a set of methodologies and tools to model and to analyze SoSs based on the Unified Profile for DoDAF and MoDAF (UPDM). In particular,

DANSE focuses on the six models that can be represented as executable forms of SysML as partially reported in [81], according to a well-defined formalism to relate basics SoS concepts and their relationships. In the context of DANSE, the Goal Specification Contract Language (GSCL) assures the achievement of dependability and security requirements and it guarantees the timely response of an SoS.

The following works are referred here because they focus on different viewpoints in SoS modelling, but they are not in the Table 4 as they do not provide a profile. The approach in [83] presents a theoretical model to describe autonomy, belonging, connectivity, diversity, and emergence in Systems of Systems. The work first models these characteristics and their properties, then it uses a computer simulation to demonstrate the presented model. SoSADL [84], currently under development, is a formal language derived from π-ADL and targeted to SoS architectures. SoSADL includes static and dynamic architectural specifications, with specific focus on reconfiguration modelling.

As shown, in the literature different attempts exist to apply SysML approaches to specific viewpoints that we deemed essential in supporting the design of SoSs. These approaches have shown the utility of adopting SysML formalisms to model architectural aspects of SoSs thus supporting different types of analysis and a first step towards executable artifacts which can be automatically derived. Although these approaches provide detailed insights for different viewpoints aspects, it is still missing (i) an homogeneous synthesis at a more abstract level of key design-related SoS concepts and (ii) and a viewpoint-based vision. Bringing this perspective in one single consistent reference model, it is possible to provide solutions to specific design problems while still keeping the required interconnections among viewpoints.

4.2 MAPPING RATIONALE

This section describes how the SoSs conceptual model, described in natural language in Section 2, could be mapped with its SysML representation. As suggested by project reviewers, we have selected a set of SoS concepts in order to highlight and represent only the main fundamental concepts following the viewpoint-driven approach introduced in Section 3. To this end, we have distinguished between the following:

- viewpoints describing architectural and behavioral basics concepts of an SoS, namely Structure, Evolution, Dependability and Security, Time and Multi-criticality;
- viewpoints representing the concepts of Dynamicity and Emergence of an SoS which can be captured observing an operational SoS.

In the former viewpoints all the concepts are represented through elements of a SysML profile, while for Dynamicity and Emergence, beyond using specific SysML profile elements, we made also use of a Smart Grid case study in order to apply formerly defined basics SysML elements thus capturing operational aspects of an SoS.

Our proposed SoS profile supports both system and software designer to describe all these viewpoints. In particular for each physical element or predictable behaviors we have mapped exactly one or more SoS profile elements. This profile is distributed in sub-packages that are fully described in Section 4.3. For Dynamicity and Emergence concepts, which are abstract and hard to predict, we adopt a methodology of representation and a running example beyond the definition of profile elements. This choice derived from the need of supporting designers in describing the topology of an operational SoS and in analyzing/anticipating SoS anomaly behaviors.

We introduce a Smart Grid household scenario to clarify our profile and to simplify the description of SoS anomaly behaviors. A Smart Grid household scenario represents a modernized residential electrical grid that uses information and communication technology to gather and act on information, such as information about the behavior of suppliers and consumers, in order to improve the efficiency and reliability of the production and distribution of electricity.

The Smart Grid household scenario is described as an SoS able to produce and to provide electricity to all electrical appliances (or CSs). Figure 10 shows a small example of a Smart Grid composed by the following components:

DER: Generic Distributed Energy Resource.

Smart Meter: This component provides production and consumption values.

Flexible Load: Represent the load that can be modulated.

EMG: Energy management gateway with the ability to control flexible loads. It is connected with a Coordinator in order to optimally tune the energy consumption.

Command Display: is a display showing consumption values and providing additional functionalities enabling consumers to interact with their own environment.

DSO: Specific Distribution System Operator of a specific region. It receives information from a Meter Aggregator that is a functional entity that provides access to one or more Local Network Access Point (LNAP) connected to the neighborhood network (NNAP).

Coordinator: An entity receiving energy prices and collecting energy flexibilities. It applies optimization functions in order to shift power consumption and generation when energy prices are favorable and it keeps balanced consumption and production values.

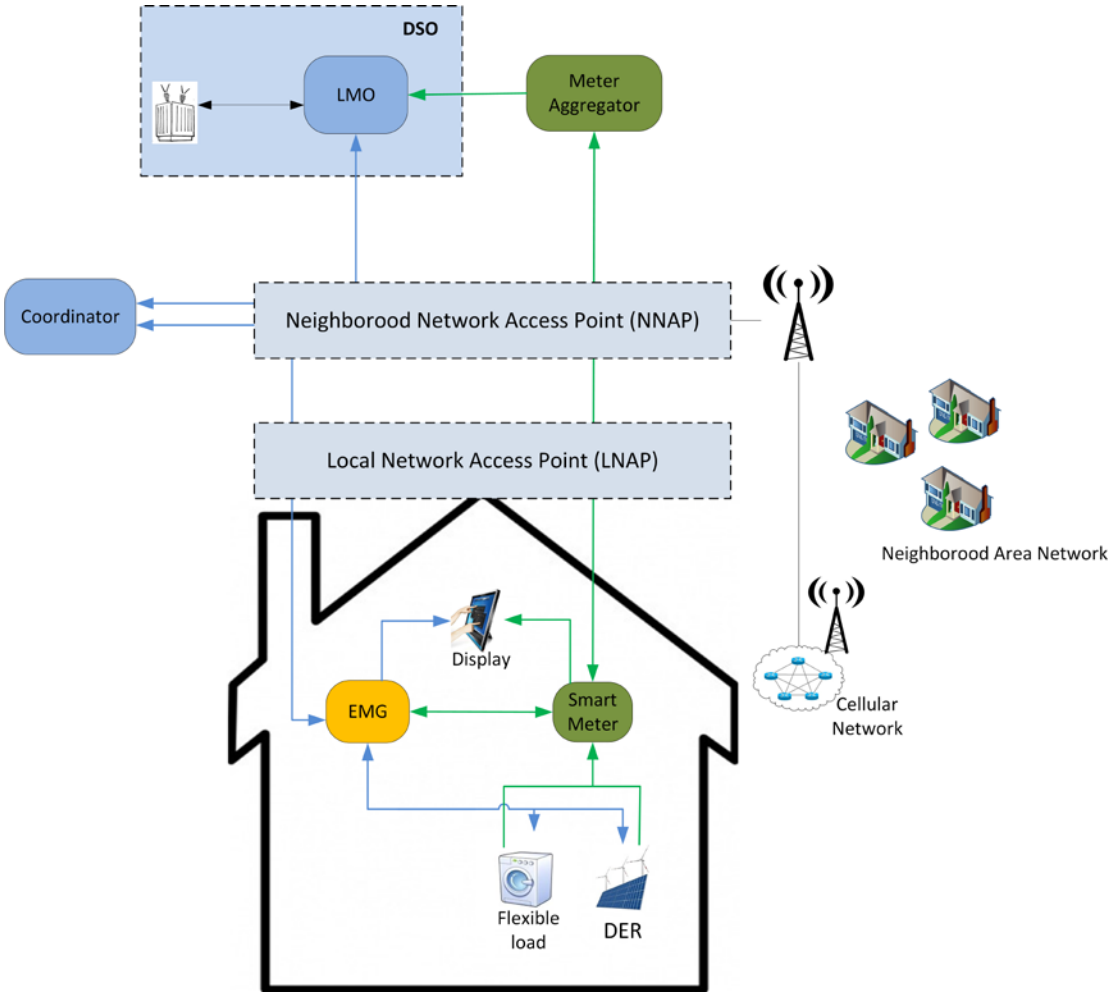

Figure 12: Smart Grid case study

4.3 PROFILE DEFINITION

We define the basic SoS concepts and their relationships within an SoS profile. This profile is built by taking into account common features and the modeling constructs offered by already available languages and standards like SysML and MARTE [56].

The SoS profile will be used as an abstract model to represent the topology and the state evolution of an operational SoS. The profile diagrams contain the SoS basic concepts distributed in sub-packages as follows:

- *SoS Architecture*: describes the basic architectural elements and their semantic relationships.
- *SoS Communication*: provides the fundamental elements in order to describe the behavior of an SoS in terms of sequence of messages exchanged among CSs.
- *SoS Interface*: describes all the points of integration that allow the exchange of information among the connected entities.
- *SoS Dependability*: provides the basic concepts related to SoS dependability.
- *SoS Security*: provides the basic concepts related to SoS security.
- *SoS Evolution*: provides the main elements to describe the process of gradual and progressive change of an SoS.
- *SoS Dynamicity*: provides basic concepts related to dynamicity of an SoS.
- *SoS Scenario-based reasoning*: provides the basic concepts for supporting the generation, evaluation and management of different scenarios resulting from SoS dynamicity, thus supporting decision-making in an SoS.
- *SoS Time*: provides the fundamental elements to describe time concepts.
- *SoS Multi-Criticality*: provide the basic concepts to describe the multi-criticality aspects of an SoS.
- *SoS Emergence*: provides the main elements to describe the SoS emergence concepts.

It is worth noticing that most of the above packages come from a direct mapping to the views previously defined except for *SoS Architecture, SoS Communication* and *SoS Interface* that all together implement the Structure view, and for *SoS Dynamicity* and *SoS Scenario-based reasoning* which map into the Dynamicity view.

We have implemented the whole profile by exploiting the Eclipse integrated development environment, jointly with Papyrus. Eclipse is an open source environment and offers all the related advantages in terms of cost, customizability, flexibility and interoperability. Papyrus is an Eclipse plugin, which offers a very advanced support to define UML profiles.

In the following sections, we will discuss the key elements belonging to the different packages. All the new introduced stereotypes extend the “Block” stereotype of SysML, if not differently specified. For the sake of readability, we will not represent such relations in the SysML diagrams describing the different packages.

4.3.1 Structural components

The structural properties of an SoS are described using three different packages “*SoS Architecture”*, “*SoS Communication”* and “*SoS Interface”*. The first defines Stereotypes useful to describe the topology of an SoS, whereas the second provides Stereotypes in order to describe the communication aspects between the Constituent Systems of an SoS. Finally, “*SoS Interface”* semi-formalizes internal and external points of interaction of an SoS.

4.3.1.1 SoS Architectural component

Architectural components are defined within "*SoS Architecture*" package (see Figure 13). This package extends SysML Block Definition Diagram (BDD) in order to model the topology and the relations of an SoS. Blocks in SysML BDD are the basic structural element used to model the structure of systems [58] and they can be used to represent: systems, system components (hardware and software), items, conceptual entities and logical abstractions. A Block is depicted as a rectangle with compartments that contain Block characteristics such as: name, properties, operations and requirements that the Block satisfies. A Block provides a unifying concept to describe the structure of an element or a system: System, Hardware, Software, Data, Procedure, Facility and Person.

This type of diagram helps a system designer to depict the static structure of an SoS in terms of its constituent system and possible relationships.

The first Stereotype is "**entity**" and it extends the SysML metaclass "Block". We distinguish between two different kinds of entities: "**thing**" or "**construct**". They extend the properties of "entity" and so they are also represented as Blocks.

A "**System**" is a type of entity (thereby a Block), it has the same characteristic but it is also capable of interacting with its environment. As it is expressed by the "**sys_type**" Enumeration, a system can be: autonomous, monolithic, open, closed, legacy, homogeneous, reducible, evolutionary, periodic, stateful and stateless. A system can be influenced by an "**Architectural style**" and can provide communication "**Interfaces**" and it has a "**boundary**". A "**Subsystem**" is a subordinate system that is part of a system and is related to "System" by a composite relation.

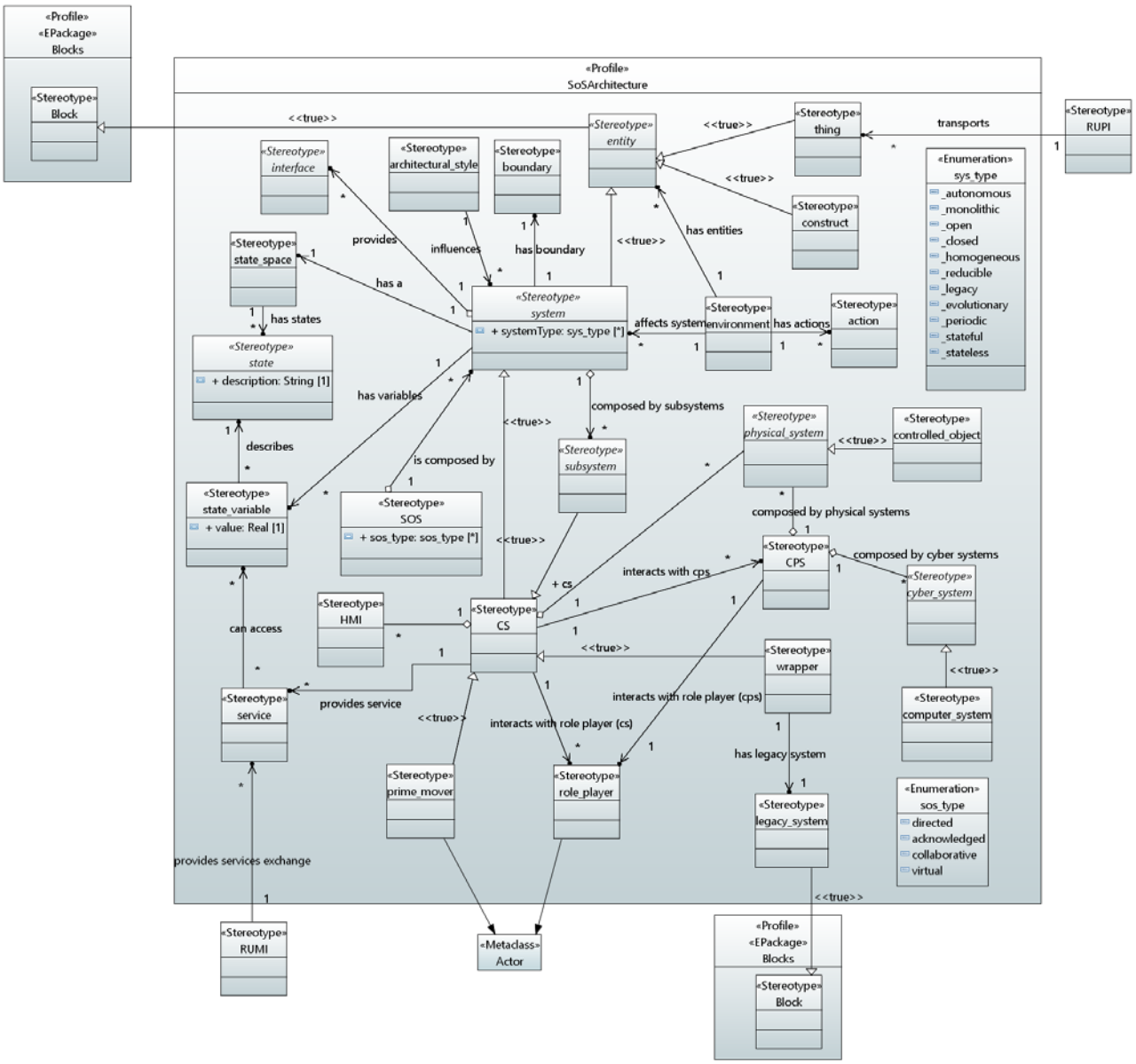

Figure 13: SoS Architecture package

As defined in Section 2, a Constituent System or "**CS"** is an autonomous subsystem of an SoS, consisting of human machine interfaces "**hmi**" and possibly of physical "**controlled_object**" and it provides a given "**service**" by interacting with "**role_player**" through the "**RUI**" (that is introduced in SoS Communication package). RUMI represents a message interface where the services of a CS are offered to other CS of an SoS and "**RUPI**" Stereotype represents a physical interface where things are exchanged among the CSs of an SoS. A "**wrapper**" to a "**legacy_system**" and a "**prime_mover**" are "**CS**". In this profile "cs" is a Stereotype that extends the property of "**system**", which contains multiple "sub_system", which in turn can be "CS". A system has a "**state_space**" composed of states described by the variables that may be accessed by the CS service. In addition, a CS interacts with cyber physical systems. Stereotype "**sos**" Stereotype represents the integration of systems, i.e., CS which are independent and operable, and which are networked together for a period of time to achieve a certain goal. An SoS can be directed, acknowledged, collaborative or virtual as it is expressed by the "**sos_type**" Enumeration.

A Cyber-Physical System ("**cps**") is composed by a set of "**cyber_system**" (i.e., computer systems), and "**physical_system**" (i.e., controlled objects).

Using this package it is possible to represent the topology of any System of Systems. Now we show how to use the SoS profile (in particular *SoS Architecture* package) through the Smart Grid household case study described in Section 4.2.

First of all, it is necessary to decide what are the main constituent systems involved, and how to represent them. For each system component we use a Block element of a Block Definition Diagram and through the connections we show the relations between them. Using the stereotypes defined in "*SoS Architecture*" package it is possible define the Smart Grid household as a system of systems ("sos") and all the other elements as constituent systems ("cs").

Figure 14 shows a model example of a Smart Grid with the application of our profile ("*SoS Architecture*" package described in Section 4.3.1). The "SG_Households" is a Block and it is stereotyped as a "sos"; it is composed by 5 "cs" (Constituent Systems), which exchange information. Among others, the block "Flexible Load" is stereotyped as a "cs" and it is composed by a set of household electrical appliances: Microwave, Washing Machine, Clothes Dryer, etc. These latter are switched on and off dynamically based on the current needs.

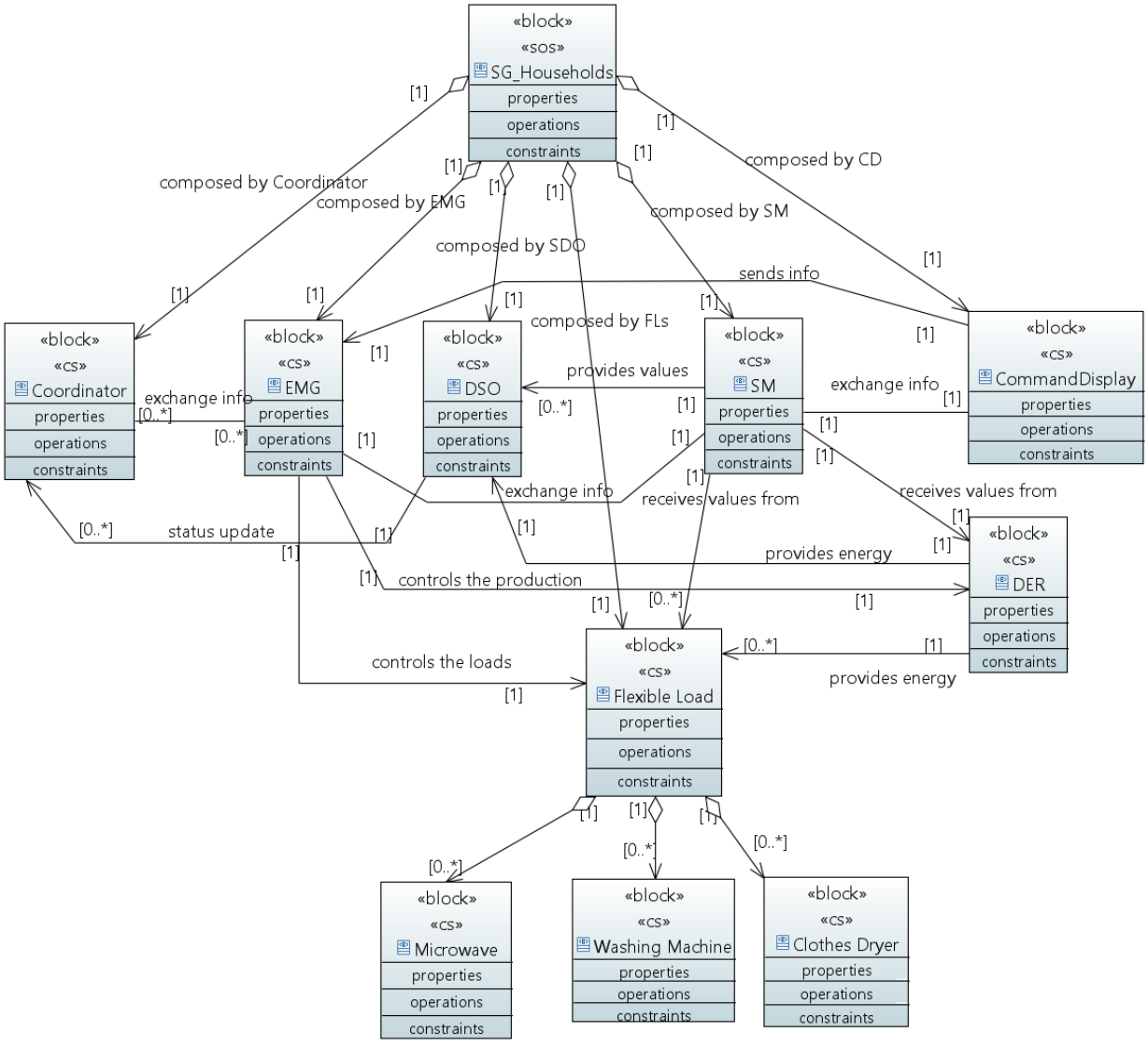

Figure 14: Smart Grid Household Block Definition Diagram

4.3.1.2 SoS COMMUNICATION COMPONENT

In an SoS the communication among the CSs by the exchange of messages is the core mechanism that realizes the integration of CSs. "*SoS Communication*" provides the SysML stereotypes in order to describe the main characteristics of communication protocols among CSs.

The "*SoS Communication*" package (see Figure 15) is composed of CSs that exchange information with other elements. In order to represent the exchanged information during the progression of time we use a SysML Sequence Diagram and we represent a CS not only as a Block entity of BDD but also as "Lifeline" metaclass. "Lifeline" is a metaclass and part of Sequence Diagrams. Through a Sequence Diagram it is possible to represent the behavior of a system in terms of a sequence of messages exchanged between parts and a "Lifeline" defines the individual participants in the interaction (Constituent System). Moreover through a "Lifeline" it is possible to describe the temporal behavior of an SoS. The time is showed by the length of the "Lifeline" and it passes from top to bottom: the interaction starts near the top of the diagram and ends at the bottom.

A "**RUI**" Stereotype represents an external interface of a CS where the services of a CS are offered to other CSs. It extends "**external_interface**" (defined in "*SoS Interface*" package) and guarantees the exchange of information among CSs ("cs" is defined in "*SoS Architecture*" package). A RUI can be represented also as a Sequence Diagram in which the CSs are represented by the lifelines that exchange information. A RUI can be either a "**RUMI**" or a "**RUPI**" and it is monitored through "**probes**". A RUI, having a "**connection_strategy**", is instantiated complying to possibly multiple "**dependability_guarantees**" and satisfying "**security**" constraints.

A "**RUMI**" represents a message interface for the exchange of information among two or more CSs and extends the "**RUI**" Stereotype. While messages are exchanged through the RUMI, physical elements are exchanged among the CSs of an SoS through the "**RUPI**"; physical elements are things or energy.

In this package we model the concept of a stigmergic channel. This type of channel transports information via the change and observation of states in the environment. In order to represent a stigmergic mechanism (as defined in Section 2.5), we have introduced the Stereotype environment (already shown in *SoS Architecture*) which is affected by the RUI.

As defined in Section 2, a message is a data structure that is composed by a "**data_field**", a "**header**" and a "**trailer**" and it flows through a "**transport_service**". The main transport protocol classes to send a message from a sender to a receiver are listed in the "**transport_service**" Enumeration data type, i.e., "**datagram**", "**PAR-Message**" and "**TT-Message**". A message can be classified as valid, checked, permitted, timely, correct or insidious.

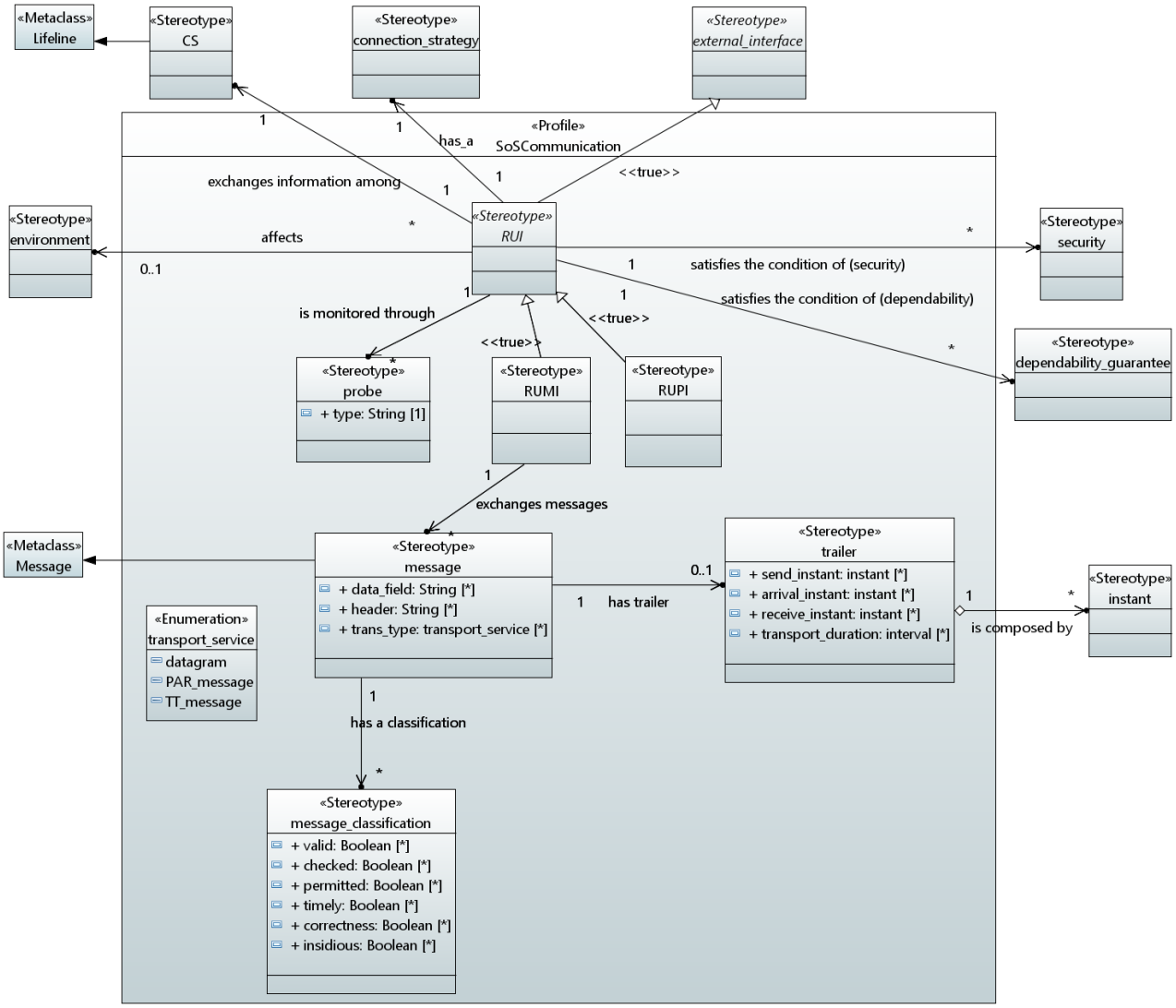

Figure 15: SoS Communication package

An application example of the main SoS communication concepts are shown in Figure 16. Through the Smart Grid household case study we describe a set of communication messages exchanged between the involved constituent systems (“cs”).

First of all, it is necessary decide which are the involved elements in the communication and how many message are exchanged. We identify a “Lifeline” as a constituent system and a “message” as exchange data between two constituent systems. A message could contain all the properties defined in Figure 15 and they can be displayed using a constraint or a comment box. Figure 16 shows the message properties using a comment box (e.g., “data_field”=2kW, “header”=wm to EMG, “trailer”=t1, “trans_type”=PAR_message).

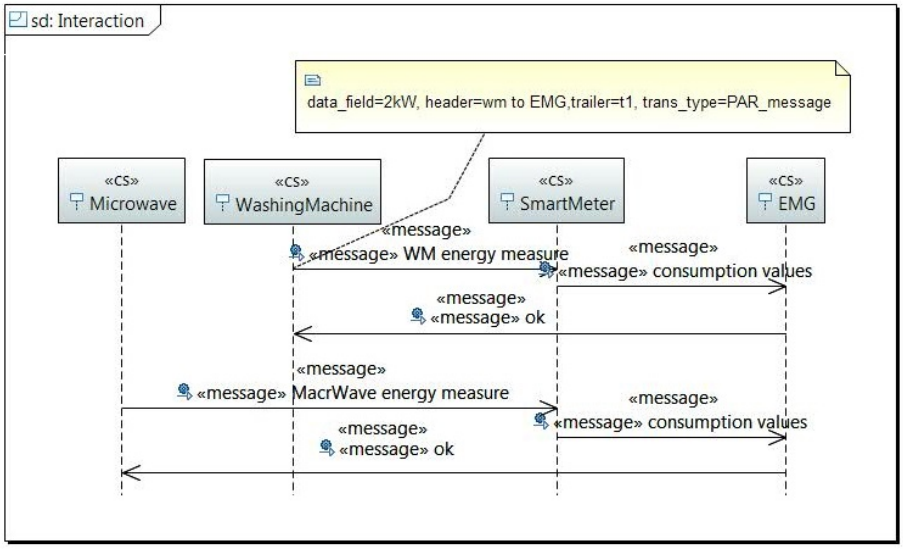

Figure 16: Message exchange between constituent systems of a Smart Grid

4.3.1.3 SoS Interface component

The interfaces are the key issue to the integration of systems and in this section we introduce an in-depth analysis of the SoS interface concepts. Figure 17 shows the SoS Interface Package.

An interface (described within the *SoS Architecture* package) can be an "**internal_interface**" a "**physical_interface**", a "**message_based_interface**" and an "**external_interface**". The internal interface connects two or more subsystems of a CS (the Stereotype "subsystem", defined in *SoS Architecture* package, is connected with "internal_interface" in order to represent this relation). The physical interface consists in three different type of elements, namely "**sensor**", "**actuator**" and "**transducer**". The "**message_based_interface**" allow the transmission of message" by means of "**message**" which are defined in terms of "**message_variable**". Finally, the external interface connects two or more CS ((the Stereotype "CS" is connected with "**external_interface**"). A different type of "external_interface" is "**utility_interface**" that aims to configure, update, diagnose the system and interacts with its remaining local physical reality. So the utility interface is specialized into three different types of interfaces: Configuration and update interface "**c-interface**", an interface for diagnosis "**d-interface**" and a local I/O Interface "**local_IO_Interface**". The latter uses sensors, actuators and transducers, which do not need to be considered for the integration of SoSs thus they hare hidden behind RUIs.

An interface has a specification ("**interface_specification**") with different kind of levels: Interface Cyber-Physical Specification ("**cp-spec**"), Interface Itom Specification ("**i-spec**"), Interface Service Specification ("**s-spec**"). "**cp-spec**" is extended by "m-spec" which specifies interface properties related to cyber message. "**m-spec**" is further extended by the "**transport_specification**" Stereotype to describe all properties of the communication system for correctly transporting a message from the sender to the receiver(s). "cp-spec" is also extended by "p-spec" which specifies the interfaces properties related to physical interactions. If the interfaces are service-based, this means that the system provides many services. We have introduced the Stereotype "**SLA**" Service Level Agreement which defines the service relationship between two parties: the "**provider**" and the "**recipient**". "**SLA**" consists of one or more "**SLO**", i.e., the Service Level Objectives (see Section 2.13.3). In addition we have created a new Stereotype that represents the "**reservation**". The reservation is a commitment by a service provider that a resource that has been allocated to a service requester (upon request at "**request_instant**") at the reservation allocation instant ("**allocation_instant**") will remain allocated until the reservation end instant ("**end_instant**"). A "**registry**" contains multiple service specifications allowing multiple "**service_composition**" according to the "**SLA**".

A “**channel**” connects interfaces, and it can be physical or logical (“**physical_channel**”, “**logical_channel**”). The interaction enabled by the channel has the following attributes described in Section 2: ”**transferred_info**”, “**temporal_property**” and “**dependability_req**”. Through channel interactions, the information is transmitted by means of messages”. A “**channel_model**” describes the effects of the channel on the transferred information.

An “**interface_model**” contains the explanation of the interface. An interface, associated to an “**interface_port**” has an afferent and an efferent “**interface_model**” which are affected and may affect the interface, respectively.

A “**connection_strategy**” Stereotype is defined and connected to a RUI.

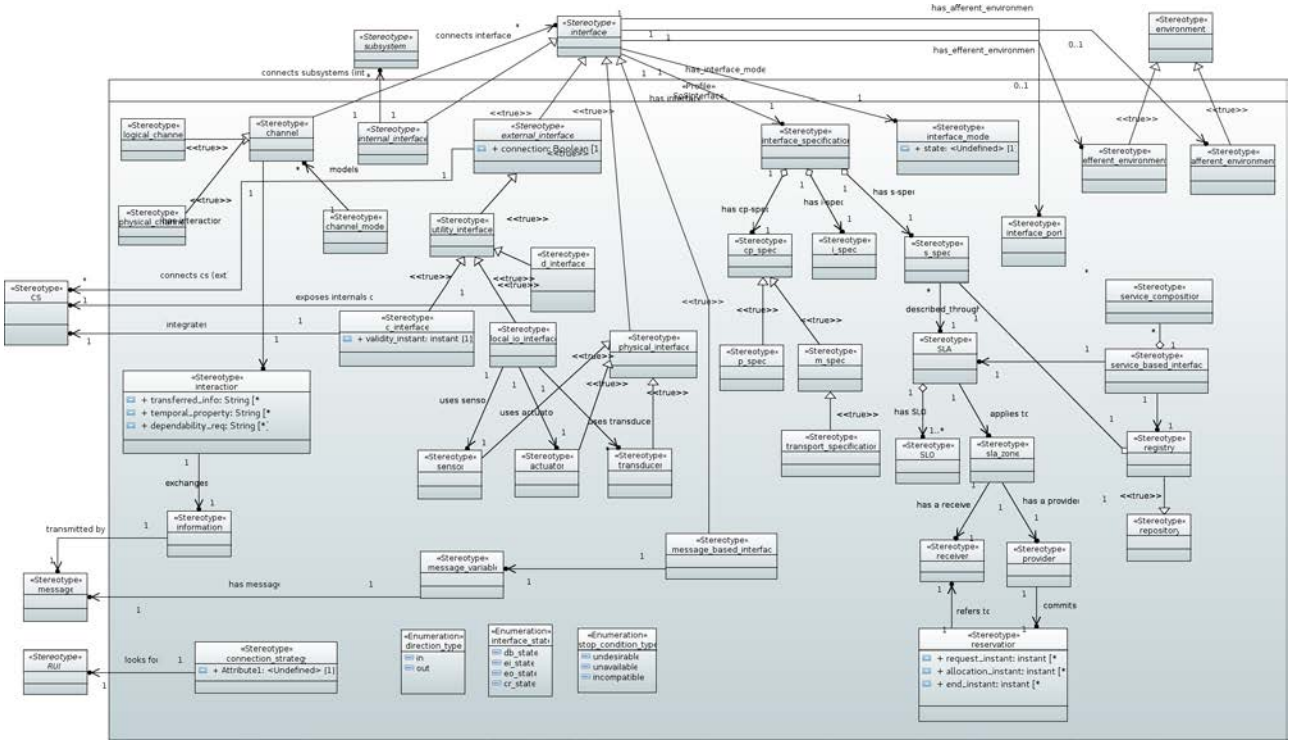

Figure 17: SoS Interface Package

4.3.2 Evolution and Dynamicity components

The Evolution and Dynamicity components are described by means of three different packages, i.e., "SoS Evolution", "SoS Dynamicity" and "SoS Scenario-based reasoning". As far as dynamicity is concerned, we also make use of commonly adopted interaction diagrams to represent the dynamic behavior of an SoS.

4.3.2.1 SoS Evolution component

As defined in Section 2, Evolution is a process of gradual and progressive change or development, resulting from changes in its environment (primary) or in itself (secondary). It does not have a positive or negative direction, evolution refers to maintaining and optimizing the system.

In order to describe this type of processes we have chosen a Block Definition Diagram, because it is designed to show the generic characteristics and structures of a system.

The main SoS concepts are modelled within the “*SoS Evolution*” package of our SoS profile. Figure 18 shows the “**evolution**” Stereotype as a Block of a BDD, aiming at describing an SoS change. In our concept model we envision two different types of evolution: “**managed_evolution**” and “**unmanaged_evolution**”. An SoS evolution has a “**goal**”, improves the “business value” by

means the exploit of a "**system_resources**" and can be affected by the environment. Evolution is achieved by modifying CSs and consequently the whole SoS.

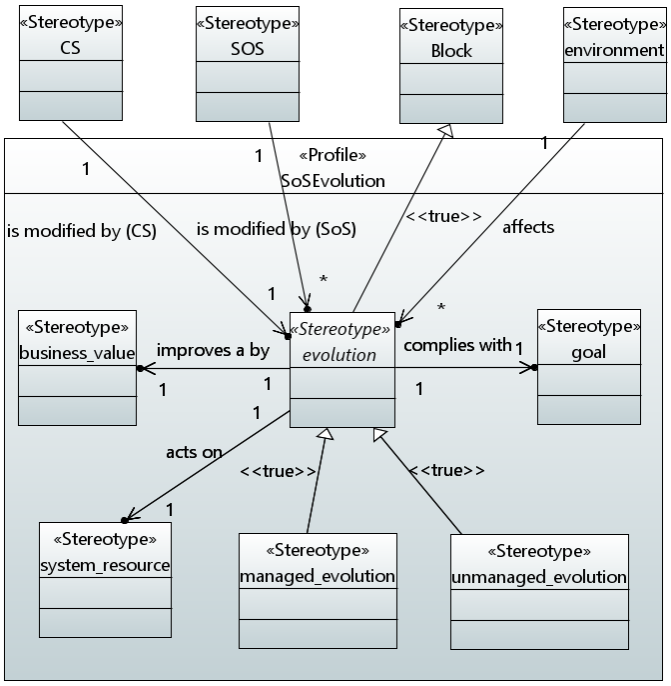

Figure 18: SoS Evolution package

The evolution concept can be easily shown by exploiting the smart grid example described in Section 4.2. Let us supposing to replace the Command Display (depicted in Figure 19) with a new device (e.g. a Smartphone) by which it is possible showing consumption values and providing new interactive actions.

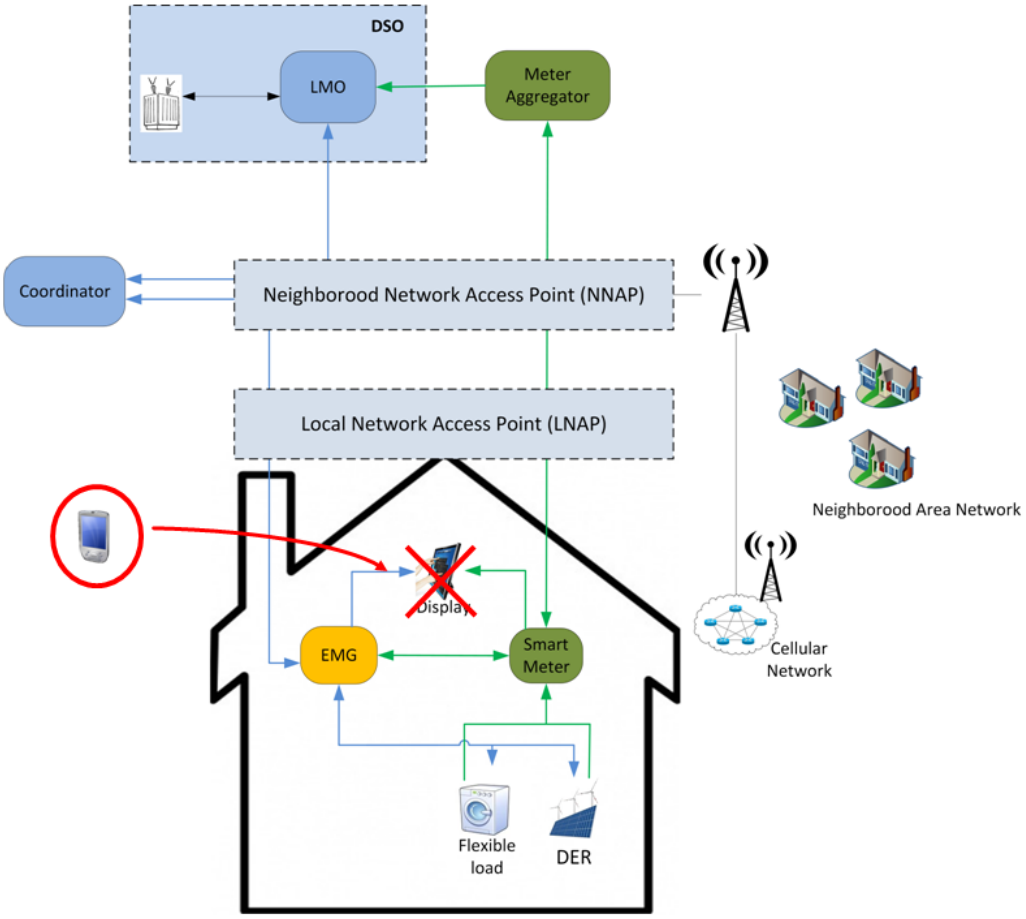

Figure 19: Example of system evolution in a Smart Grid Household

Figure 20 describes the evolution behavior of such a household Smart Grid. The evolution consists in modifying the whole SoS "SG_Household", to support the introduction of a new technology ("NewAvailableTechnology" stereotyped as a "goal"), thus maximizing the "Usefulness" which represents "Business value" of interest. With this aim the evolution acts on a new system resource by replacing the "CommandDisplay" with the "Smartphone".

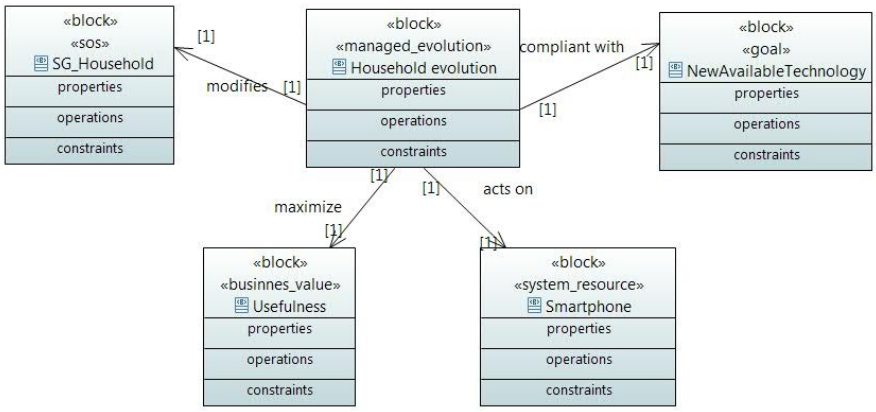

Figure 20: Smart Grid evolution modeling

The "*SoS Evolution*" package makes it possible to describe SoS evolution behavior in order to help the system and software designers to reflect on how a progressive change or development can impact on the SoS.

4.3.2.2 DYNAMICITY COMPONENT

Dynamicity is the property of an entity which constantly changes in term of offered services, built-in structure and interactions with other entities.

In this section we show how to use a semi-formal language in order to represent the dynamicity of of an SoS. Our objective is to (1) identify which parts of an SoS are dynamic at a certain extent and (2) to represent the dynamic behavior through the interactions among CSs.

As presented in Figure 21 we have introduced the concept of **"dynamicity"** (belonging to the already defined stereotype **"entity"**), which can be applied either to a CS or to a whole SoS. Dynamicity may be of different nature, either **"dynamic_service"**, or **"reconfigurability"**, i.e., the variation to the CSs architecture, or **"dynamic_interaction"**. Already defined concepts like **"service"** and **"interaction"** are the object of a dynamic behavior.

Eliciting dynamicity behavior of different nature that applies to different portions of an SoS, is not enough to have a full understanding of the dynamic behavior. With this aim, along with the dynamicity package, we have considered interaction diagrams in order to focus on the message interchange between a number of lifelines: Sequence Diagrams. We propose a methodology to be used to represent dynamicity as it follows:

- Making use of Sequence Diagrams to represent the system behavior in terms of a sequence messages exchanged between parts.
- Selecting the constituent systems involved in the communication.
- Describing the most common interactions.

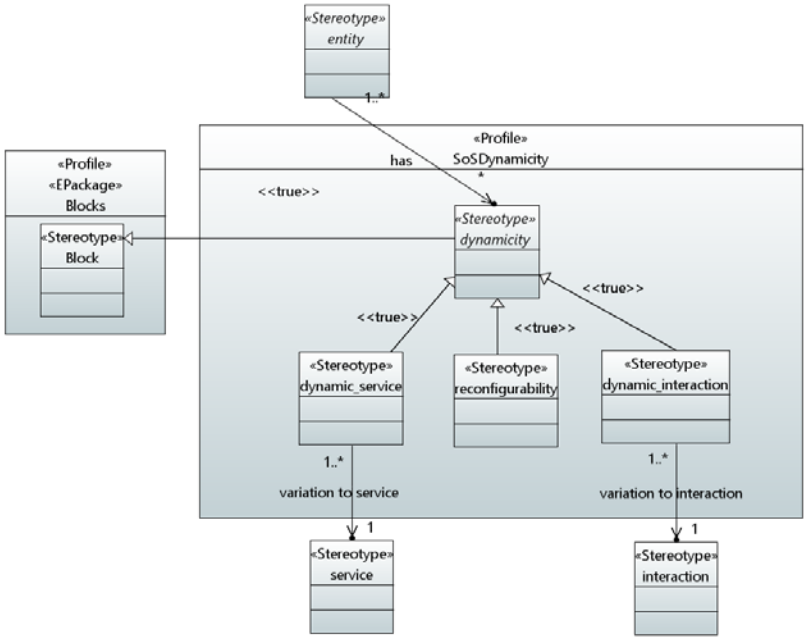

Figure 21: SoS Dynamicity package

This type of representation helps a system designer to understand which are the properties of an SoS that are constantly changing and how the SoS can change and rearrange its components. The dynamic introduction, modification or removal of constituent systems can introduce new system behaviors that need to be analyzed.

We have chosen of modeling the dynamicity aspects through sequence diagrams by making use of the Smart Grid case study already introduced in Section 4.2. Using "SoS Communication" we

represent the dynamicity concept by showing a possible scenario to support the provision of energy in the Smart Grid (see Figure 22). Let us assume that a household electrical appliance has to be switched on. It sends a request of energy to the EMG. The latter receives, periodically aggregated consumption and production energy values from the Smart Meter. Nevertheless the EMG within the household is not able by itself to determine how much energy can be taken from the Smart Grid. To this end, the EMG forwards the request (with currently available production and consumption values) to the Coordinator entity which is connected to the neighborhood network with the aim of keeping as much as possible balanced the production and the consumption of energy for a set of connected households (i.e., the neighborhood). According to the information received by the EMG and based on the current global consumption and production values of the neighborhood, the Coordinator entity decides if the request of energy can be satisfied. The EMG receives a positive or a negative acknowledge from the Coordinator entity and it forwards the answer back to the electrical appliance.

Noteworthy, the Smart Meter measures the consumed and produced energy from the household appliances (i.e., loads and DERs) and evaluates the consumption/production values. These values are provided to the EMG in order to let it monitor consumption/production of the household and check if these values are coherent w.r.t. the ones agreed with the coordinator. In case of a overconsumption or overproduction it can send a command to turn some appliance/DER off, in order to satisfy the agreed thresholds dictated by the Coordinator.

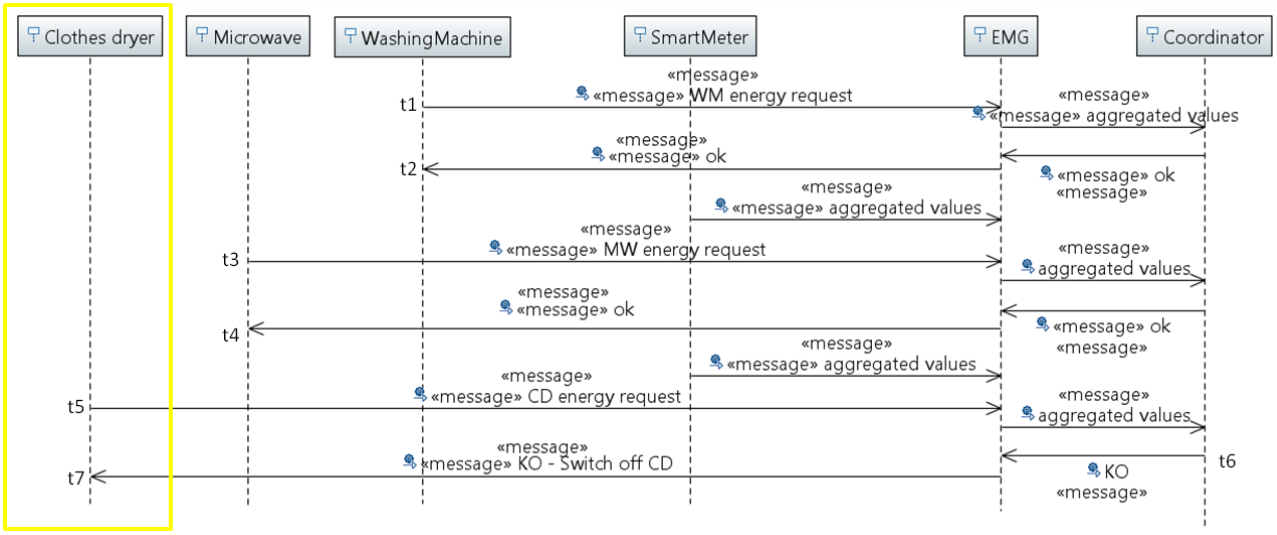

Figure 22: Dynamicity behavior of a Smart Grid

The illustrative example in Figure 22 represents a case of SoS dynamic behavior. The household electrical appliances want to be switched on one at a time, sequentially from t1 to t7. This example represents the continuous reconfiguration of the grid performed by the household electrical appliances with the support of the Coordinator entity, which satisfies energy requests according to the current global consumption and production values. Figure 22 shows that if the current energy load can become unbalanced by allowing the clothes dryer to be switched on (as requested at time t5), the Coordinator entity can decide to prohibit the clothes dryer to be connected at time t6 and t7. In the figure we have highlighted the only electrical appliance, which cannot be switched on according to the message received by the gateway EMG.

4.3.2.3 SoS Scenario-based reasoning component

Scenario based reasoning package aims at supporting dynamicity and evolution of an SoS. By means of this component of the profile we aim at supporting the generation, evaluation and management of different scenarios thus supporting decision-making in an SoS. As shown in Figure 23, the main concept of this component is **"scenario"** which is composed by a set of **"scenario_state"** each of which associated to an **"event"** to be applied at each state. A state is in

instantiation of a set of **"variables"** which are relevant for the decision-making. Such variables can be extracted by means of an **"inference_process"** and they pertain to a **"domain_model".** The latter defines relationships among variables in terms of correlations (**"causal_model"**) and causation (**"causal_graph"**) dependencies.

The process of generating scenario results from the **"situation assessment"** which depends on the **"environment"**. **"decision_making"** is the process to select a course of action among different possible alternate scenarios. A multi-criteria decision analysis **"mcda"** may be also applied to improve the decision-making process. Finally, scenarios are subject to pruning and updating operations in order to discard non-correct or un-likely scenarios and to update scenarios dealing with newly available information.

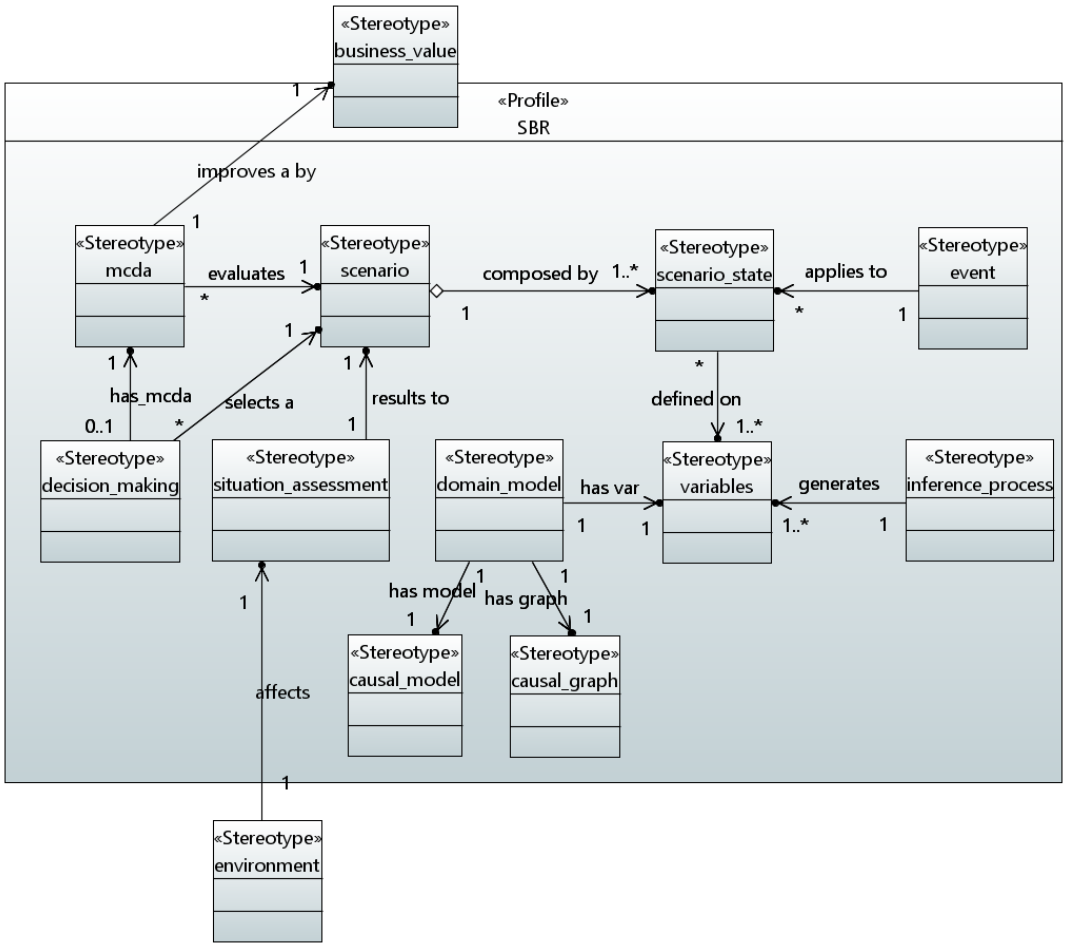

Figure 23: SoS Scenario-Based Reasoning (SBR) package

4.3.3 Dependability and Security components

This chapter describes how we have modeled through a SysML profile the basic concepts related to dependability and security. Dependability and security are important properties for System of Systems since they impact availability, reliability, maintainability, safety data integrity, data privacy and confidentiality.

We describe two different packages "*SoS Dependability*" and "*SoS Security*". The former defines Stereotypes useful to describe dependability concepts of an SoS and the second focused on specific security concerns. The terminology is based on the definition provided in Section 2 and it is in line with the canonical definition of dependability and security concerns as defined in [4].

4.3.3.1 DEPENDABILITY COMPONENTS

In our profile we define the "*SoS Dependability*" package by restricting the focus on main concepts defined in Section 2. The latter have been selected to support system analysis and design.

Figure 24 shows the key concepts captured within the dependability package. A CS or a whole SoS may require possible multiple "**dependability_guarantee**" through the achievement of possible different dependability "**metric**" by means of possible different "**technique**".

A technique is exploited to reduce the occurrence of faults:

- "**fault_prevention**" represents a particular technique aiming to prevent the introduction of faults during the development phase of the system.
- "**fault_tolerance**" represents a particular technique aiming to avoid the occurrence of failures by performing error detection and system recovery over time.
- "**fault_removal**" represents a particular technique aiming to remove faults during the development phase, by performing verification, diagnosis and correction and during operational life, by performing corrective and preventive maintenance actions.
- "**fault_forecast**" represents a particular technique aiming to forecast faults by performing an evaluation of system behavior.

A "**measure**" represents a property expected from a dependable system expressed in terms of a quantitave "**target_value**":

- "**availability**": this element represents a dependability measure that quantifies the alternation between deliveries of proper and improper service.
- "**reliability**": this element represents a dependability measure of the continuous delivery of service.
- "**maintainability**": this element represents a dependability measure of the time to restoration from last experienced failure.
- "**safety**": this element represents a dependability measure of the time to catastrophic failure.
- "**integrity**": this element represents the measure that quantifies the absence of improper system state alterations.
- "**robustness**": this element represent a dependability mesure with respect to external faults (including malicious external actions)

The profile supports the definition of a "**fault_containment region**", "**error_ containment**" and "**error containment_region**". The first contains components operating correctly regardless of any arbitrary fault outside the region. These components may have erroneous output actions that are alleviated with the definition of "**error_containment**", which prevents propagation of errors by employing error detection and mitigation strategies. This leads to the definition of "**error_containment_region**" which contains more "**fault_containment region**" having "**error containment**". A Fault Containment Region (FCR) is a collection of components that operates correctly regardless of any arbitrary fault outside the region.

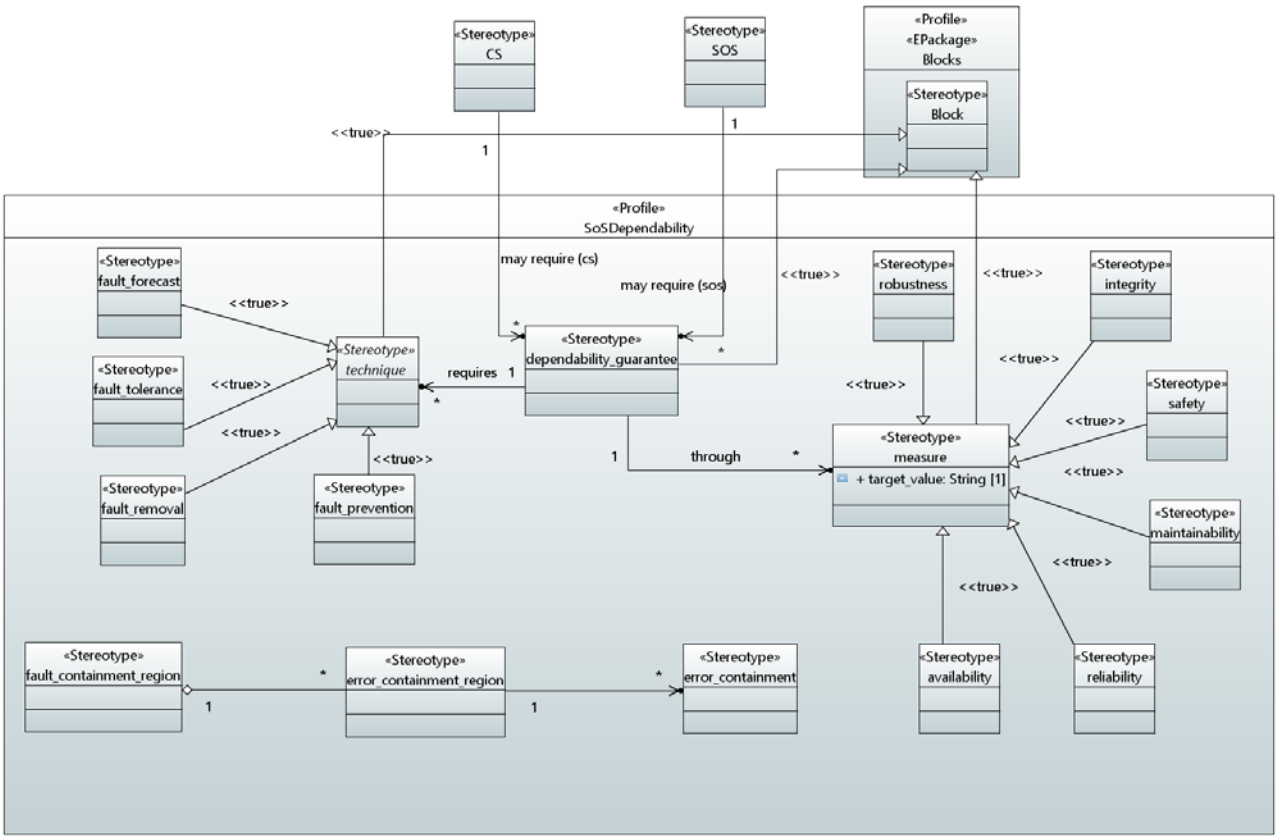

Figure 24: Dependability conceptual model.

4.3.3.2 SECURITY COMPONENTS

This section describes the fundamentals elements used by a system designer to represent security aspects of an SoS.

"*SoS Security*" package shows a set of security concepts as defined in Section 2. As shown in Figure 25, we connect the Stereotype "**SOS**" and "**CS**" to "**security**" Stereotype to satisfy the security conditions of an SoS. To this end we use "**cryptography**" based on symmetric ("**symmetric_cryptography**") or public key ("**public_key_cryptography**") infrastructure.

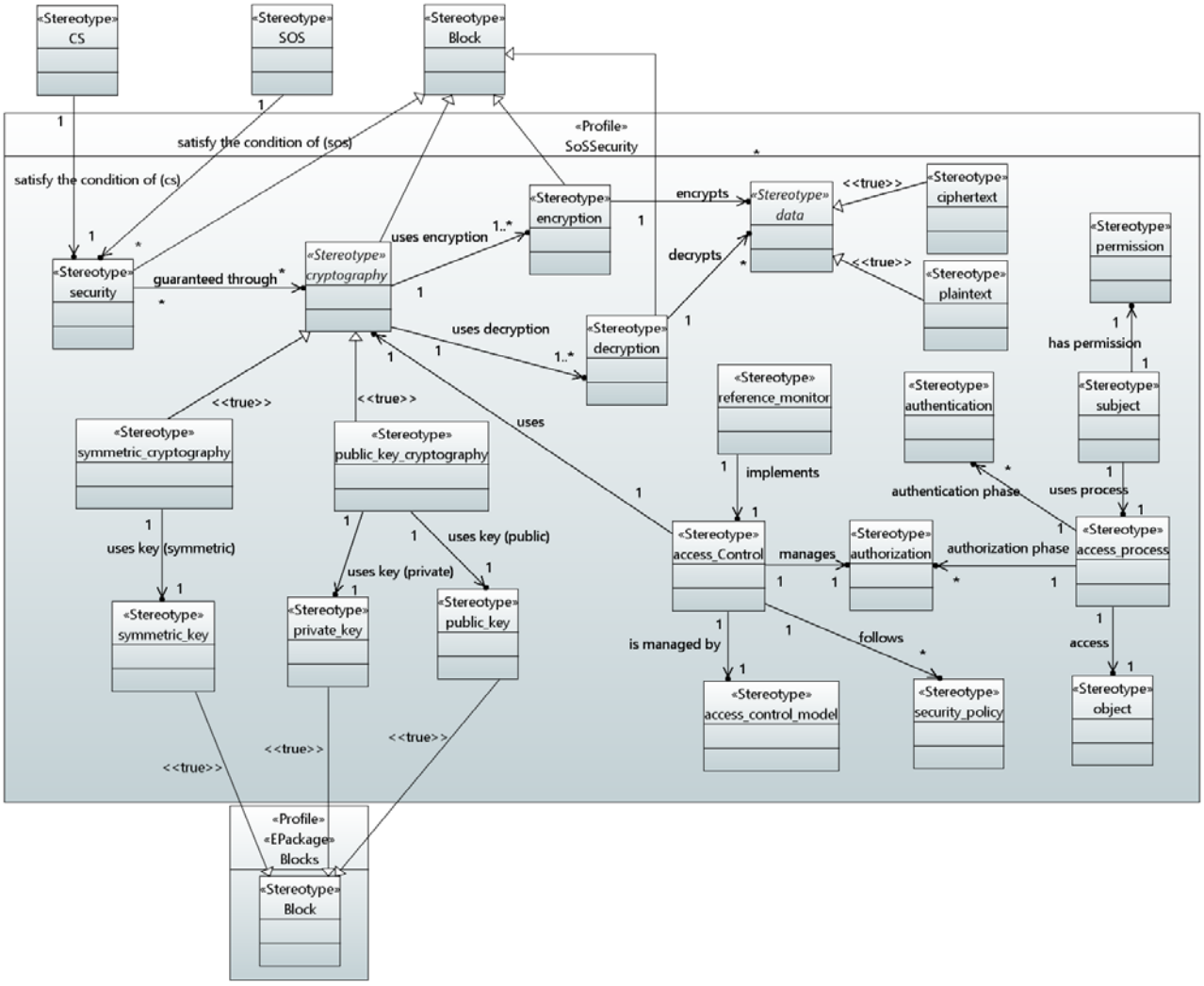

Figure 25: SoS Security package.

The “**encryption**” Stereotype represents the process of encoding information or data in such a way that attempts to hide the information from possible interceptors. In this way data exchanged between Constituent Systems are processed using a cryptography key (three types of key have been represented: “**symmetric_key**”, “**private_key**” or “**public_key**”). Symmetric key is exploited for symmetric cryptography, while private and public keys for the public-key critpography.

The information exchanged (also called “**data**”) can be encrypted (“**ciphertext**”), or not encrypted (“**plaintext**”); the “**decryption**” Stereotype represents the process of turning ciphertext to plaintext.

During the cryptography phase the access control (“**accessControl**”) consists in a set of actions that are permitted or not allowed by the system. Figure 25 shows a “**subject**” that represents an active user, a process or a device that causes information to flow among objects or changes the system state. A subject may have attributes (“**permission**”) that describe how the subject can access to objects. An “**object**” is a passive system-related devices, files, records, tables, processes, programs, or domain containing or receiving information. Access to an object implies access to the information it contains. The “**accessProcess**” is composed by the “**authentication**” and the “**authorization**”. The former represents the process of verifying the identity or other attributes claimed by or assumed of a subject or verifying the source and integrity of data. The latter represents the mechanism of applying access right to a subject.

The “**referenceMonitor**” represents the mechanism that implements the access control model and the “**accessControlModel**” captures the set of allowed actions within a system as a policy. The

access control follows a "**securityPolicy**" that represents a set of rules that are used by the system to determine whether a given subject can be permitted to gain access to a specific object.

4.3.4 Time component

Progression of time is one of the main topic investigated in AMADEOS. We express the time-related concepts by adopting the MARTE standard [56]. MARTE is an UML profile that provides support for non-functional property modelling, defines concepts for software, hardware platform modelling and concepts for quantitative analysis (e.g. scheduling, performance).

We measure time through clocks by defining a clock stereotype which extends the one defined in the MARTE profile. A MARTE Clock Stereotype is considered as a means to access to time, either physical or logical. The MARTE Clock is an abstract class and it refers to a discrete time.

Figure 26 shows a set of main time aspect defined in Section 2. A Constituent System (defined in *SoS Architecture* package) can share a clock. The Stereotype "**clock**" is also defined as a SysML Block in order to model this concept through a Block Definition Diagram. A "**timeline**" represents the progression of the time and it is designed with a Stereotype that extends the metaclass "Lifeline" of a Sequence Diagram. The "timeline" is composed by an infinite number of instants ("instant" Stereotype) measured using a "**time_code**" and a "**time_scale**". A "clock" could be based on an "**internal_sync**" or an "**external_sync**", it could be a "**reference_clock**" or a "**primary_clock**" and it could have the following properties from Section 2:

- "**accuracy**"
- "**granularity**"
- "**tick**"
- "**offset**"
- "**frequency_offset**"
- "**stability**"
- "**wander**"
- "**jitter**"

If a clock is a physical clock, we use the "**drift**" measure in order to describe the frequency ratio between the physical and the reference clock. A digital clock consists of an "oscillator", represented as a Stereotype, with a "**nominal_frequency**" and a "**frequency_drift**", represented as properties. A "**coordinated_clock**" is a particular type of a clock, it is synchronized within stated limits to a reference clock. A "**clock_ensamble**" is a collection of clocks operated together in a coordinated way with a certain "**precision**". We also represent GPSDO as a particular type of clock where its time signals are synchronized with information received from a GPS receiver. "**gpsdo**" is the Stereotype that represents this type of clock and "**holdover**" is a property that expresses the duration during which the local clock can maintain the required precision of the time without any input from the GPS.

The "**timestamp**" is the state of a selected clock at the instant of event occurrence. It depends on selected clock and if we use the reference clock for time-stamping, we call the timestamp "**absolute_timestamp**". An ensemble of clocks could synchronize in order to establish a "**global_time**" with a bounded precision.

An "**instant**" is a cut of the "**timeline**" and an "**interval**" is a section of timeline composed by two instants. The latter is defined as an "IntervalConstraint" of a Sequence Diagram.

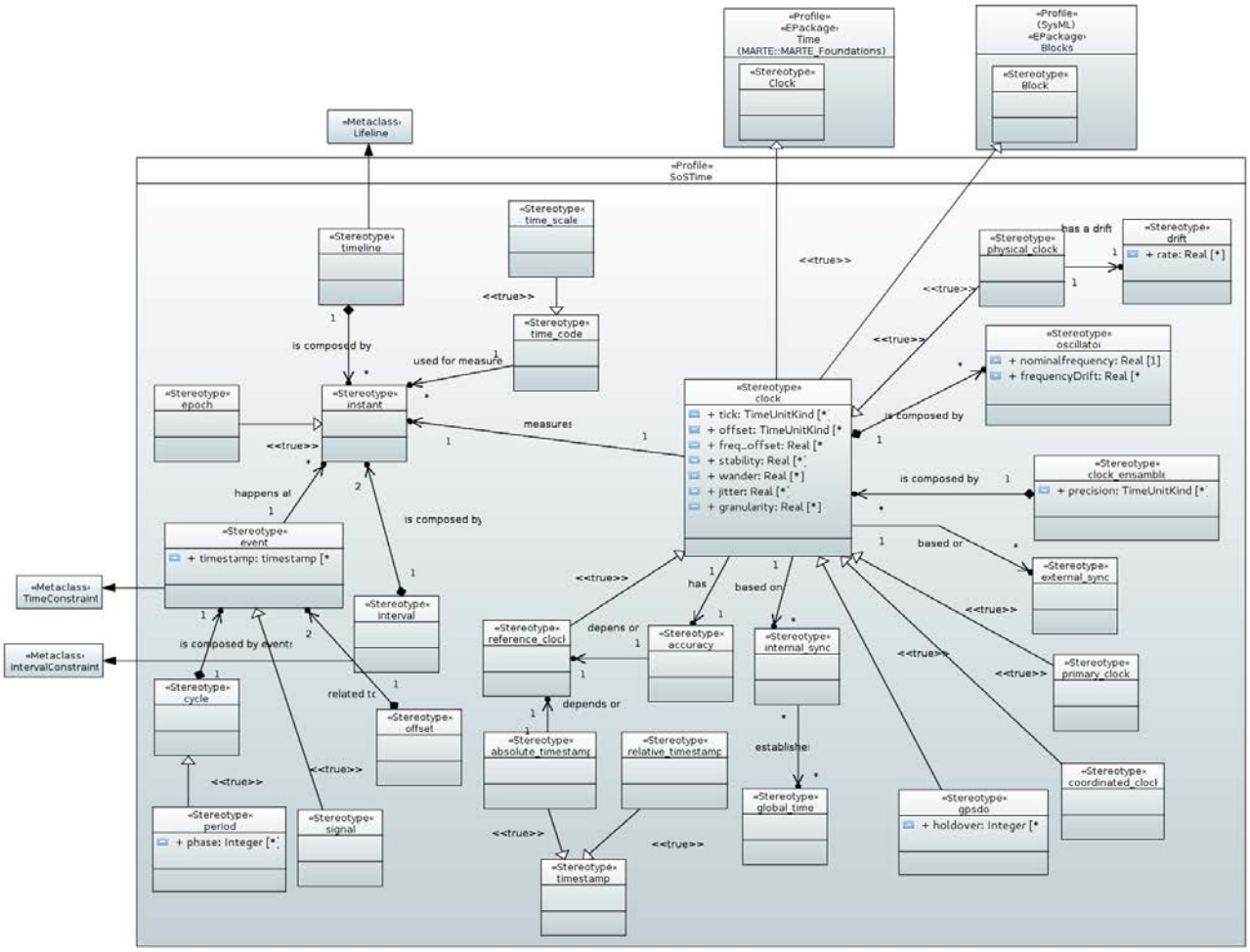

Figure 26: SoS Time package.

An "**event**" can happen at a particular instant and in order to represent this type of information we have used a "TimeContraint" of a Sequence Diagram. A "**signal**" is a particular event used to convey information typically by arrangement between the parties concerned. An "**epoch**" is a particular instant on the timeline chosen as the origin for the time-measurement. A "**cycle**" is a temporal sequence of significant events whereas a "**period**" is a specific type of cycle marked by a constant duration between the related states at the start and the end at the end of the cycle, called "**phase**". The offset of two events denotes the duration between two events and it is represented by the "**offset**" Stereotype.

4.3.5 Multi-criticality component

A multi-critical SoS is a system containing several components that execute applications with different criticality, such as safety-critical and non-safety-critical.

The architecture of safety-critical applications shall be built taking into account that while some part of the system may have strong safety-critical requirements, other parts may be not so critical.

In order to represent this type of architecture, in line with the definition of Section 2.8.4, we introduced the concepts of "**critical_service**" as a particular type of "**service**" having a certain "**critical_level**" (see Figure 27). The latter is associated to "**dependability_guarantee**" and "**security**" to allow the The definition of the stereotype "service" belongs to the SoS Architecture package where it is linked to the CS, i.e., the component, being able to provide the service itself. Thus the concept of "critical_service" is indirectly linked to definition of "SoS" and "CS" by means the definition of "service". The criticality_level is associated to a dependability/security guarantee.

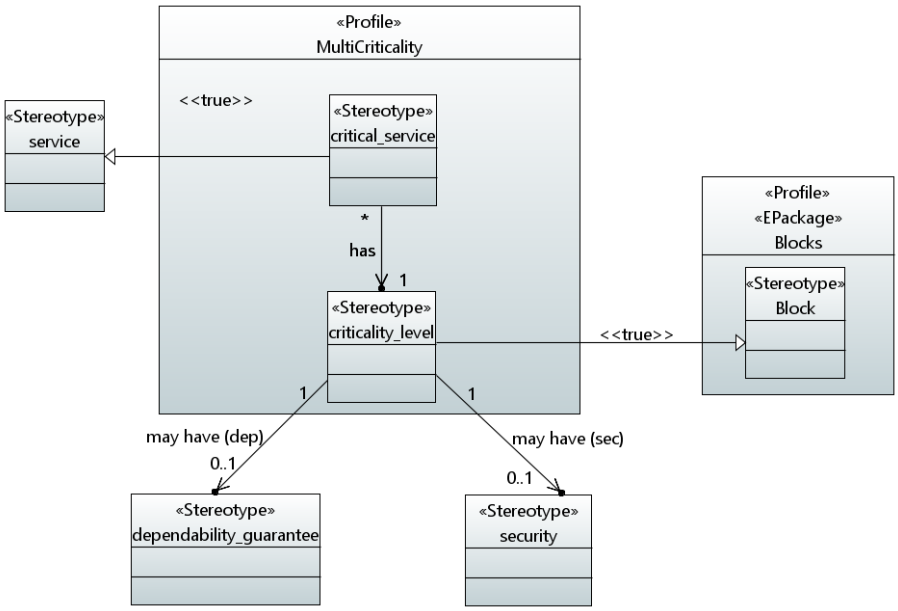

Figure 27: Multi-criticality package.

4.3.6 Emergence component

The concept of Emergence is one of the most important challenges of AMADEOS. As already described in previous sections, SoSs are built to realize new services that CSs separately cannot provide.

In this section we show how to use a semi-formal language in order to represent an emergent behavior of an SoS. Nevertheless, because of the nature of the emergence concept, it is not sufficient defining a semi-formal language thus only eliciting an emergent behavior. Our aim is also capturing operational aspects related to emergence by considering an SoS in action.

For these reasons we propose two different types of representation that a system designer can choose:

- Block Definition Diagram
- Sequence Diagram.

Figure 28 shows the profile package for the emergence behavior as a block definition diagram. This package represents the main concepts of emergence using a Block Definition Diagram. We represent a "**phenomenon**" as a block and we distinguish an "**emergent_phenomenon**" from a "**resultant_phenomenon**". An emergent phenomenon can be explained ("**explained_emerg_phenomenon**") or unexplained ("**unexplained_emerg_phenomenon**") and in the former case there is a trans-ordinary law ("**transOrdinal_law**") that explains the behavior.

An SoS with emergent phenomena has an emergent behavior that could be expected, unexpected, beneficial or detrimental. For this reason we consequently defined the four following blocks:

- "unexpected&detrimental"
- "expected&detrimental"
- "unexpected&beneficial"
- "expected&beneficial"

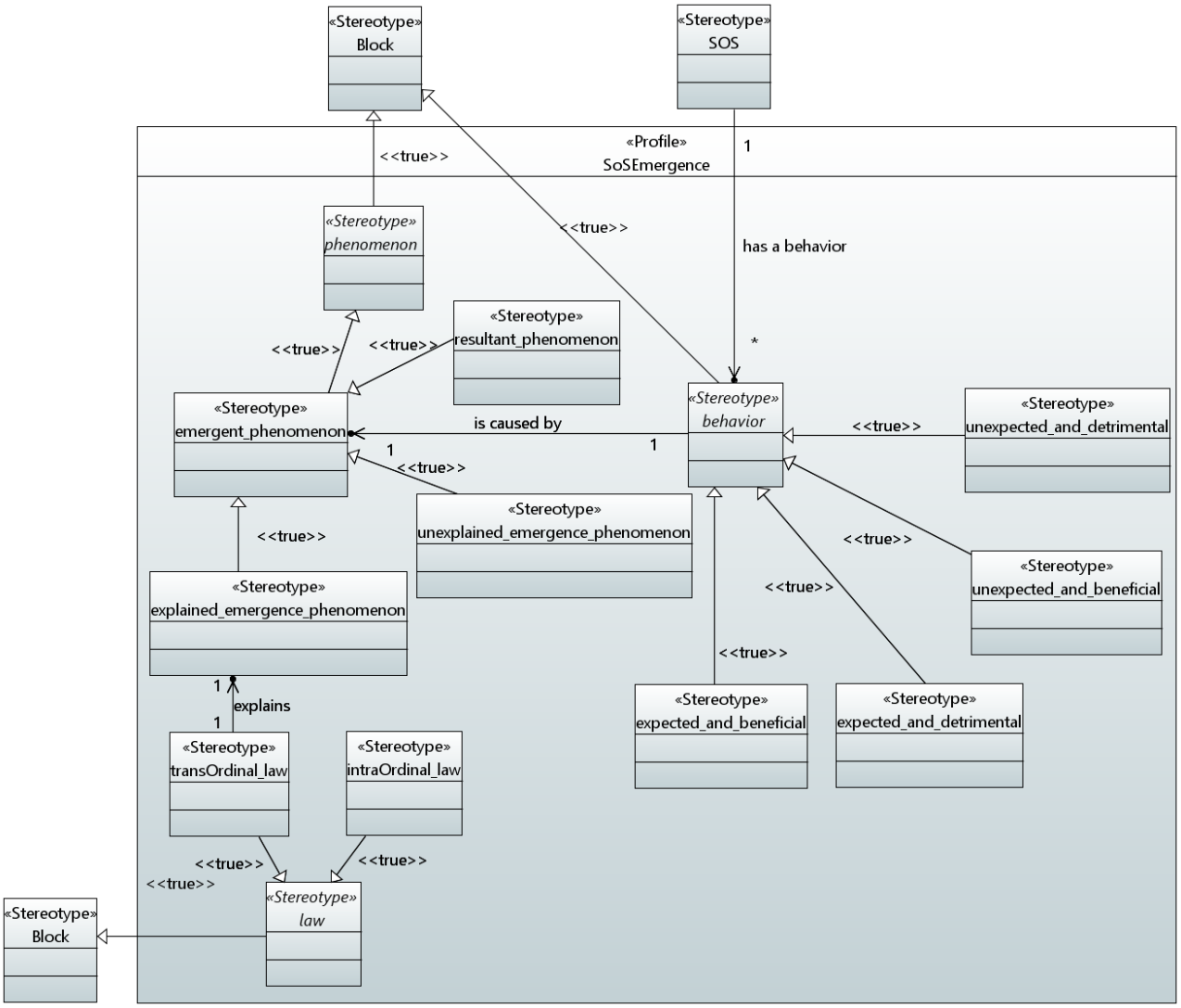

Figure 28: SoS Emergence package.

While a Block definition diagram defines stereotypes and related elements to capture statically the emergence behavior, a sequence diagram is able to define dynamic interactions leading to emergence. As for the dynamicity component, we adopt a sequence diagram where each lifeline represents a constituent system and each message specifies the kind of communication between the lifelines, the sender and the receiver. An SoS is prone to changes; sometimes constituent systems are incremented, modified or removed. To this end, this kind of diagram helps the system designer to easily update and analyze new system behaviors. The diagram not only describes the communication but it also helps to represent the SoS behavior during the progression of time.

To show the difference between a “static” and “dynamic” representation of emergence, we consider a particular scenario of the Smart Grid previously described. The dynamicity of the household electrical appliances which request to be switched on may lead to an emergent behavior of the system in case of a peak of request of energy which comes from the neighbourhood. Let us assume that because of a public event, an exceptional lighting of specific public spaces has to be supported by the Smart Grid. In this case, while in the household it was commonly possible to turn on microwave and washing machine together, we end up in a very limited provision of energy, which is not sufficient for both the electrical appliances. This phenomenon represents an emergent behavior of the Smart Grid since it is not possible to devise it if we only look at the interactions of the internal household CSs without considering the neighbourhood CSs.

Figure 29 shows how, through a BDD, the public event lighting is represented as a explained and detrimental emergent phenomenon explained by the balancing behavior of the Coordinator and causing reduced energy for the electrical appliances. This phenomenon causes an unexpected and detrimental behavior of the SoS, which allows it to only satisfy a subset of energy requests.

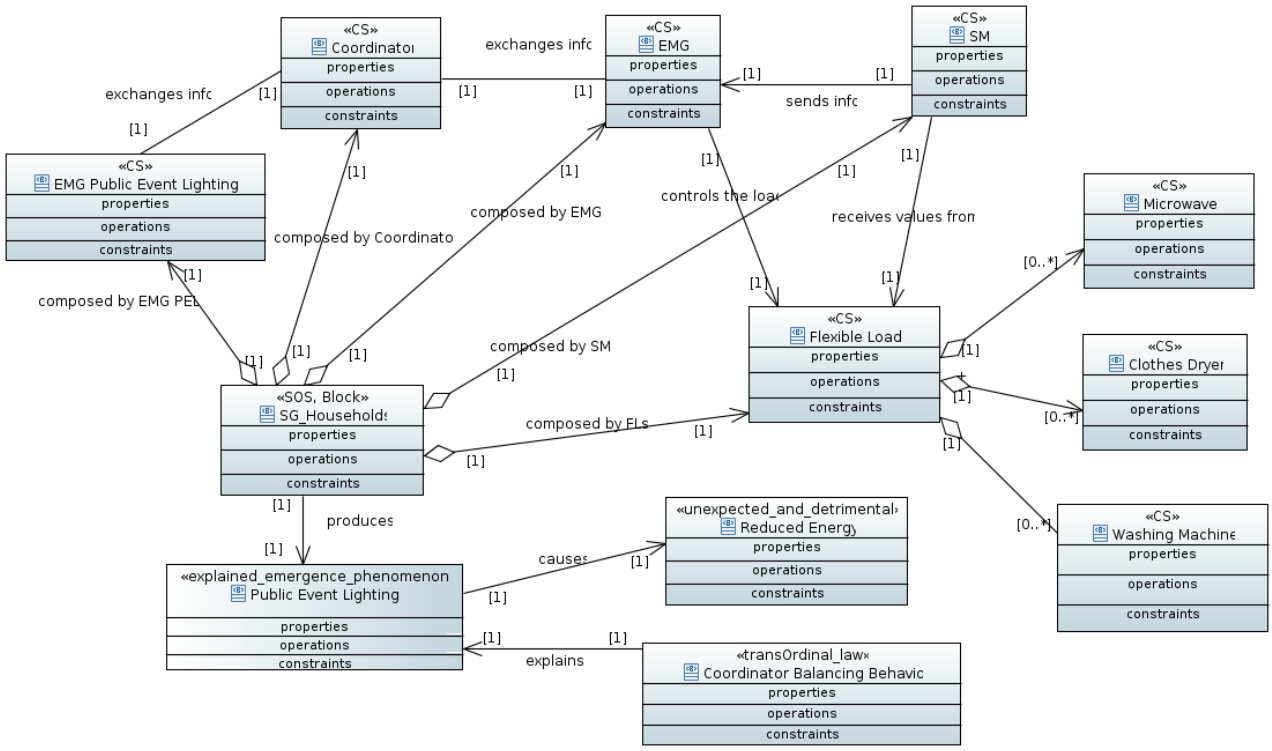

Figure 29: Smart Grid Household – Emergent behavior.

However using this type of diagram we are not able to represent the progression of time and the semantic of message that may contribute to reveal emergence phenomena. Especially, the above representation does not attach greater importance to capture the time aspects of the SoS emergent phenomena.

We now introduce the representation of the exceeding peak energy request by using a sequence diagram. We adopt a sequence diagram to show the emergent behavior of the electrical appliances request by means of the interaction among related CSs of the Smart Grid.

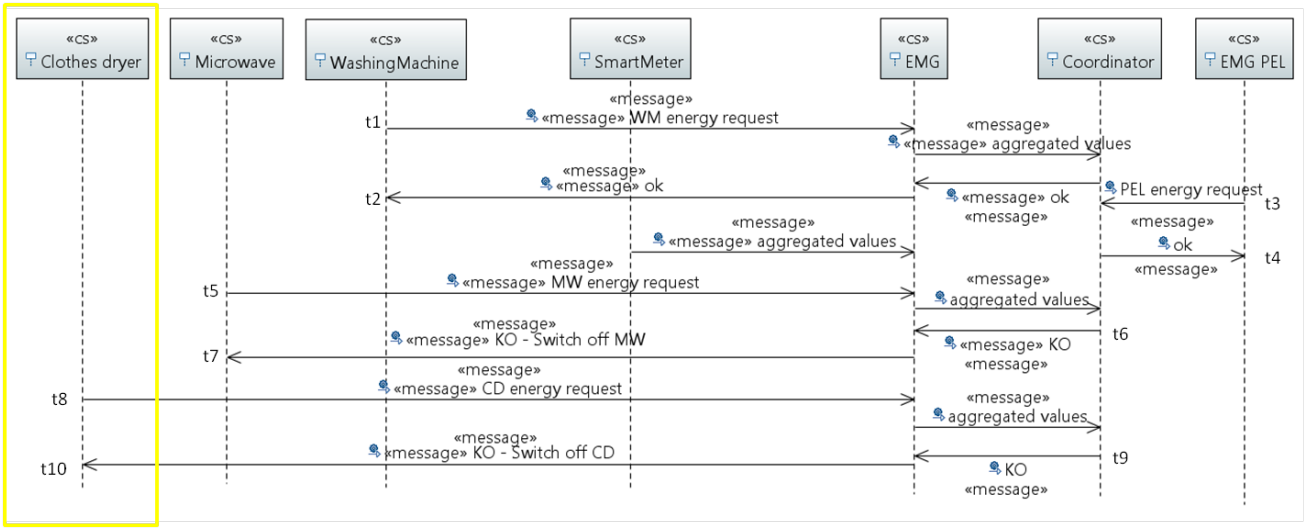

Figure 30: Smart Grid Household SysML Model – Emergent Behavior description.

As shown in Figure 30, “electronic appliances” CSs are represented as “Lifeline” and their interactions are represented through directed labeled arrows. As shown in Figure 30, Washing Machine is switched on at t2 after the agreement allowed from the Coordinator. As next, the Coordinator receives (at time t3) and grants (at time t4) the energy for switching on the public lighting for the exceptional event. This request is forwarded to the Coordinator by the Public Event Lighting (PEL) EMG, which is external to the household. At time t5, microwave issues its request to be connected to the Smart Grid but it receives a negative acknowledgment at time t7. Usually, the household would be able to switch on the washing machine and the microwave at the same time. On the contrary, because of the public event lighting resulting in a peak of energy from the house neighborhood it results that only a reduced amount of energy is available for the electrical appliance (emergent behavior). Indeed, right before the requests issued from the microwave (time t5) and the clothes dryer (time t8), the Coordinator allocates the energy for the public lighting event and consequently no further requests of energy from the house can be granted.

This illustrative example shows that networked individual systems together to realize a higher goal, which none of individual system can achieve in isolation, could lead to an emergent behavior: impossibility of satisfying commonly granted energy requests. The Emergent behavior is shown through the message exchange and it consists of unexpected and detrimental emergent behavior caused by a system dynamicity property.

A further example on emergence is in ANNEX B which refers to a famous case from the Ethernet LAN domain [59].

4.4 EXAMPLE OF PROFILE APPLICATION IN MDE

As we have already described, the semi-formal language we used to describe SoS concepts is SysML [62], a general-purpose visual modeling language for System Engineering applications. Our SoS profile can be adopted along with a Model-Driven Architecture (MDA) approach. MDA is an approach to system development and it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. For a model-driven architecture, our SoS profile is a Platform Independent Model (PIM) or, in other words, a view of the system from the platform independent viewpoint. It provides a set of technical concepts involving SoS architecture and behavior without losing the platform independent characteristics.

This kind of independent architecture makes possible to analyze step by step all the PIM viewpoints and to obtain one or more Platform Specific Models (PSM). A PSM is a view of a system from the platform specific viewpoint. It combines the specifications in the PIM with details that specify how that system uses a particular type of platform.

In other words, our SoS profile represents a Platform Independent Model, which is the first step for a Model-driven methodology in order to realize a Platform Specific Model. Furthermore the SoS PSM can represent the base step for a lot of activities such as the following:

- Source code generation: through an automatic transformation the SoS model can be translate in source code
- System analysis: the SoS model can be the starting point for a lot of system analysis like: hazard analysis (HA), Failure Mode and Effect Analysis (FMEA), Fault Tree Analysis (FTA)
- System testing: the SoS model can be the basic layer to identify test procedures or resolve problems of testing coverage

The SoS profile can be also specialized and improved according to the SoS scope and domain.

4.5 EXAMPLE OF PROFILE APPLICATION IN MULTI-AGENT BASED SOS DESIGN

Multi-agent frameworks are a good match for SoS design and implementation [67][68][69], where the multi-agent framework can function as a common, distributed SoS world-model and toolkit,

where all CSs can regard each other as autonomous entities with private goals, activities, and governance, and various interaction methods between CSs are available. This section discusses how multi-agent frameworks can benefit from applying the concepts created within the AMADEOS project.

Multi-agent frameworks come in very different shapes and sizes [70], ranging from low-level 'middleware-like' frameworks (e.g. to support large numbers of simple agents), to high-level frameworks dictating specific internal agent models. In the domain of multi-agent frameworks, the two main system-modelling concepts are:

- **Agent Models:** Internal models describing how individual agents internalize knowledge and respond to the outside world. These models vary from simple reactive models up to models incorporating human concepts such beliefs, desires, and intentions (BDI).
- **Agent interaction patterns:** Describing families of higher-level communication 'protocols' enabling groups of agents to interact. The most well-known patterns to emerge out of this are negotiation patterns and auctions.

If multi-agent frameworks are to support the AMADEOS SoS concepts, they should provide the necessary components for (i) allowing agents to internally represent and reason about these concepts, and (ii) interaction methods to allow agents to communicate about these concepts.

In this section, two of the AMADEOS SoS viewpoints are selected and transferred to the domain of multi-agent system design, and possible approaches are identified to make this mapping possible.

4.5.1 Viewpoint of Dynamicity in multi-agent based SoS domain

This viewpoint addresses changes in the configuration of an SoS, allowing it to cope with the operational realities such as failing CSs or supporting infrastructure, or other reconfiguration needs.

As described in Section 4.3.6 (check ref), SoS dynamicity can be addressed by modeling the SoS configuration and internal interactions over time (specifically, using Sequence Diagrams).

4.5.1.1 INTERNAL AGENT MODEL:

To enable CSs to detect and subsequently handle irregular events, the agents representing the CSs can be initialized with an internal model based on the semi-formal representation presented in 4.3.6. Using this model, individual CSs can maintain an internalized representation of the state of other CSs, and decide upon actions if problems occur (e.g. stop interactions with the failed CS, reporting the problem via the SoS management infrastructure). Furthermore, more sophisticated agent frameworks can update these representations continuously based on evidence gathered during normal SoS operation (i.e. learning what behavior is considered normal vs. abnormal).

In the figure below, a commonly applied 'situated agent' model is used to show the inclusion of the dynamicity viewpoint models, as well as information on the current state of other CSs (based on their observed communication behavior) within the SoS.

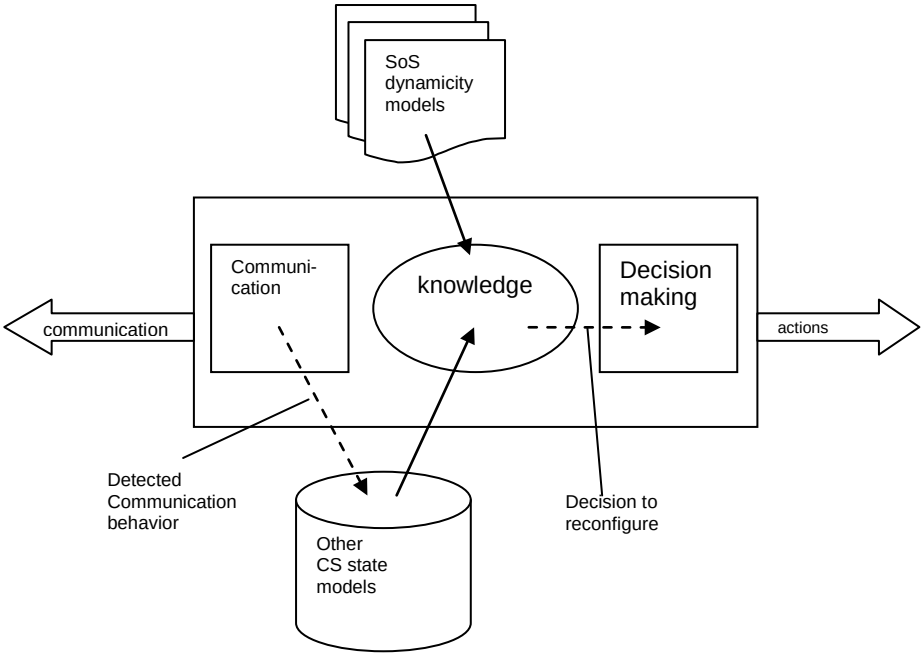

Figure 31: Internal representation of dynamicity viewpoint in CS agent model.

4.5.1.2 INTERACTION MODEL:

In the context of multi-agent based SoSs, the agents representing individual CSs can use meta-level interactions to establish a shared awareness on the current state of other CSs. This can for example be addressed by enabling CS agents to exchange the dynamicity models to increase the combined shared awareness. Similarly, CS agents can share their internal model of the state of other CSs, allowing other CSs to include this information in their reasoning about the current state of other CSs. In the figure below, an example is shown where a CS updates its model of another CS based on a discrepancy between the dynamicity model and the observed interaction patterns, and then shares this information with other CS agents.

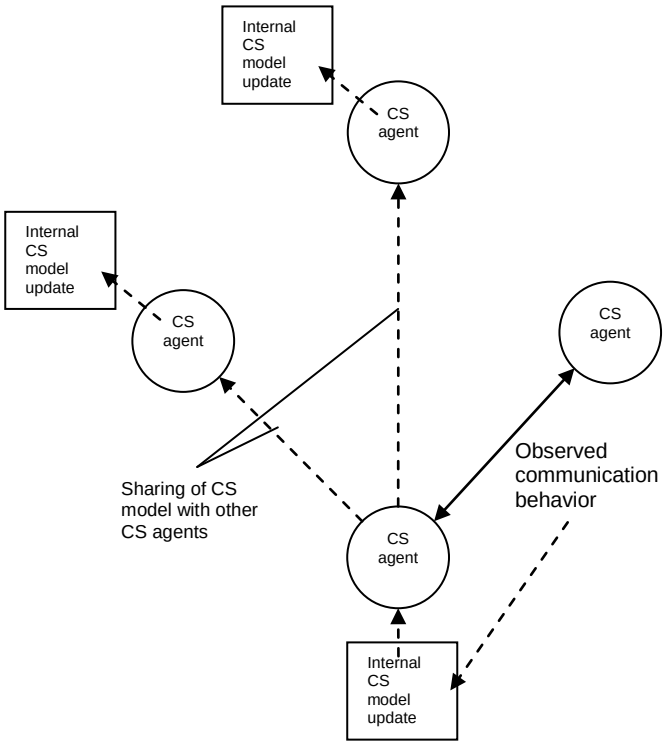

Figure 32: Sharing of Dynamicity viewpoint information between agents representing CSs.

4.5.2 Viewpoint of Emergence in multi-agent based SoS domain

This viewpoint addresses the emergence of global SoS behavior resulting from local CS interactions. Explicitly representing the emergence concept using the model as described in 4.3.6 allows agents representing CSs to observe and reason about the higher-level consequences of the various local interaction methods they apply. Individual CS agents can then modify the configuration of these local interaction methods to influence the global resulting behavior.

As an example, consider the case where CS agents apply some negotiation protocol to establish Quality of Service parameters between a number of CSs. Based on observed emerging results of the performance of the SoS as a whole, individual CS agent may change their negotiation strategies to improve the resulting SoS behavior.

In the figure below, a CS agent observes that in the current operating context, the bandwidth to another CS is insufficient. Based on this observation, the CS agent decides to alter its negotiation strategy to allow for more bandwidth to be allocated to this CS, at the expense of bandwidth to other neighbouring CSs.

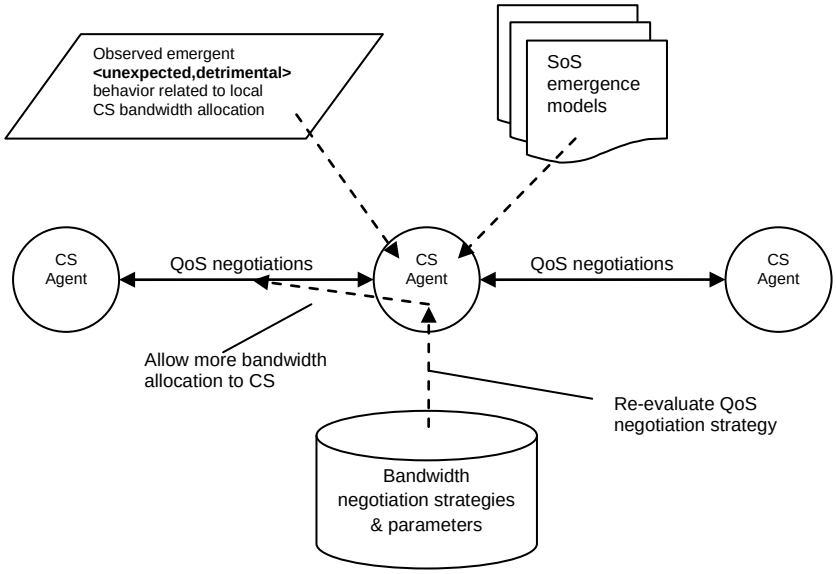

Figure 33: Applying emergence models and observations to adapt local QoS negotiation behavior.

5 CONCLUSION

The document presented the final version of the AMADEOS conceptual model. It discussed the rationale for conceptual modeling of System-of-Systems (SoSs) starting from the achievements of deliverables D2.1 and D2.2. Revisions and extensions to the earlier versions of the conceptual model have been carried out in order to align the conceptual model to the parallel activities in the others WPs, and to include further refined concept definitions and advancements of the conceptual model with respect to interfaces, emergence and evolution. Comments from EU and advisory board reviewers have been considered for these revisions and extensions. The latest achievements in D2.3 are compatible extensions over D2.2, such that a consistent use of the now final AMADEOS conceptual model with the AMADEOS architecture, and the overall evaluation efforts remained possible.

The document presented the SoS basic concepts and relationships graphically according to 7 different views namely *Structure*, *Dynamicity*, *Evolution*, *Dependability and Security*, *Time*, *Emergence* and *Multi-criticality.* Following the graphical representation, the document connects all of the introduced viewpoints with a detailed, conceptual discussion on the *interface* between a Constituent System (CS) and its environment. Further, the document described a SysML profile to represent the conceptual model of SoSs according to the 7 previously identified views.

In a set of annexes – among other topics – a comparison of main AMADEOS concepts with related EU SoS cluster projects and the results of the AMADEOS workshop on emergence in Cyber-Physical Systems-of-Systems (CPSoSs) are presented.

In conclusion, the AMADEOS conceptual model is an explicit attempt to conceptualize the domain of Systems-of-Systems (SoSs). Such a conceptualization provides a language that enables the collaboration among SoS experts to discuss, analyse, design, and evolve SoSs. This deliverable made the AMADEOS conceptual model available for SoS communities to better understand SoSs and further advance the research and exploitation of SoSs.

REFERENCES

[1] *CNSS Instruction No. 4009: National Information Assurance (IA) Glossary*, April 26, 2010. Retrieved from Committee on National Security Systems: http://www.ncix.gov/publications/policy/docs/CNSSI_4009.pdf.

[2] Kopetz, H. "*Real-Time Systems: Design Principles for Distributed Embedded Applications", 2nd ed., Springer 2011.*

[3] Laprie J., LAAS-CNRS, "Resilience for the scalability of dependability", Proc. ISNCA 2005, pp. 5-6.

[4] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Trans. on Dependable and Secure Computing, vol. 1, no. 1, Jan. –Mar. 2004.

[5] BIPM, Joint Committee for Guides in Metrology (JCGM). International Vocabulary of Metrology. Third Edition.

[6] T.B. Quillinan, "Secure Naming for Distributed Computing using the Condensed Graph Model", University of Ireland, 2006.

[7] Security Engineering, Ross Anderson. Second Edition, Wiley. 2008

[8] Schneier, B. "Applied Cryptography". Second edition, Wiley. 1996.

[9] J. Rushby, "The Design and Verification of Secure Systems" In the 8th ACM Symposium on Operating System Principles (SOSP), pp 12—21. Asilomar, CA, December 1981.

[10] The Department of Defence Trusted Computer System Evaluation Criteria (TCSEC) (commonly known as the Orange Book).

[11] "Trust" Oxford English Dictionary, Retreived on 28.04.2014 from: http://www.oxforddictionaries.com/definition/english/trust?q=Trust

[12] Jackson, D et al. *"Software for Dependable Systems: Sufficient Evidence?"*, National Academic Press, Washington, 2007.

[13] Boulding, K. *"The Image: Knowledge in Life and Society",* Univ. of Michigan Press. 2010.

[14] "Dependability" Oxford English Dictionary, Retrieved on 02.05.2014 from: http://www.oxforddictionaries.com/definition/english/dependable

[15] Vigotsky, L.S. "*Thought and Language"*. MIT Press. Boston, 1962.

[16] Hayakawa, S.I. "*Language in Thought and Action"*. Mariner Books, 1991.

[17] Wikipedia, "*Conceptual Model in Computer Science"*. 2014.

[18] Chakravarty, A. "*Scientific Realism".* Stanford Encyclopedia of Philosophy, 2011.

[19] Popper, K. Thr*ee Worlds. "The Tanner Lecture on Human Values".* Univ. of Michigan, 1978.

[20] *"Final Version of the DSOS Conceptual Model*". Technical Report CS-TR-782, University of Newcastle upon Tyne. GB, 2003.

[21] *"D1.1 – SoSs, Commonalities and Requirements".* AMADEOS Project. 2014.

[22] Simon. H. "*The Science of The Artificial".* MIT Press, 1969.

[23] Jamshidi, M. "*Systems of Systems Engineering—Innovations for the 21st century".* Wiley and Sons, 2009.

[24] Dahman, J.S. and Baldwin, K.J. *"Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering*". Proc. of 2nd Annual IEEE Systems Conference. Montreal. IEEE Press. 2008.

[25] Withrow, G.J. "*The Natural Philosophy of Time"*. Oxford Science Publications. 1990.

[26] Winfree, A.T. "*The Geometry of Biological Time"*. Springer Verlag. 2001.

[27] Decotignie, J.D. "*Which Network for which Application?"* in: *The Industrial Communication Technology Handbook.* Ed.: R. Zuwarski, Taylor and Francis, Boca Raton, US, 2005.

[28] Kopetz, H., Ochsenreiter, W., "*Clock Synchronization in Distributed Real-Time Systems".* IEEE Trans. on Computers. Vol 36(8). pp. 933-940. 1987.

[29] Lombardi, M.A. "*The Use of GPS Disciplines Oscillators as Primary Frequency Standards for Calibration and Metrology Laboratories"*. Measure—The Journal of Measurement Science. pp. 56-65. 2008.

[30] US Government Accountability Office: "*GPS Disruptions: Efforts to Assess Risk to Critical Infrastructure and Coordinate Agency Actions Should be Enhanced".* Washington, GAO -14-15. 2013.
[31] Zins, C. "*Conceptual Approaches for Defining Data, Information and Knowledge"*. Journal of the American Society for Information Science and Technology. Vol 58 (4). pp. 479-493. 2007.
[32] Kopetz, H. *A. "Conceptual Model for the Information Transfer in Systems of Systems".* Proc. of ISORC 2014. Reno, Nevada. IEEE Press. 2014.
[33] Floridi, L. "*Is Semantic Information Meaningful Data?"* Philosophy and Phenomenological Research. Vol 60. No. 2.Pp.351-370. 2005.
[34] Glanzberg, M. "*Truth"*. Stanford Encyclopedia on Philosophy. 2013.
[35] World Wide Web Consortium. Extensible Markup Language. URL: http://www.w3.org/XML/ Retrieved on April 18, 2013.
[36] Aviation Safety Network. Accident Description. URL: http://aviationsafety. net/database/record.php?id=19920120-0 retrieved on June 10, 2013.
[37] Siegel, J. "*CORBA 3 Fundamentals and Programming"*. John Wiley, 2000.
[38] Stephan, A., *"Emergence—A Systematic View on its Historical Facets".* In: Beckerman, E at al. Editors. Emergence or Reduction? Walter de Gruyter, Berlin, 1992.
[39] Beckerman, A et al. (ed.) *"Emergence or Reduction—Essays on the Progress of Non-reductive Physicalism".* Walter de Gruyter, Berlin,1992.
[40] Clayton, P., and Davies, P. "*The Reemergence of Emergence".* Oxford University Press, 2006.
[41] Kim, J. *"Emergence: Core Ideas and Issues".* Retrieved from: http://cs.calstatela.edu/~wiki/images/b/b1/Emergence-Core_ideas_and_issues.pdf. Published online on August 9, 2006.
[42] Bedau, M.A. and Humphreys, P., *"Emergence, Contemporary Readings in Philosophy and Science"*. MIT Press, 1968.
[43] Koestler, A. *"The Ghost in the Machine".* Hutchinson. London. 1976.
[44] O'Connor, T. *"Emergent Properties".* Stanford Encyclopedia of Philosophy. 2012.
[45] Davies, C.W. "The Physics of Downward Causation". The Reemergence of Emergence. Ed. By Clayton P and Davies, P. Oxford University Press. 2006.
[46] McLaughlin, B and Bennet, K. "Supervenience". Stanford Encyclopedia of Philosophy. 2011.
[47] Stanislaw H. Zak. "Systems and control". Oxford University Press. 2003.
[48] Mealy, G.H.. Another Look at Data. 1967 Proc. of the Fall Joint Computer Conference.
[49] International Telecommunication Union (ITU). Glossary and Definition of Time and Frequency Terms. Recommendation ITU-R TF686-3. Dec. 2013.
[50] Frei, R., Di Marzo Serugendo,G. Concepts in Complexity Engineering. Int. Journal of Bio-Inspired Computation. Vol.3. No. 2. pp.123-139. 2011.
[51] Kopetz, H., Direct versus Stigmeric Information Flow in Systems-of-Systems. TU Wien, 2014, not yet published. Nov. 2014.
[52] Jiang Y. et al. Stigmergy-Based Collaborative Conceptual Modeling. Pro of ICGSEW 2014. pp. 45-50. IEEE Press. 2014.
[53] Camazine, S., et al. Self-Organization in Biological Systems. Princeton University Press. 2001.
[54] Grasse, P.P. La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La theorie de la stigmergie. Insectes Sociaux Vo. 6., pp. 41-83. 1959.
[55] Khaleghi, Bahador, et al. ''Multisensor data fusion : A review of the state-of-the-art.'' Information Fusion 14.1 (2013). p28-44. 2013.
[56] A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2. OMG Document Number: ptc/2008-06-09.
[57] CHESS: "Composition with guarantees for High-integrity Embedded Software components aSsembly". ARTEMIS-2008-1-100022. url: http://www.chess-project.org/.
[58] Modeling Structure with Blocks-Block Definition Diagrams (Part 1 –SysML Concepts), Joe Wolfrom, The Johns Hopkins University Applied Physics Laboratory LLC.
[59] Mogul, Jeffery C. "Emergent (Mis)behavior vs. Complex Software Systems". In EuroSys, 2006: 293-304.

[60] Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. ACM Trans. On Computer Systems, 15(3):217–252, Aug. 1997.
[61] Redmond, P.J.; Michael, J.B.; Shebalin, P.V., "Interface hazard analysis for system of systems", System of Systems Engineering, 2008. SoSE '08. IEEE International Conference on , vol., no., pp.1,8, 2-4 June 2008.
[62] Model-Based Systems Engineering with the System Modeling Language (SysML) Course Overview, Objective, Approach, and Details. Joe Wolfrom, The Johns Hopkins University Applied Physics Laboratory LLC
[63] Pasquale, T.; Rosaria, E.; Pietro, M.; Antonio, O.; Segnalamento Ferroviario, A., "Hazard analysis of complex distributed railway systems," Proceedings. 22nd International Symposium on Reliable Distributed Systems, pp.283,292, Oct. 2003
[64] Lee, Edward A., and Haiyang Zheng. "Operational semantics of hybrid systems." Hybrid Systems: Computation and Control. Springer Berlin Heidelberg, 2005. 25-53.
[65] Brooks, Christopher, et al. "Ptolemy II-heterogeneous concurrent modeling and design in Java." (2005).
[66] Burns, Alan, and Robert Davis. "Mixed criticality systems-a review." Department of Computer Science, University of York, Tech. Rep (2013).
[67] Alonso, E.; Karcanias, N.; Hessami, A., “Multi-Agent Systems: A new paradigm for Systems of Systems”, Proceedings of The Eighth International Conference on Systems, 8-12, 2013.
[68] Wooldridge, M. (2002). An introduction to multiagent systems. Chichester, UK: Wiley, Feb.
[69] M. Luck, P. McBurney, and O. Shehory and S. Willmott, Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), AgentLink, 2005. ISBN 085432 845 9
[70] Moya, L.; Tolk, A., “Towards a taxonomy of agents and multi-agent systems”, Proceedings of the 2007 Spring Simulation Multiconference, SpringSim 2007, Norfolk, Virginia, USA, March 25-29, 2007, Volume 2
[71] Valckenaers, Paul, Martin Kollingbaum, and Hendrik Van Brussel. "Multi-agent coordination and control using stigmergy." Computers in industry 53.1 (2004): 75-96.
[72] Hoeftberger, O. “*Report on the AMADEOS Workshop on Emergence in Cyber Physical-Systems of Systems."* Vienna University of Technology. May 2016. Available at: http://publik.tuwien.ac.at/showentry.php?ID=249527
[73] Hempel, Carl G., and Paul Oppenheim. *"Studies in the Logic of Explanation."* Philosophy of science 15.2 (1948): 135-175.
[74] Nielsen, Claus Ballegaard, et al. "Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions." ACM Computing Surveys (CSUR) 48.2 (2015): 18.
[75] T. V. Huynh, and J. S. Osmundson, “An Integrated Systems Engineering Methodology for Analyzing Systems of Systems Architectures”, Asia-Pacific Systems Engineering Conference, Singapore, 2007.
[76] J. A. Lane and T. B. Bohn, “Using SysML Modeling To Understand and Evolve Systems of Systems”, System Engineering, vol. 16(1), pp.87-98, 2013.
[77] M. Rao, S. Ramakrishnan and C. Dagli, “Modeling and Simulation of Net Centric System of Systems Using Systems Modeling Language and Colored Petri-nets: A Demonstration Using the Global Earth Observation System of Systems”, Systems Engineering, vol. 11(3), pp. 203-220, 2008.
[78] J. Bryans, J. S. Fitzgerald, R. Payne, and K. Kristensen, “Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML”, INCOSE, 2014.
[79] Ingram, Claire, et al. "Integrating an Upgraded Constituent System in a System of Systems: A SysML Case Study." INCOSE International Symposium. Vol. 25. No. 1. 2015.
[80] COMPASS, “Guidelines for Architectural Modelling of SoS. Technical Note Number: D21.5a Version: 1.0”, September 2014 -http://www.compass-research.eu.
[81] T. Gezgin, C. Etzien, S. Henkler and A. Rettberg, “Towards a Rigorous Modeling Formalism for Systems of Systems”, ISORCW, pp.204-211, IEEE, 2012.
[82] DANSE Consortium, DANSE Methodology V2 - D_4.3 - https://www.danse-ip.eu.

[83] Baldwin, W. Clifton, and Brian Sauser. "Modeling the characteristics of system of systems." System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on. IEEE, 2009.
[84] Franck Petitdemange, Isabelle Borne, and Jeremy Buisson. 2015. Approach based patterns for system-of-systems reconfiguration. In Proceedings of the Third International Workshop on Software Engineering for Systems-of-Systems (SESoS '15). IEEE Press, Piscataway, NJ, USA, 19-22.
[85] MDT/Papyrus. Eclipse Model Development Tools (MDT). http://wiki.eclipse.org/MDT/Papyrus-Proposal.
[86] G. Canfora, and M. Penta, “Service-Oriented Architectures testing: a survey,” In Software Engineering, A. Lucia and F. Ferrucci (Eds.), LNCS 5413, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 78-105.
[87] M. Greiler, H. Gross, and K. A. Nasr, “Runtime integration and testing for highly dynamic service oriented ICT solutions -- An Industry Challenges Report,” In Proceedings of the 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques (TAIC-PART), IEEE Computer Society, Washington, DC, 2009, pp. 51-55.
[88] A. Ceccarelli. "Analysis of Critical Systems Through Rigorous, Reproducible and Comparable Experimental Assessment". Ph.D. thesis. Università degli Studi di Firenze. Dottorato in Ingegneria Informatica e dell'Automazione (XXIV Ciclo). May 9th, 2012.

ANNEX A - Emergence Workshop

Note that this annex is also available as a separate technical report [72].

Editor

Oliver Höftberger, Technical University of Vienna

List of Participants

Andrea Bondavalli	University of Florence
Bernhard Frömel	TU Wien
Erwin Heberle-Bors	University of Vienna
Francesco Brancati	ResilTech
Hermann Kopetz	TU Wien
John Rushby	SRI International
Mariken Everdij	Netherlands Aerospace Centre
Mark Bedau	Reed College
Oliver Höftberger	TU Wien
Radu Grosu	TU Wien
Roozbeh Sangi	RWTH Aachen, L4G Project
Somayeh Malakuti	TU Dresden
Sorin Iacob	Thales NL
Uwe Aßmann	TU Dresden

List of Presentations

Hermann Kopetz	Examples of Emergence in Systems of Systems
Erwin Heberle-Bors	Emergence in Biology
Mark Bedau	A defense of Pluralism about Emergence
Hermann Kopetz	Emergence in Cyber-Physical Systems of Systems
Mariken Everdij	Emergence in Air Transport Operations
Uwe Aßmann	Role-Based Emergence: Modeling Emergence with Roles and Contexts
Mark Bedau	How emergence drives the scientific challenges and opportunities, and the philosophical implications, of CPSoS and BioSoS
Andrea Bondavalli	A view on emergence in SoS and what we may do..... about
John Rushby	On Emergent Misbehavior

Sorin Iacob	Local detection of global effects: Principles and Approaches for Anticipating, Detecting, and Measuring Emergence in CPSoS
Somayeh Malakuti	Component-based Representation of Emergent Behavior in Software
Oliver Höftberger	Detection of Causal Dependencies: Anticipation of possible Emergent Behavior
Roozbeh Sangi	Introduction to Local4Global Project
Hermann Kopetz, All	Conclusions

A.1 Introduction

The objective of the AMADEOS Workshop on Emergence in Cyber-Physical Systems-of-Systems (CPSoS) was to establish an agreed definition of *emergence in CPSoSs*, to clarify the issues around the occurrence of emergent phenomena in CPSoSs and to arrive at design guidelines for the control of emergent phenomena in CPSoSs.

The ultimate purpose of building System-of-Systems (SoS) is to realize novel services and/or functions that go beyond what can be provided by any of the constituent systems (CSs) in isolation. These novel services are emergent services. The concept of emergence is thus at the core of SoS engineering and needs to be fully investigated (understood and deployed).

This report subsumes the presentations and discussions during the workshop. Most of the content in the document is taken from the slides or transcriptions of the talks of the presenters. However, in many cases statements of several participants have been combined. Therefore, knowledge gained from this report is considered to be an intellectual property of all participants listed above. The remaining document starts with different examples of emergence in computer systems and in biology (Chapter A.2). Then a definition of emergence is provided in Chapter A.3 , followed by a discussion of guidelines, modeling and detection techniques in Chapter A.4 . In Chapter A.5 future challenges and questions to be investigated are provided. Finally, a short conclusion is given in Chapter A.6 .

A.2 Examples of Emergence

Aristotle already concluded that the gliwhole is greater than the sum of the parts. If things are put together, then something new appears, where the new cannot be traced down to the properties or behavior of the parts. Emergent phenomena are ontologically novel. It does not mean that they are novel to the knowledge of the user, but conceptually novel to the properties and the behavior of the parts. This is in contrast to the opinion of some philosophers and other people researching in the area of emergence, who say that emergence only exists if it is new to the user. These people claim that if an emergent phenomenon can be explained, it is not emergence anymore. However, this kind of definition is subjective, as for some users the phenomenon is emergent, while for others it is not.

A.2.1 Emergence in Computer Systems

In computer science there are emergent phenomena and effects, which can indeed be explained. In contrast, other emergent phenomena, like the *stock market crash in May 6, 2010*, are still unexplained even after many years of investigation. The example of the stock market shows, that systems are built that are not fully understood, and these systems are even at the center of market economy.

Examples of explained phenomena that have been called emergent in the computing literature are:

- A **deadlock between badly synchronized processes**: This is a simple example of emergence that is fully explained. When this phenomenon (i.e., the computer system stopped) appeared the first time, nobody knew why this was happening. After investigation it was found that two processes are not compatible and block each other. This phenomenon was considered emergent by different scientists. The novel phenomenon is a complete stop of the system that holds forever. The notion of a permanent halt does not exist at the micro-level (i.e., the level of the individual processes). A causal dependency between the totality of the processes that acts on each individual process can be observed. This causality is referred to as *downward causation:* the totality of actions/interactions causes/influences/inflicts a change of behavior of the individual processes. On one side the individual processes are part of the totality of processes, and on the other side the totality of processes constraints/influences the behavior of the individual process itself. This leads to a *causal loop* from the lower level to the level above and from above to below. It manifests in a file-based information transfer from one process to the other process, which goes from the upper level back down to the individual processes.

  A deadlock is not predictable with certainty, but probabilistically it is. In other words, in theory deadlocks are predictable (e.g., by simulations), but in practice it is infeasible. Here predictability means that we are sure whether a deadlock occurs in the next round, or not .In order to predict the deadlock it would be necessary to go to the level of the atoms of the transistors in a computer (considering the temperature, air pressure, etc). One possibility to predict a deadlock is, if a symbolic representation (e.g., an interaction graph) of the system (i.e., the interacting processes) exists, where the feedback loop (e.g., circle in a graph) can be found automatically. However, the model (i.e., the symbolic representation) abstracts from reality and it omits certain relevant conditions. The model can only be used to detect the potential of a deadlock, but not to predict a concrete deadlock. Due to the indeterminism caused by true simultaneity (i.e., at the atom level of the transistors) a deadlock is not predictable in principle, when a truly simultaneous system is assumed, even if everything (i.e., the complete state and behavior) at the macro-level is known. While for good emergence the prediction of potential emergent phenomena is sufficient, for bad emergence this is not the case.

- **Fault tolerant clock (FTC) synchronization**: The new phenomenon is the tolerance of a failure in a clock of the system. This phenomenon is explainable. Downward causation results from the algorithm that is used to build an agreed global notion of time among N clocks. This global notion of time inflicts a change of the state of the local clocks. Each clock is determined by the physics of the oscillator and it influences the global notion of time, which leads to upward causation. The phenomenon is predictable. The FTC is based on the assumption that only one clock is faulty. If this assumption is violated, the emergent property might not exist anymore.

- **Thrashing in alarm monitoring**: An event in a plant (e.g., a broken pipe) causes a stigmergic information flow (i.e., in the physical part of a CPSoS) to several sensors, which leads to correlated messages at those sensors. This incident is followed by an accumulation of messages at a common communication link to the control center, where the sending of messages is retried and even more messages are produced until the real-time properties of the system are violated. The novel phenomenon is the breakdown of communication, which is explainable. The delay caused by the ensemble of concurrent messages in a link of finite capacity causes the real-time communication to break down, which can be interpreted as downward causation. The phenomenon of thrashing is also predictable.

- **Conway's Game of Life (the glider):** The glider in Conway's Game of Life is called emergent by John Holland [Hol00] (and others). Its rules impose a strict cyclic behavior (i.e., evaluation of the next state of each cell). Downward causation appears, as each cycle determines what the next cycle will look like. However, emergence is just a consequence of observation. When observing at a higher level of abstraction something (i.e., a pattern) can

be seen that cannot be observed at the lower level of abstraction. Due to the cyclic evaluation of the cells there is no notion of simultaneity in the Game of Life, and therefore, each step is predictable. If the evaluation of the rules of each cell does not happen in strict cycles, it is not predictable anymore.

- **Stock Market Flash Crash on May 6, 2010:** The complexity of the situation (i.e., huge number of traders, trading algorithms, economic circumstances, news and information exchange, etc.) made it impossible to provide an explanation for this phenomenon, even after years of investigation. As in such complex situations *correlation* is easily confused with *causation*, it might also be impossible to determine the actual root cause of this happening.

  While the concept of causation is highly controversial in modern physics (e.g., quantum mechanics), unidirectional temporal cause-effect relations play a prominent role in our subjective models of the world. Often the word cause is only used for events that are controllable. When looking for a cause, the subsidiary conditions that were necessary are usually neglected.

In almost all examples, which are classified as emergent in the literature, downward causation plays a dominant role.

A.2.2 Emergence in Biological Systems

Emergent phenomena are also observed in biological systems, where the key concepts of emergence are:

- Self-organization, stigmergy
- Feedback
- Interactions, interconnectivity
- Patterns, noise
- Unintended consequences

Four levels of evolution in cosmos are distinguished:

- Physical evolution (emergence of the elements in stars, galactic evolution)
- Chemical evolution (emergence of molecules, 2nd generation star systems)
- Biological evolution (emergence of living organisms)
- Cultural evolution (emergence of cultures and civilizations)

As an example, emergence in chemical evolution is considered. In the world of RNA (Ribonucleic acid), ribozymes are capable of self-replication, mutation and catalysis of other nucleic acids, including DNA and peptides. These emergent capabilities are encoded in the nucleotide sequence. The simplest definition of life requires these three capabilities to exist (self-replication, mutation, catalysis). The RNA world shows properties of life and it existed already before the actual life was formed on our planet. Chemicals form simple molecules that form structures like RNA, which in turn ‘simulate’ processes of life. Finally, the ribozyme surrounds itself with a membrane in order to become a cell.

In a more precise definition of life, a cell is the unit of life with

- metabolism,
- signal recognition and signal transduction,
- inheritance including change (mutations), and it is
- capable of growth, development and self-replication.

Biologists look at the whole organisms, not just the cells, DNA or RNA. In an organism, there are structures with different levels of complexity, which form the hierarchy of life:

- A **tissue** is a group of similar cells.
- An **organ** consists of different tissues.
- An **organ system** consists of different organs.
- Organs and organ systems fulfill specific **functions** in the organism.

- An **individual** is an organism with distinctive properties.
- A **population** is a group of individuals, part of a species, living in the same area and exchanging genes.
- Individuals of populations of different species, cohabiting in a certain area, represent an ecological community, a specific **ecosystem**.
- Ecosystems can be divided, according to their vegetation, into **biomes**.
- All the biomes of the Earth together represent the **biosphere**.

Emergence also appears in the world of genetics. Genes are associated with traits and interact during trait formation in specific ways with other genes and the environment. Due to the influence of the environment, genes are not 'producing' traits. The individual phenotype (sum of the observable characteristics or traits) emerges in the interaction between the individual's genotype (sum of the variations of individual genes) and the environment. Additional random variations in the genes enable the evolution of organisms. Genes only store information, but do not act or produce anything. Proteins are encoded by the genes and are the actors in a cell. They interact with other proteins and with the environment. Signals are sent between cells by, for instance, hormones or neuro transmitters. These signals are often chemicals conveying a specific message, or other factors like humidity, temperature or the amount of light in the environment. The receiver of a signal requires an appropriate receptor, as otherwise signal could not convey its message and it would not be a signal anymore. This is similar to sensors in CPSoSs, which are needed to read information of the physical environment. Furthermore, in biology, signals are unidirectional.

The cooperation of cells in an organism leads to the formation of tissues, organs and organ systems. This requires a multitude of signals to be exchanged. Obviously, the more cells an organism has, the more signals are required for coordination.

There is a deterministic relationship between genes and proteins: DNA provides the code that is transcribed by an enzyme into RNA, which is further transcribed into proteins. The enzymes are like processors reading the code and producing output. Regulatory genes are those genes that are directing processes of higher complexity. The hierarchy within an organism (tissue, organ, organ system) is reflected in a hierarchy of transcription factor genes. Homeotic genes define the form of organs, while cataster genes define the number of organs. The type of each individual cell is given by the concentration of morphogens (i.e., chemical signals). Within an organ a specific pattern is formed by the gradient of these morphogens and thresholds that define where one tissue ends and another one starts (cf. French Flag Model).

The term *anagenesis* in evolution describes the transformation of one species into another, while *cladogenesis* refers to branching speciation. There are two essential views on biological evolution: *phyletic gradualism* – new species gradually emerge – and *punctuated equilibrium* – organisms do not change for a long time, but then they change very rapidly and strongly. Organisms with new body plans might emerge from the accumulation of silent mutations or from macro-mutations that are mutations of homeotic genes. Evolution led to an increasing complexity of body parts, to body homeostasis in order to increase the independence from the environment (e.g., control the body temperature), to incremental shaping of the biosphere by the organisms. Changes on Earth are not arbitrary, but also not circular, nor gradual, but directional, linear, and unpredictably random. A theory exists that evolution has an inherent upwards tendency. Actually, evolution is a staged process where no feedback occurs (i.e., the actions of an organism do not affect the genes of the organism itself, but it might influence the phenotype of its descendants.

A.2.3 Emergence in Air Traffic Management (ATM)

In Air Traffic Management (ATM) different components (agents) are involved: aircrafts, pilots, monitoring via radar, satellites, air traffic controllers on the ground, environment (weather). Interactions among these agents are stochastic (random), discrete (occurring at certain times), continuous interactions or timed interactions.

Example of Emergence in ATM: ATM regulation. Upward causation by the behavior of different participants in the early days of air traffic. The regulation itself creates downward causation on the agents acting in the field.

The term emergence encompasses many different interesting phenomena that are worth studying. A definition of emergence needs to be sufficiently broad in order to capture all different viewpoints. The goal should be:

- Understanding the process of emergence in complex systems: this is necessary to create new forms of complex and robust systems, while being prepared for disturbances
- Understanding the different types of emergence: necessary if we want to understand and master complex systems in science and engineering

A.3 Definition of Emergence

A.3.1 Philosophical View on Emergence

Many competing opinions on the definition of emergence led to the '*emergence wars*', where people try to find and defend their one true view of emergence. Different people disagree what the term emergence means. In order to defend their position they want to show that other points of view are wrong. General to the discussion of emergence is that a whole emerges from its parts. Therefore, the key general idea of emergence is:

- The whole depends on its parts, and
- the whole is autonomous/independent from its parts.

The concepts of dependence and autonomy seem to contradict each other, but if one kind of dependence is used with a different kind of autonomy, then these are not necessarily inconsistent. One can mean many different things by dependence and there can be many different meanings for autonomy.

In order to avoid the emergence wars, a *pragmatic pluralism* should be adopted. Pragmatic pluralism says that depending on the dependence and autonomy selected, different types ('colors') of emergence are meant, which are perfectly consistent. The question for a pluralist is, *whether the dependence and autonomy are coherent* (i.e., logically consistent). As long as it is coherent, it is a valid kind of emergence. But even if a certain kind of emergence is coherent, it might not be useful and no examples exist. Therefore the pragmatist asks *if that kind of emergence is theoretically useful*, if it fits reality and if there are examples.

A *bottom-up whole* means:

- The material of a particular token whole at time t is nothing but the organized combination (i.e., the design) of the materials of its parts at t.
- The state of a whole is nothing but the combination of the states of its parts.
- The cause and effect of the state of a whole is nothing but the combination of the cause and effects of its parts.
- Supervenience is implied by a bottom-up whole.

A *robust pattern* refers to different patterns that are produced by the same rule, where different initial conditions are given. However, all these patterns have the same type, i.e., growth rate, density, maze-pattern, etc. Examples can be found in Conway's Game of Life.

An exemplary distinction of emergence could be:

- **Strong emergence:**
  - Dependence is based on:
    - Matter: the matter of the whole is nothing the matter of the parts.

        - Supervene: different micro states (i.e., states of the parts) might lead to the same macro state (i.e., state of the whole), but different macro states cannot be produced by the same micro state.
    - Autonomy:
        - Undefinable: The properties of the whole cannot be defined in terms of the properties of the parts.
        - Brute (downward) cause: it cannot be explained in principle.
    - Example: conscious mind
    - Because conscious mind might be explained in the future by explaining the interactions of neurons, this concept/model might be incoherent (no brute cause is given, no other valid examples available). There is no evidence that this model fits reality. Therefore, pragmatism would reject this concept (for now).
- **Nominal emergence:**
    - Dependence:
        - Bottom-up whole
    - Autonomy:
        - Inapplicable to parts: there is a property of the whole that is just not defined for the parts.
    - Examples: Glider in Conway's Game of Life (new property: motion, speed, direction), vesicles of lipids (new properties: inside and outside, permeable, self-assemble), traffic jams, cellular automata, fault-tolerant clocks, …
- **Weak emergence:**
    - Dependence:
        - Button-up whole
    - Autonomy:
        - Robust complex cause: the properties of the whole have a complex cause from the properties of the parts.

          A whole's robust patterns of behavior is caused by its parts being in an incompressible causal web (rule b1s1 in Convey's game of life). In order to understand what happens in the future, the causal web has to be crawled ("brute force search"). In contrast, with an arbitrary initial state of a compressible causal web it is possible to predict arbitrarily far in the future what will happen, what pattern will show up (e.g., rule b8s8 in the game of life). The amount of work is always the same. This is like calculating a fixed-point.
    - Examples: fault-tolerant clocks, traffic jams, vesicles, origin of life, cellular automata, …

It is an hypothesis that a causal web is incompressible ('complex') iff it is

- highly parallel (large population),
- highly recurrent (causal loops),
- highly context-dependent (synergy), and
- highly non-linear (summing insufficient).

The epistemic consequences are, that it is impossible to derive the ultimate global state in an incompressible causal web, even if the initial and boundary conditions are completely known, except by brute force search in the causal web. Robust global patterns can be derived from extensive empirical observations. However, in an incompressible causal web some global properties might still be derived without crawling the web.

From the example categories above, nominal emergence is the easiest kind of emergence and applicable to most cases of emergence. An example for nominal emergence, which cannot be categorized as weak emergence, is a circle, where the individual dots are the parts of the circle. The dots do not have a size, but the circle has. The circle also has an inside and an outside.

When looking for a definition of emergence, the most appropriate definition for a specific purpose should be chosen, rather than looking for the one true kind of emergence. For the purpose of CPSoSs, the definition of nominal emergence fits best.

A.3.2 Definition for CPSoSs

An SoS is an integration of a finite number of autonomous constituent systems (CS), e.g., embedded systems, which are independent and operable, and which are networked together for a period of time to achieve a certain SoS goal.

Generally, the SoS goal is not achievable by any of the constituent systems in isolation. However, in some cases there might not be a specific goal of the SoS, but the CSs are put together and interact by chance.

A *CPSoS* consists of CSs that communicate in cyber space via messages on a *cyber channel* (i.e., the *Relied Upon Message Interface* – RUMI) or in the physical space via *stigmergic channels* (i.e., the *Relied Upon Physical Interface* – RUPI). The physical systems interact via actions in the environment. *Information* is a preposition about a state or a process in the world. An information item – referred to as *Itom* – is transferred between the CSs. It consists of the *timed data* (e.g., the bit pattern in a cyber system) and the *explanation* of that data. While the data is carried explicitly in a message, the explanation and the time are often implied by context. In an SoS, the context and the time of the sender can be different from the context and the time of the receiver. If this is the case, then a message that carries data without an explanation can be interpreted differently by the sender and the receiver (e.g., 30°F versus 30°C). A stigmergic information channel is present if one cyber-physical system (CPS) acts on the environment common to many CPSs, changing the state of this physical environment and another CPS observes relevant properties of the changed state at some later point in time with an appropriate sensor. Since stigmergic Itoms are derived from the state of the physical environment (not in cyber space) they are exposed to the full spectrum of *environmental dynamics*.

As an example, ants use pheromones, which they distribute on their routes to coordinate their search for food and nest building activities. The ants react based on the intensity of the pheromones. If a source of food becomes unattractive or because of environmental influences (e.g., rain, wind) the amount of pheromones on a route is decreased and the ants stop to use that route. Stigmergic channels often (unintentionally) close feedback loops, and thus, are the reason for downward causation. For instance, the information flow among drivers on a busy road is mainly of the stigmergic type.

A *multi-level hierarchy* is a modelling structure that limits the overall complexity of a single model and supports the step-wise integration of a multitude of different models. Each level of a hierarchy possesses its unique set of regularities, either natural laws or imposed rules (in the design of artifacts). The phenomenon of emergence is always associated with levels in a hierarchy. However, in some systems the hierarchic levels cannot be clearly distinguished.

The term *holon*, which was introduced by Koestler, refers to the two-faced character of an entity that is considered a whole at the macro level and an ensemble of parts at the micro level. In a holon and in hierarchies of holons – also referred to as *holarchies* – there are two interfaces. One interface where the parts of the holon interact with each other, and the other interface at which the whole interacts with other wholes. This results in a clean multi-level hierarchy. Viewed from the outside (i.e., the macro level) a holon is a stable whole that can be accessed by an interface across its surface. Viewed from below (i.e., the micro-level) a holon is characterized by a set of confined interacting parts. The concept of a holon is almost similar to button-up wholes (see section above), except that buttom-up wholes do not require the interacting parts to be confined. However, if there is no confinement at the lower level, no reasonable abstraction at the higher level can be achieved. The explicit interface at the higher level is required in order to allow the interaction of the lower levels of different holons. For instance, at the cell level the receptors are needed in order to enable molecules at the lower level to interact with the molecules of other cells.

A multi-level hierarchy is a recursive structure where a system – the whole (i.e., the holon) – at the level of interest (i.e., the macro-level), can be taken apart at the level below (i.e., the micro-level), into a set of sub-systems (i.e., the parts) and a design that controls the interactions of the parts. Each one of these sub-systems can be viewed as a system of its own when the focus of observation is shifted from the level above to the level below. This recursive decomposition ends when the internal structure of a sub-system is of no further interest. In the model of a multi-level hierarchy there are two essential relations:

- *Level relations*: How is the upper level related to the lower level? This corresponds to the dependence of the whole from its parts.
- *Interaction relation*: How do the entities at a particular level interact? This corresponds to the autonomy of the whole.

Several different non-exclusive level relations exist. For instance:

- Containment: The whole contains or consists of the parts and the design of the interactions, forming a nested hierarchy. Example: Hierarchy of atoms, molecules, cells, etc.
- Relatedness: The parts of the micro-level are related closer to each other than the wholes at the macro-level. Example: different representations of an Itom and the Itom itself.
- Control: The whole constrains or (partially) controls the behavior of the parts. Example: Blinking of fireflies.
- Description: The parts and the design can be described at different levels of abstraction. Example: Conway's Game of Life.

For SoS design the control relation is the most relevant one. In a control hierarchy, in order to support the simplification at the macro-level and establish an hierarchical control level, a control hierarchy must

- on the one side constrain some degrees of freedom of the behavior of the parts, but
- on the other side it must abstract from, i.e. allow some degrees of freedom of behavior to the parts at the micro-level.

Control originates from two different sources:

- Authority from the outside of the holon: For instance, the authority of a general over the soldiers in a military hierarchy.
- Authority form the inside of the holon: The ensemble of parts at the macro level exercises control over the individual parts at the micro level. This implies that the higher level is equipped with causal powers of its own so that it can inflict effects on the lower level that is causing it.

From the point of view of emergence, authority from the inside is most relevant.

The following two interaction relations are distinguished:

- *Physical Interactions*: come about by force fields, (e.g., electromagnetic or gravitational fields). They are synchronic. Physical structures (e.g., a molecule) are formed by force fields according to physical laws. These interactions are characterized by the distance among the parts, force fields among the parts, relaxation time or frequency of interactions among the parts. When we move up the levels of a material hierarchy the distances increase, the force decreases and the frequency of interactions decreases.
- *Informational Interactions*: come about by the designed exchange of Itoms, either across message channels or stigmergic channels. They are *diachronic*. These interactions happen either direct via state or event messages, or indirect via file-based (in computer systems) or stigmergic (in the physical world) communication.

Emergent behavior in SoSs is caused by informational interactions according to an algorithm. The algorithm that controls the informational interactions is part of the system design. Therefore, the

essence for the occurrence of emergent phenomena at the macro-level lies in the organization of the parts, i.e., in the static or dynamic relation among the parts caused by physical or informational interactions among the parts at the micro-level.

Depending on the type of interactions, the following two kinds of emergence can be distinguished:

- *Synchronic emergence:* These are characterized by static interactions, like in crystals leading to the property of hardness or brilliance of a diamond, caused by the interactions of the atoms.
- *Diachronic* emergence: means, that emergence evolves over time – it does not happen at an instance. For example, a traffic jam is a diachronic emergent phenomenon.

***Definition of CPSoS emergence:* A phenomenon of a whole at the macro-level is emergent if and only if it is of a new kind with respect to the non-relational phenomena of any of its proper parts at the micro-level.**

Conceptual/ontological novelty at the macro-level relative to the world of concepts at the micro-level is the landmark of this definition of emergence. The autonomy in this definition is given by the properties of the whole that the part do not have. The novel phenomena can be structures, behavior or properties. In SoSs we are primarily interested in emergent behavior and emergent properties that are defined based on the behavior of the system (e.g., safety is an invariant of the behavior of the system – i.e., the behavior of the system must be such that nothing bad happens), which are associated predominantly with control hierarchies.

The proper conceptualization of emergent phenomena can lead to an abrupt simplification at the next higher level. Therefore, novel concepts must be formed and new laws may have to be introduced at the macro-level to be able to describe the emerging phenomena at the macro-level appropriately. For instance, the concepts of liquidity or the hydrodynamic laws. Since the concepts at the macro-level are new with respect to existing concepts that describe the properties of the parts, the established laws that determine the behavior of the parts at the micro-level will probably not embrace the new concepts of the macro-level. However, it may be possible to formulate inter-ordinal laws (also called *bridge laws*) to relate the new concepts of the macro-level to the established concepts at the micro-level. In contrast to the view of a number of philosophers, a novel phenomenon at the macro-level is still considered emergent, even if it can be explained by the state of knowledge about the properties and laws that govern the parts at the micro-level. Therefore, the state of knowledge of one person does not impact the classification of a phenomenon as emergent. In a weak emergent system one can even predict what will happen if a simulation exists that is faster than reality.

The Stanford Encyclopaedia on Philosophy states about 'Scientific Reduction': *The term 'reduction' as used in philosophy expresses the idea that if an entity x reduces to an entity y then y is in a sense prior to x, is more basic than x, is such that x fully depends upon it or is constituted by it. Saying that x reduces to y typically implies that x is nothing more than y or nothing over and above y*.

In an artifact, such as an SoS, emergent properties appear at the macro-level if the parts at the micro-level interact according to a design provided by a human designer – this is more than the parts considered in isolation.

Hempel and Oppenheim outlined the following schema for a scientific explanation of a phenomenon: *Given statements of the antecedent conditions and general laws then a logical deduction of the description of the empirical phenomenon to be explained is entailed.* A weaker form of explanation is provided if the general laws in the above schema are replaced by established rules. There are fundamental differences between *general laws* and *established rules*:

- General laws are eternal, inexorable and universally valid, while established rules are context dependent and local.

- Rules about the behavior of things are based on more or less meticulous experimental observations in a limited context.

A special case is the introduction of imposed rules, e.g., the rules of an artificial game, such as chess.

In the field of modern physics, such as quantum mechanics, the meaning of the concept of *causation* is highly controversial. However unidirectional temporal cause-effect relations play a prominent role in our subjective models of the world, and particularly in the design of CPSoSs. *Downward causation* means, that the interaction of the parts at the micro-level cause the whole at the macro-level, while the whole at the macro-level can constrain the behavior of the parts at the micro-level. This results in a *causal loop*.

In a multi-level hierarchy, emergent phenomena often originate from causal loops, which are formed between the micro-level that forms the whole at the macro-level and this whole (i.e., the ensemble of parts) that constrains the behavior of the parts at the micro-level. However, also with linear cause and effect relations (with no feedback), like cascading effects (e.g., nuclear chain reactions, epidemic distribution of viruses), phenomena can be observed that are called emergent.

In a holon there is upward causation by natural laws or from imposed laws and downward causation by the ensemble of parts or from an outside authority (the canon/the algorithm). Within the limits of upward and downward causation, the parts might exhibit free behavior.

Supervenience is a relation between the emergent phenomena of adjacent levels in a hierarchy:

- Sup_1 (Autonomy): A given emerging phenomenon at the macro-level can emerge out of many different arrangements or interactions of the parts at the micro-level.
- Sup_2 (Dependence): A difference in the emerging phenomena at the macro-level requires a difference in the arrangements or the interactions of the parts at the micro-level.

Because of Sup_1 one can abstract from many different arrangements or interactions of the parts at the micro-level that lead to the same emerging phenomena at the macro level. The proper conceptualization of the new phenomena at the macro-level is at the core of the simplifying power of a multi-level hierarchy with emergent phenomena. Whenever the observed emergent behavior at the macro level deviates from the intended behavior, there must be determinants at the micro-level (e.g., the cause of the observed failure – refer to Sup_2), which enables fault diagnosis.

However, even if a difference in the macro-level requires a difference in the micro-level, there need not be a causal relation. For instance, in case of synchronic interactions, where no time is involved, no cause-effect relation exists. As an example, a circle can be considered, where a segment of the circle is removed. Then this circle does not have the emergent properties inside/outside or diameter. The reason why these properties do not emerge at the macro-level is, that a group of dots (a segment of the circle) at the micro-level is missing. However, this incident is not causal. Therefore, supervenience is not always limited to causal contexts. In CPSoSs we want to limit supervenience to causal contexts.

In a CPSoS different types of causal interactions are observed that might lead to emergent behavior:

- *Causal chains*: Finite sequence of dependencies between states of different constituent systems (CS) of different type. The interaction with other CSs does not influence the CS itself.
- *Causal pseudo-loops*: Finite sequence of dependencies between states of different CSs, where the sequence of CS types forms a periodic pattern. The interaction with other CSs does not influence the CS itself, but CSs of the same type. For instance, a car in a traffic jam does not prevent itself from moving, but it prevents other cars to move forward.
- *Causal loops*: Infinite sequence of circular dependencies between states of different CSs. A CS's interaction with other CSs also impacts later inputs for the CS itself.

Balancing between the upward causation and downward causation (control of properties) is called adaptation.

Jochen Fromm introduced in [Fro05] the following taxonomy of emergence based on different types of feedback:

- **Nominal:** Only upward causation; properties at macro-level are different from properties at micro-level.
  - *Simple Intentional*: by design. Example: Function of simple machines (e.g., steam engine, mechanical clocks). Emergence: The function of the whole is not equal to the function of the components.
  - *Simple Unintentional*: just happening. Consist mostly of homogenous (equal) components, like grains or snowflakes. Example: avalanches, pressure of gases.
- **Weak:** simple form of downward causation
  - *Weak stable:* negative feedback (stabilizing feedback). Examples: flocks of birds or fish, termites' nests, Wikipedia.
  - *Weak unstable:* positive feedback (amplifying feedback). Example: people copy the behavior of other people without a common goal. For instance, ghettos are formed if richer people move away from a district and poorer people move in. This causes other rich people to leave that district as well. Also certain forms of economic crashes are caused by this type of emergence.
- **Multiple:** multiple feedback loops, learning, adaptation
  - *Multiple feedback:* combination of short term unstable feedback (mutations) and longer term stable feedback. Example: Patterns in stripes of zebras or tigers, the game of life (different structures move around).
  - *Adaptive multiple:* by sudden changes in existing complex structures certain barriers are overcome that enable a new kind of emergence. Example: the extinction of the dinosaurs enabled other types of life to occur.
- **Strong:** Strong and supervenience, Multi-level emergence with huge amount of variety. Many layers with a combinatorial explosion of states, such that macro properties cannot be predicted or explained from the micro states. (No fundamental limitation in understanding, but too many involved layers in the hierarchy). Example: life, culture.

A possible classification of emergence in CPSoSs is based on explanation. Classification by explanation is relative to the state of knowledge (or the state of ignorance) – it may change as knowledge is increased:

- No explanation as of today – mind over neurons
- Explanation in principle – stock market crash
- Explanation by simulation – air traffic example
- Explanation by logic reasoning – clock synchronization

Another classification of emergence could be by prediction. Prediction is in some sense orthogonal to explanation. Classification by prediction is relative to the state of knowledge (or the state of ignorance) – it may change as knowledge is increased:

- Not predictable in principle
- Stochastic predictability (e.g., by simulation)
- Logic predictability

Therefore, the starting point and goal of prediction must be clearly stated.

The term *misbehavior* originates from a paper of J. Mogul [Mog06]. It does not only refer to failures of complex systems that cannot be predicted from their components, interactions, or design, but also to undesired behavior in general.

In CPSoSs there are three new sources of failure:

- Downward Causation: Suppose we have a function and property that depend on clock monotonicity (the clock always goes forward). When this clock is included in a synchronized system, the clocks can go backwards as well. This might lead to the property failing. Therefore, we must recognize that mechanisms of emergence may not preserve prior properties.
- Stigmergic channels: Components may communicate through unanticipated channels and may then get spontaneous and unexpected emergent behavior (e.g., router synchronization, traffic waves). Partitioning is used to eliminate unintended data channels. But stigmergic channels are more difficult. Conjecture: We may need to consciously design emergence on the stigmergic channels to avoid spontaneous unintended ones.
- Ontological novelty: Missed opportunities (not a failure, but less than optimal behavior): For instance, when a pace maker with 60 beats per minute and a lung machine with 20 beats per minute are connected to a patient. Then the patient establishes a relationship between these systems, leading to possible emergent effects. It is known that recovery is better if both systems are harmonic (i.e., synchronized). The systems should already foresee a possible interaction with other systems (e.g., provide an interface), which requires ontological knowledge about the other systems (e.g., the concept of heart beat).

We need to be able to anticipate, detect, and measure not only emergence, but also global properties in general. There is a distinction between *intrinsic properties* and *extrinsic properties*. Intrinsic properties describe properties of a CS and can reliably be measured locally within a short time interval, without the need for external references. Extrinsic properties, in contrast, describe properties of the SoS. They need an external reference/perspective to be measured. It is conjectured that emergent properties are extrinsic, so no combination of intrinsic quantities can describe an emergent property.
Other global properties are either aggregations or abstractions of intrinsic properties and also important for the proper design and management of SoSs.

Local causes of global effects can be categorized as follows:

- Strongly emergent effects cannot be traced down to specific micro-level causes. Usually this may not happen in CPSoSs.
- Deterministic predictable global effects (e.g., oscillations, linear diffusion) appear in linear non-dissipative systems. They are not really interesting from the perspective of emergence, but useful for SoS management and reaction strategies.
- Deterministic unpredictable global effects (e.g., aperiodic oscillations, bifurcations, phase transitions) are caused by
    - Nonlinear response of CS (far from equilibrium states).
    - Nonlinear composition of CS interactions (e.g. double pendulum, logistic map).
    - Nonlinear or non-isotropic propagation of information.
    - Causal loops (positive feedback), reinforcing causal chains.
    - Dissipative boundary conditions.
- Non-deterministic predictable global effects (e.g., self-organized criticality) are caused by noisy observations, or stochastic state transitions, but they are predictable without exact knowledge of local behavior.
- Non-deterministic unpredictable global effects seem to be close to strong emergence. No examples known.

A.4 Guidelines, Modeling and Detection Techniques

A.4.1 Detection Techniques in Air Traffic Management (ATM)

In ATM situation models are used to simulate interactions between agents. These models are stochastic and involve continuous, discrete and timed processes. The simulations are used to model the properties of the whole system after it is changed.

Monte Carlo simulation (with petri nets) are used to evaluate the safety of active runway crossing operations on an airport. Several millions of simulations are required for this safety evaluation. The simulations are conducted to identify event sequences that are relevant for safety and to omit other ones (in further safety assessments) for cost reasons.

A.4.2 Modeling Emergence with Roles and Contexts

As the limits of a language set the limits of the world (that can be modelled or described), dedicated language concepts are needed to describe emergence. One possibility is to use a role-based modeling language. Thereby, *roles* partition *objects* and their types for separation of concerns. Roles allow to compartmentalize the state of an object. Objects themselves consist of core attributes that do not change (e.g., name of a person), and of attributes that might change over time (e.g., affiliation). In role modelling, one distinguishes between core attributes, which are intrinsic and which the object never loses, and roles, which are attributes that can be lost over the lifetime of the object. Roles belong to a context. They change if context changes. For instance, the role that a person is employed by a certain company is only valid as long as the context of employment (consisting of the person as employee and the company as employer) exists.

Roles and contexts should be first class language concepts. This allows to program with objects, roles and contexts. There is a difference in the memory allocation of these concepts because of the differences in their life-time. Furthermore, roles can improve knowledge about the independence of states.

A context encapsulates a set of roles. Contexts can be composed (parallel, refined, nested) to obtain more complex contexts with more roles. Contexts may also have contracts. A parallel composition means, that an object plays a role in more than one context (e.g., a person can be employee and customer of a certain company at the same time). New context can be formed by composing existing contexts (e.g., a context ‘business’ and a context ‘employment’ can be composed to a new context ‘professional life’ that contains the previous two). Refinement of context means to add more information to the refined context (e.g., the context ‘business’ can be refined by a context ‘history’ that provides information about the business behavior of a customer over time). By nesting contexts, a context is augmented with new roles (e.g., adding supply chain information to the context of a business).When two systems are composed, the context of one system can be superimposed by the context of another system, such that the original context is removed and replaced by the context of the second system. This allows for change which has not been foreseen when the system was designed.

Role- and context-based emergence can be modeled by superimposition, when two systems are put together to obtain an SoS, the context of one system can be superimposed by the context of the other system. The compound of an SoS forms an outer context for all constituents and their inner contexts.
Therefore, an SoS forms a hierarchy of contexts. An SoS may superimpose new contexts to all inner contexts of the constituents, i.e., replace inner contexts of the constituents. A constituent system of an SoS may superimpose new contexts to other constituents of the SoS, i.e., replace inner contexts of the other constituents.

A.4.3 Engineering of Emergent Properties in Synthetic Biology

The goal of synthetic biology is to engineer complex systems with desired weak emergent properties. Actually synthetic biologists learn how to control these properties rather than getting rid

of them. While exact emergent behavior of complex mechanisms is unpredictable, typical emergent behavior can be approximately predicted given enough empirical observation ("experience") – trial and error.

The process of *directed evolution* is an artificial selection in a test tube. Starting with an initial population with random variations (e.g., of genetic molecules), a test is performed that measures the success of each variant. In the next step, those variants which have been selected are varied (i.e., making copies with errors built in). Again the best ones of the new population are selected. This process repeats until the desired emergent properties are obtained. This only works for certain kinds of things that can be expressed with genetic molecules. But this technique does not fit for CPSoSs.

How to engineer complex systems with desired emergent properties or behavior with the predictive design technology (PDT):

1) Sample the experimental space (start randomly)
2) Create a model of the experimental space (e.g., by using methods from statistics)
3) Predict optimal next experiments with which the most information about the experimental space can be obtained.

An example use case for this technique is drug formulation, where the effectiveness of drugs is increased by experiments with different combinations.

A.4.4 Component-based Representation of Emergent Behavior

In some systems it might be important to represent emergent behavior as an entity, because the system has to work with that emergent behavior (e.g., it contains some state information) or perform operations on that. Therefore the question arises, how can emergent behavior (e.g., traffic congestion) be represented in software?

Design and implementation of software imply design and implementation of emergent behavior. Modularization of software behavior implies modularizing simple intentional/nominal emergent behavior.

The following characteristics of emergent behavior have been identified:

- Emergent behavior may have a transient nature: It appears when certain conditions hold, and disappears when the conditions no longer hold.
- Emergent behavior may have a crosscutting nature: It arises from the interactions among multiple entities.
- Emergent behavior may have an elastic nature: The multiplicity of its constituents may change over the time.

Event-based modularization can be employed to model emergent behavior in software by using:

- First-class abstraction: event module / gummy module
- Sub-abstraction: event producer / consumer
- Interaction: Required / published events
- Composition: Inheritance / Aggregation

Event-based algorithms are adopted in the multi-agents community to detect emergent behavior. Emergent behavior is represented as a complex event that may appear as a result of causal relations of simpler events. In this sense an event denotes a change in a state. A state change that leads to an emergent phenomenon is called an emergent event. There is a special kind of module that observes the behavior of the constituents in the system and concludes whether an emergent behavior in the system has appeared or not. In case an emergent behavior appeared, the module

is activated and the system is able to react on the emergent phenomenon, otherwise the module is deactivated.

An existing system can be extended with event-interfaces generating emergent events in order to indicate an onset of emergent phenomena. This is like a wrapper around the existing system with required and provided interfaces. This representation is independent from the actual implementation of the system. Finally, only event-based interfaces are used to communicate between the constituents of the system.

This technique only allows the online detection of anticipated emergent behavior.

A.4.5 Guidelines for CPSoS Design

With proper observation and documentation of interactions among the CSs the occurrence of the emergent phenomena in SoSs can be explained. Still as SoS designers and/or users we would like to gain a complete awareness of emergent phenomena and be able to control (or mitigate the effects of) the detrimental ones.

In SoSs countless causes of unexpected emergence exist, but in many cases unexpected emergence is driven by *ignorance* about:

- The complete set of requirements that need to be addressed in the SoS.
- The complete set of behaviors of each CPS.
- The complete set of interactions among the CPSs.
- The impact of the environment.

System designers/engineers are ignorant about the full set of system behaviors even at the micro-level (CS level). Moreover, things exacerbate when scaling to SoS.

There is also ignorance about the behavior of components:

- Software/hardware systems operate in an unexpected way. For instance, caused by wrong design or implementation or inappropriate use and inclusion in a system (Software wrong reuse).
- Problem of COTS and legacy CPSs used in CPSoS design, as they may come with unknown development faults, they may contain unknown faults (bugs, vulnerabilities, etc.), or their specifications may be incomplete or even incorrect.

In order to contrast detrimental emergence (and boost the good one), we need to reduce the ignorance, eliminate unexpected behavior and interactions. In the past people worked on tracing ignorance as part of requirements engineering by qualitatively quantifying it (e.g., low, medium, high), and tried to understand how this ignorance propagates through simple designs (AND, OR). As a consequence, for the critical parts possibly hiding unexpected emergent behaviors, specific corrective actions could be triggered:

- Replacement of the components identified as potential sources of unwanted emergence.
- Adoption of different architectures, with different correlations between components.
- Cleaning of the environment context, removing possible sources of unknowns.

The ignorance analysis can be profitably used to guide the selection of the system components among several alternatives, e.g., for COTS selection.

When a legacy system or another CS is integrated into an SoS, different assumptions about the behavior of that integrated system are made. A monitor system can be added to the SoS that continuously observes the real system and checks if these assumptions are violated. In case of a violation an alarm is triggered to indicate that an anomaly could occur. Wrapping is another method to constrain some of the behavior of a legacy system and to allow only intended behavior. However, when the system diverges from the expected behavior, a reaction strategy has to exist.

Black and Koopman [Bla(09] observe that safety goals are often emergent to the system components, e.g., the concept (no) ‘collision’ might feature in the top-level safety goal for an autonomous automobile. But ‘collision’ has no meaning for the brake, steering, and acceleration components. They suggest identifying local goals for each component whose conjunction is equivalent to the system safety goal, recognizing that some unknown additional element X may be needed (because of emergence) to complete the equivalence. An objective is then to minimize X.

Reductionist approaches to system design and understanding may no longer be appropriate as systems are built from incompletely understood components or other systems. System goals are far removed from component functions. Widespread emergent misbehavior seems inevitable. In some cases, one can attempt to reduce emergence and restore validity of reductionism. In other cases, one should embrace emergence and aim for adaptation and resilience.

Eliminate unanticipated behaviors and interactions: Behaviors and interactions due to superfluous functionality, e.g., use of a COTS component where only a subset of its capabilities is required, or functions with many options where only some are required. These can be eliminated by wrapping or partial evaluation. Interactions that use unintended pathways (e.g., A writes into B's memory, or interferes with its bus transmissions, or monopolizes the CPU). Strong partitioning of resources can eliminate these hazards. But we remain vulnerable to pathways through the plant.

Control unanticipated behaviors and interactions: Unanticipated behaviors on intended interaction pathways, e.g., unclean failures or local malfunctions. These can be controlled by strong monitoring. For instance, monitor component behavior against system requirements and shutdown on failure, monitor assumptions and treat source component (or self?) as failed when assumptions are violated, use interface automata to monitor interactions, use inline reference monitors (IRMs) to monitor security.

Engineer for Resilience: Diagnosis here is similar to Perrow's Normal Accidents [Per11]. In his terms, we aim to reduce interactive complexity and tight coupling. One way to do both may be to increase the autonomy of components, i.e., they function as goal-directed agents, e.g., substitute runtime synthesis for design-time analysis (both use formal methods, but in different ways). But then it may be more difficult to design the overall system. Furthermore, this opens the door again to emergent misbehavior. Actions of intelligent components frustrate system goals.

Linking intrinsic to extrinsic quantities in CPSoSs can be done:

- Hierarchical
  - Pro: increasing abstraction and semantic levels
  - Con: abstraction removes potentially relevant details, requiring domain models
  - Approaches: Bayesian networks, nonlinear (non-Gaussian) scale-spaces
- Networked
  - Pro: no model required
  - Con: training required (how to cope with unexpected behaviors?)
  - Approaches: unsupervised machine learning

Causal loops/chains are often enabled by stigmergic channels in the environment. The transformation of information in the environment or the mutual interdependency between states in the environment should not be ignored during the development of a CPSoS. Stigmergic channels are not only relevant where CSs comprise dedicated sensors and actuators to interact with the environment, but the environment might also influence CSs via ‘implicit sensors’. E.g., the temperature in the environment might influence the temporal behavior of communication channels in cyber space or it might accelerate/decelerate crystal oscillators.

A network of dependencies can model the interactions between CSs, as well as with the environment at an abstract level. It then enables the automatic detection of causal (pseudo-) loops by performing a graph search. Such loops indicate the possibility of emergent behavior. Then the

system engineers can further investigate those loops in order to decide whether any emergent phenomenon caused by this loop should be prevented.

In order to prevent such unintended emergent effects, system engineers can break up unintended causal loops (e.g., prevent transport of energy or information between CSs), constrain or control the environmental states (e.g., keep the temperature constant), change the design or control algorithm (e.g., add conditions to control algorithm).

A.5 Challenges in the Context of Emergence

Questions about the environment (interactions with environment):

- What is the impact of the environment on CSs?
- Should we have to consider the environment itself as a particular kind of CS interacting with the other CS through stigmergic channels?
- How to define and understand, constrain and predict the environmental behavior?
- Can emergent behavior be triggered by the environment and what we can do about it?

CPSoS behavior cannot be fully predicted, because of

- Unknown scope of the CPSoS
- Incomplete specification of CSs (e.g., the number of specified states is smaller than the actual possible states).
- No top-down design of the SoS. The SoS results from interacting CS.
- In many SoSs there is no global coordination. Only local collaborations and competitions exist. Often there are regional hierarchies and networks.

Investigation of the following questions about local and global system properties might help for developing techniques for detection and prediction of emergent phenomena:

- Is it possible to predict global extrinsic properties from a local perspectives?
- Which global effects can be identified based on locally measured quantities?
- Which local quantities are related to global properties?
- Which local behaviors generate global effects?
- What can be measured? Are there quantifiable relationships between intrinsic and extrinsic quantities?
- How local interactions generate global behaviors? Are there stable relationships between intrinsic and extrinsic patterns?
- Are there fundamental limitations of linking intrinsic properties to extrinsic ones?
- Which intrinsic quantities are useful for anticipating emergent properties? (functional, non-functional)
- Which local system behaviors are likely to lead to unanticipated global effects (including emergence)? (non-linearity, unpredictability, indeterminism)
- What global properties result from ‘local’ (CS-level) cyber-physical interactions?
- How to integrate local measurements? Hierarchical vs. networked
- Is it generally possible to discover stable relationships between intrinsic and extrinsic quantities? Even if unexplained.

A.6 Conclusion

People in diverse fields of science consider the concept of emergence differently, which makes it almost impossible to come up with a single definition of emergence that would fit the requirements and desires of all domains and that is accepted by all parties. Furthermore, there is an analytic viewpoint on emergence from the natural science and the prescriptive viewpoint from system

engineering. As a consequence, it is more constructive to define emergence such that it best fits the domain of interest. However, it is common to all definitions that the whole is more than its parts.

In the field of CPSoSs engineers need to understand emergent behavior to properly design the system and to avoid unintended emergent effects. It is mostly accepted that strong emergence cannot appear in CPSoSs. Nevertheless, many emergent phenomena in those systems still cannot be explained and/or predicted. Sometimes it is possible to explain the behavior, while it is impossible to predict it. In other cases, it is the other way round. The impossibility of explanation and prediction is often based on the ignorance of different aspects during the design of the system. Especially modeling of stigmergic channels and the environment – including environmental dynamics – need much more consideration, as these have been neglected in the past.

A.7 References within Annex A

[Bla09] Jennifer Black and Philip Koopman, *System safety as an emergent property in composite systems*. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems & Networks 2009*, pages 369 – 378, Lisbon, 2009.

[Fro05] Jochen Fromm, *Types and Forms of Emergence*, http://arxiv.org/abs/nlin.AO/0506028, 2005.

[Hol00] John H. Holland, *Emergence: From Chaos to Order*. Oxford University Press, 2000, ISBN: 0-19-286211-1.

[Mog06] Jeffrey C. Mogul, *Emergent (mis)behavior vs. complex software systems*. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006,* ACM, pages 293 – 304, Leuven, Belgium, 2006.

[Per11] Charles Perrow, *Normal Accidents: Living with High Risk Technologies*. Princeton University Press. 2011, ISBN: 0-691-00412-9.

ANNEX B - SYSML PROFILE APPLICATION – ETHERNET CAPTURE EFFECT OF A LAN WITH CSMA-CD CONTENTION PROTOCOL

In this section we show how to use the proposed SoS profile in order to represent an emergent behavior of an SoS consisting in a LAN with CSMA-CD contention protocol.

This is a toy example related to the use of CSMA-CD (Carrier Sense Multiple Access / Collision Detection) Protocol. Our system is the LAN network that uses this kind of protocol and our purpose is showing how to model the emergence case study of the Ethernet Capture Effect [59].

The CSMA-CD Protocol is a type of contention protocol used by LAN networks and it provides a set of rules determining how network devices respond when two devices attempt to use a data channel simultaneously. If no transmission is taking place at the time, one station can transmit, otherwise if two stations attempt to transmit simultaneously, this causes a collision, which is detected by all participating stations. After a random time interval, the stations that collided, attempt to transmit again and if another collision occurs, the time intervals from which the random waiting time is selected, are increased step by step. This is known as exponential backoff phenomenon.

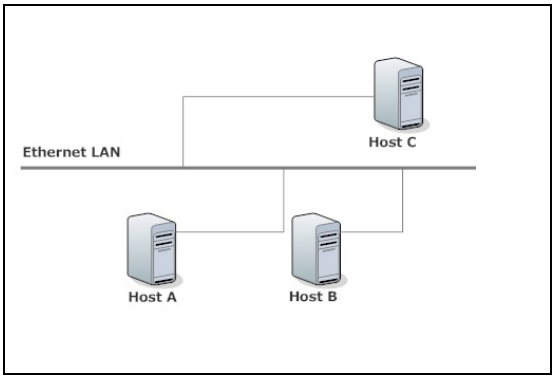

Figure 34: Ethernet capture effect

As shown in Figure 34 the SoS is composed by a LAN with three hosts: A, B and C. A and B have packets to send and simultaneously detect that the channel is free. A and B both will send their own packets thus causing a collision after which A and B calculate a random backoff [59].

Each station has a collision counter, *n*, which is zero at the beginning. Let us suppose that each station picks a backoff time value which is uniformly distributed from 0 to 2^n-1 slots [60]. If station B picks a backoff of 1 (50% probability), A picks a backoff of 0. Since A "won" the first collision, its collision counter is reset and the expected value of B's random backoff is larger than A's. So A will probably win again and B's chances get progressively worse.

This problem was not seen until Ethernet hardware had been in significant commercial use for many years. It only appeared once Ethernet chips were fast enough to fully exploit the timing allowed by the specification. Thus, the capture effect appeared not because of a "problem" with any of the components, but because they were improved (in this local sense) to an optimal point.

The topology of the SoS is showed in Figure 35. We have created a SysML model and we have applied the SoS profile defined in Section 4.3.1. The Stereotypes used are "sos" and "cs". "Ethernet" is the SoS and is composed by "Host_A", "Host_B", "Host_C" and "LAN" which are stereotyped as a CS defined in "*SoSArchitecture*" package.

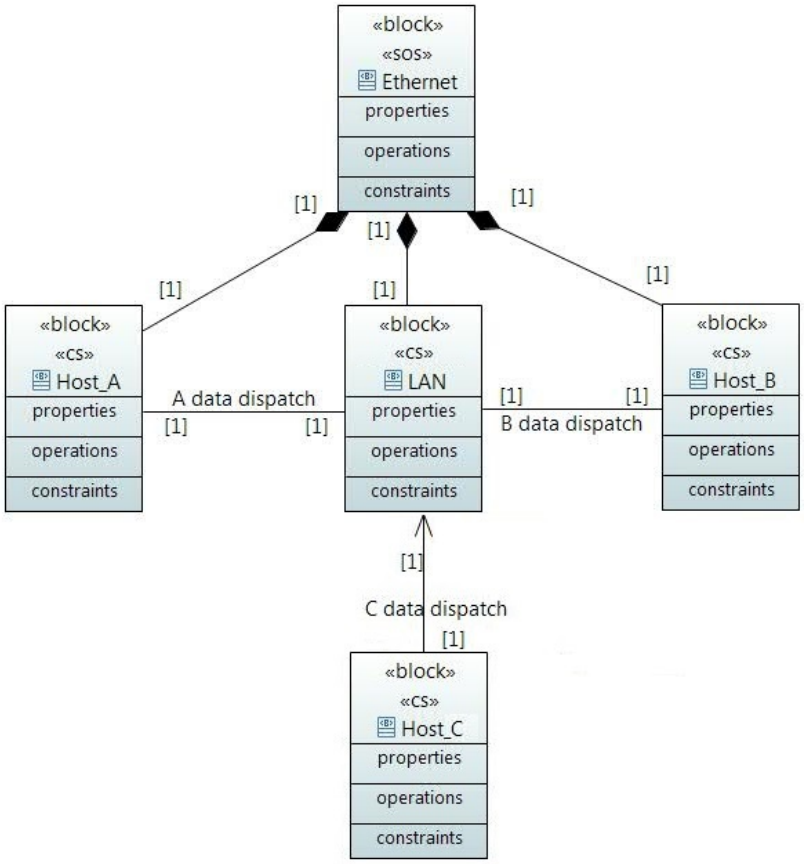

Figure 35: Ethernet Capture Effect SysML Model – Topology description

Starting from the architectural definition of the LAN network, we represent the emergent behavior of trough a Block Definition diagram and a sequence diagram. Figure 36 shows how, through a BDD, the Ethernet Capture Effect is represented as an unexplained emergent phenomenon explained by the random backoff computation. This phenomenon causes an unexpected and detrimental behavior of the SoS.

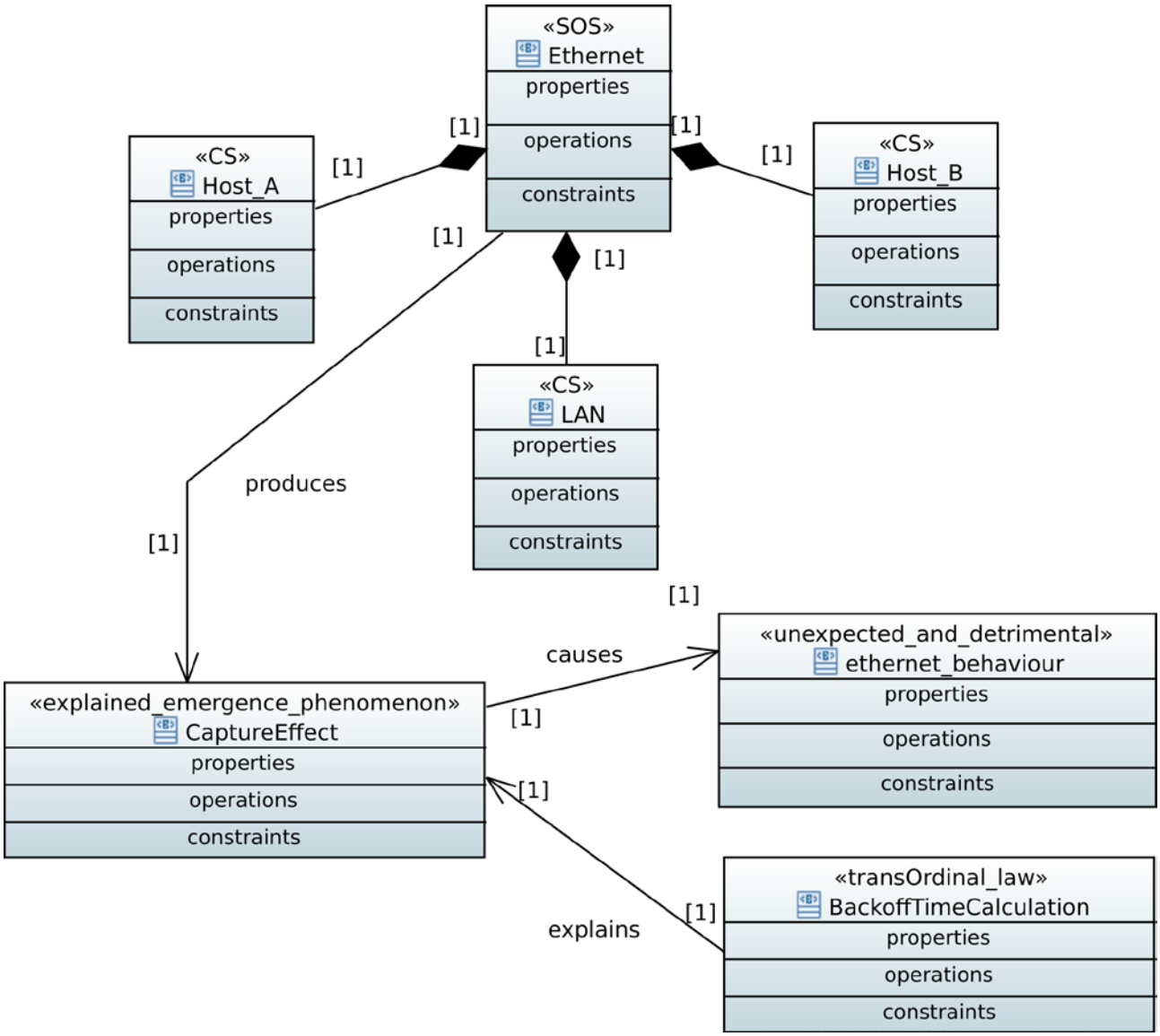

Figure 36: Ethernet Capture Effect – Emergent behavior

We introduce the Sequence diagram to represent the progression of time for the Ethernet Capture Effect leading to an emergent behavior. In this diagram we represent the interaction among the CSs of the CSMA-CD scenario.

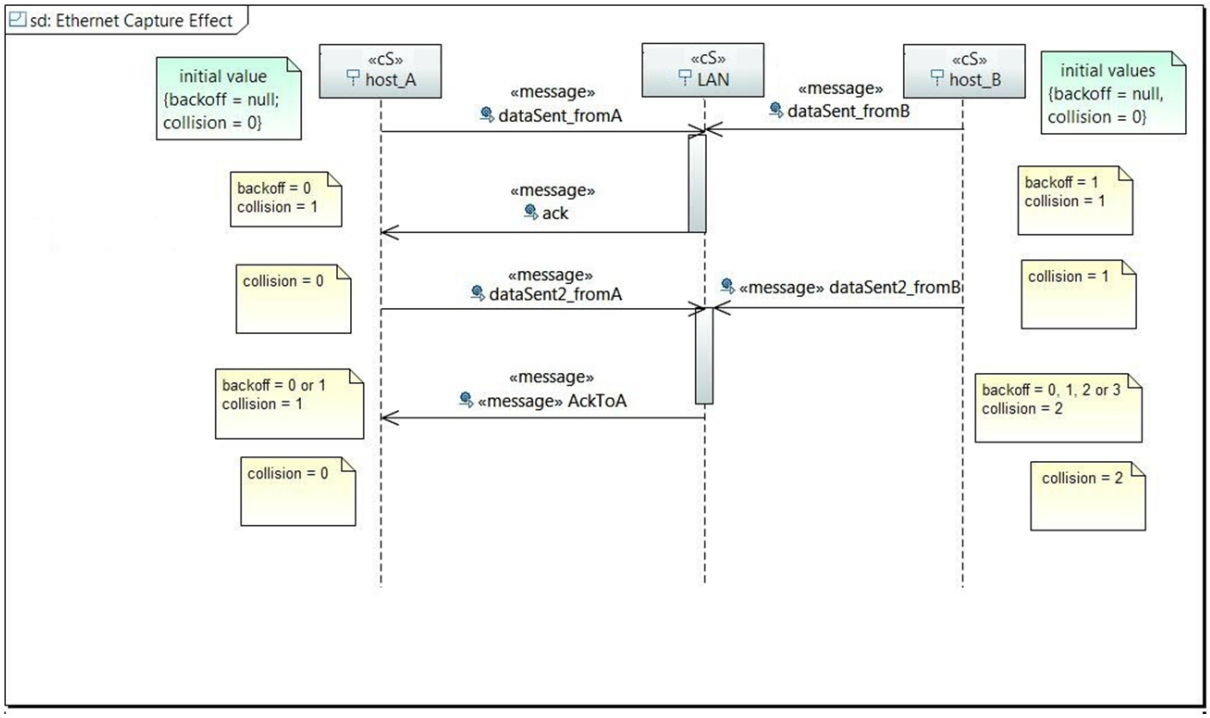

Figure 37: Ethernet Capture Effect SysML Model – Emergent Behavior description

As shown in Figure 37, “Host_A”, “Host_B” and “LAN” are the CSs represented as “Lifeline” and their properties are depicted as constraints or comments. In the figure, the Emergent behavior is shown through the message exchange and the variables updating and it consists in the possible indefinite wait time for Host B of transmitting any message through the LAN. In other words Host B could wait indefinitely without transmit any messages.

ANNEX C - TRACEABILITY MATRIX

Table 5 provides the traceability between the main set of SoS concepts selected from Section 2 and the SoS profile elements described in Section 3.2. Each concept is mapped as an element within SoS profile package or as a methodology step.

Table 5 shows for each SoS profile entity the following information:

- name
- metadata type involved
- diagram type needed in order to use the entity
- any notes

Table 5: Mapping between the main concepts of Section 2 and the SoS profile's elements

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Entity	SoSArchitecture	Stereotype - Block	entity	Block Definition Diagram (SysML)	
Construct	SoSArchitecture		construct	Block Definition Diagram (SysML)	
Thing	SoSArchitecture		thing	Block Definition Diagram (SysML)	
System	SoSArchitecture	Stereotype - Block	system	Block Definition Diagram (SysML)	
Environment of a System	SoSArchitecture	Stereotype - Block	environment	Block Definition Diagram (SysML)	
Autonomous System	SoSArchitecture	Enumeration	sys_type - autonomous	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Architectural Style	SoSArchitecture	Stereotype - Block	architectural style	Block Definition Diagram (SysML)	
Monolithic System	SoSArchitecture	Enumeration	sys_type - monolithic	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Subsystem	SoSArchitecture	Stereotype - Block	subsystem	Block Definition Diagram (SysML)	
Component	SoSArchitecture	Stereotype - Block	subsystem	Block Definition Diagram (SysML)	Component is a subsystem of a system
Legacy System	SoSArchitecture	Enumeration	sys_type - legacy	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Homogenous System	SoSArchitecture	Enumeration	sys_type - homogeneous	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Reducible System	SoSArchitecture	Enumeration	sys_type - reducible	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Prime mover	SoSArchitecture	Stereotype - Block	prime mover	Block Definition Diagram (SysML)	
Role player	SoSArchitecture	Stereotype - Block	role player	Block Definition Diagram (SysML)	
Interface	SoSArchitecture	Stereotype - Block	interface	Block Definition Diagram (SysML)	
Cyber-Physical System (CPS)	SoSArchitecture	Stereotype - Block	cps, physical_oblect, cyber_system	Block Definition Diagram (SysML)	
Closed System	SoSArchitecture	Enumeration	sys_type - closed	Block Definition Diagram (SysML)	Enumeration literal of sys_type
Open System	SoSArchitecture	Enumeration	sys_type- open	Block Definition Diagram	Enumeration literal of

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
				(SysML)	sys_type
Evolutionary System	SoSArchitecture	Enumeration	sys_type - evolutionary	Block Definition Diagram (SysML)	Enumeration literal of sys_type
System-of-Systems (SoS)	SoSArchitecture	Stereotype - Block	SoS	Block Definition Diagram (SysML)	
Human-Machine Interface	SoSArchitecture	Stereotype - Block	HMI	Block Definition Diagram (SysML)	
Constituent System (CS)	SoSArchitecture	Stereotype - Block	subsystem (o CS)	Block Definition Diagram (SysML)	
Directed SoS	SoSArchitecture	Enumeration	sos_type	Block Definition Diagram (SysML)	Enumeration literal of sos_type
Acknowledged SoS	SoSArchitecture	Enumeration	sos_type	Block Definition Diagram (SysML)	Enumeration literal of sos_type
Collaborative SoS	SoSArchitecture	Enumeration	sos_type	Block Definition Diagram (SysML)	Enumeration literal of sos_type
Virtual SoS	SoSArchitecture	Enumeration	sos_type	Block Definition Diagram (SysML)	Enumeration literal of sos_type
Relied upon Message Interface (RUI)	SoSCommunication	Stereotype - Block	rui	Sequence Diagram (UML2)	
Relied upon Message Interface (RUMI)	SoSCommunication	Stereotype - Block	rumi	Sequence Diagram (UML2)	
Relied upon Message Interface (RUPI)	SoSCommunication	Stereotype - Lifeline	rupi	Sequence Diagram (UML2)	
Diagnostic Interface (D-Interface)	SoSCommunication	Stereotype - Block	D-Interface)	Sequence Diagram (UML2)	
Monitoring CS	SoSCommunication	Stereotype - Block	Monitoring_CS	Sequence Diagram (UML2)	
Configuration Interface (C-Interface)	SoSCommunication	Stereotype - Block	C-Interface)	Sequence Diagram (UML2)	
Timeline	SoSTime	Stereotype - Lifeline	timeline	Sequence Diagram (UML2)	
Instant	SoSTime	Stereotype - Lifeline	instant	Sequence Diagram (UML2)	
Event	SoSTime	Stereotype - TimeConstraint	event	Sequence Diagram (UML2)	
Signal	SoSTime	Stereotype - Lifeline	signal	Sequence Diagram (UML2)	
Time code	SoSTime	Data Type	time_code	Sequence Diagram (UML2)	
Temporal oder	SoSTime	Stereotype - TimeConstraint	N.A.	N.A.	Internal property of a lifelne
Time scale	SoSTime	Data Type	time_scale	Sequence Diagram (UML2)	
Interval	SoSTime	Stereotype - IntervalConstraint	interval	Sequence Diagram (UML2)	
Offset of events	SoSTime	Stereotype - TimeConstraint	offset	Sequence Diagram (UML2)	
Epoch	SoSTime	Stereotype - Lifeline	epoch	Sequence Diagram (UML2)	
Cycle	SoSTime	Stereotype - Lifeline	cycle	Sequence Diagram (UML2)	
Period	SoSTime	Stereotype - TimeConstraint	periodic	Sequence Diagram (UML2)	
Periodic System	SoSArchitecture	Enumeration	sys_type - periodc	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Phase	SoSTime	Property	cycle - phase	Sequence Diagram (UML2)	
Offset of events	SoSTime	Clock attribute - MARTE	offset	Block Definition Diagram (SysML)	
Clock	SoSTime	Stereotype - Clock MARTE	clock	Block Definition Diagram (SysML)	
Nominal Frequency	SoSTime	Property	oscillator - nominalfrequency	Block Definition Diagram (SysML)	
Frequency drift	SoSTime	Property	oscillator - frequencyDrift	Block Definition Diagram (SysML)	
Frequency offset	SoSTime	Property	clock - frequency	Block Definition Diagram (SysML)	
Stability	SoSTime	Property	clock - stability	Block Definition Diagram (SysML)	
Wander	SoSTime	Property	clock - wander	Block Definition Diagram (SysML)	
Jitter	SoSTime	Property	clock - jitter	Block Definition Diagram (SysML)	
Tick	SoSTime	Clock attribute - MARTE	tick	Block Definition Diagram (SysML)	
Granularity/Granule of a clock	SoSTime	Property	clock - granularity	Block Definition Diagram (SysML)	
Reference clock	SoSTime	Stereotype - Block	reference_clock	Block Definition Diagram (SysML)	
Coordinated clock	SoSTime	Stereotype - Block	coordinated_clock	Block Definition Diagram (SysML)	
Drift	SoSTime	Stereotype - Block	drift	Block Definition Diagram (SysML)	
Drift rate	SoSTime	Property	drift - rate	Block Definition Diagram (SysML)	
Timestamp (of an event)	SoSTime	Stereotype	timestamp	Block Definition Diagram (SysML)	
Absolute Timestamp	SoSTime	Property	absolute_timestamp	Block Definition Diagram (SysML)	
Internal Clock Synchronization	SoSTime	Stereotype - Block	internal_sync	Block Definition Diagram (SysML)	
External Clock Synchronization	SoSTime	Stereotype - Block	external_sync	Block Definition Diagram (SysML)	
Primary Clock	SoSTime	Stereotype - Block	primary_clock	Block Definition Diagram (SysML)	
Accuracy	SoSTime	Property	accuracy	Block Definition Diagram (SysML)	
Clock Ensamble	SoSTime	Stereotype - Block	clock_ensamble	Block Definition Diagram (SysML)	
Precision	SoSTime	Property	clock - precision	Block Definition Diagram (SysML)	
GPSDO	SoSTime	Stereotype	gpsdo	Block Definition Diagram (SysML)	
Holdover	SoSTime	Property	gpsdo - holdover	Block Definition Diagram (SysML)	
Message	SoSCommunication	Stereotype	sos message	Sequence Diagram (UML2)	
Send Instant	SoSCommunication	Property	send_instant	Sequence Diagram (UML2)	Property of trailer
Arrival Instant	SoSCommunication	Property	arrival_instant	Sequence Diagram (UML2)	Property of trailer
Receive Instant	SoSCommunication	Property	receive_instant	Sequence Diagram (UML2)	Property of trailer

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Payload of a Message	SoSCommunication	Property	data_field	Sequence Diagram (UML2)	Property of message
Valid	SoSCommunication	Property	valid	Sequence Diagram (UML2)	Property of message_classification
Checked	SoSCommunication	Property	checked	Sequence Diagram (UML2)	Property of message_classification
Permitted	SoSCommunication	Property	permitted	Sequence Diagram (UML2)	Property of message_classification
Timely	SoSCommunication	Property	timely	Sequence Diagram (UML2)	Property of message_classification
Value correct	SoSCommunication	Property	correctness	Sequence Diagram (UML2)	Property of message_classification
Correct	SoSCommunication	Property	correctness	Sequence Diagram (UML2)	Property of message_classification
Insidious	SoSCommunication	Property	insidious	Sequence Diagram (UML2)	Property of message_classification
Datagram	SoSCommunication	Enumeration	datagram	Sequence Diagram (UML2)	Enumeration literal of transportation_service
PAR-Message	SoSCommunication	Enumeration	PAR-message	Sequence Diagram (UML2)	Enumeration literal of transportation_service
TT-Message	SoSCommunication	Enumeration	TT-message	Sequence Diagram (UML2)	Enumeration literal of transportation_service
Stigmergic	SoSCommunication	Stereotype - Block	Evironment, rui and state_space	Sequence Diagram (UML2)	
Emergence	SoSEmergence	Stereotype - Block	emergent_phenomenon	Block Definition Diagram (SysML)	Emergence phenomenon is also described and analysed using SoS Architecture package and a Sequence Diagram
Resultant phenomenon	SoSEmergence	Stereotype - Block	resultant_phenomenon	Block Definition Diagram (SysML)	
Trans-ordinal Law	SoSEmergence	Stereotype - Block	transOrdinal_law	Block Definition Diagram (SysML)	
Intra-ordinal Law	SoSEmergence	Stereotype - Block	intraOrdinal_law	Block Definition Diagram (SysML)	
Unexpected and Beneficial emergent behavior	SoSEmergence	Stereotype - Block	unexpected&beneficial	Block Definition Diagram (SysML)	
Expected and Beneficial emergent behavior	SoSEmergence	Stereotype - Block	expected&beneficial	Block Definition Diagram (SysML)	
Unexpected and Detrimental emergent behavior	SoSEmergence	Stereotype - Block	unexpected&detrimental	Block Definition Diagram (SysML)	
Expected and Detrimental emergent behavior	SoSEmergence	Stereotype - Block	expected&detrimental	Block Definition Diagram (SysML)	
Explained emergence	SoSEmergence	Stereotype - Block	explained_emerg_phenomenon	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Unexplained Emergence	SoSEmergence	Stereotype - Block	unexplained_emerg_phenomenon	Block Definition Diagram (SysML)	
Failure	SoSDependability Profile	Stereotype	failure	Block Definition Diagram (SysML)	
Failure modes	SoSDependability Profile	Stereotype	failureType, failureClassification	Block Definition Diagram (SysML)	
System outage	SoSDependability Profile	Stereotype	outageState	State Machine Diagram (SysML)	
System restoration	SoSDependability Profile	Stereotype	restorationState	State Machine Diagram (SysML)	
Error	SoSDependability Profile	Stereotype	errorModel, errorState	State Machine Diagram (SysML)	
Fault	SoSDependability Profile	Stereotype	fault	State Machine Diagram (SysML)	
Availability	SoSDependability Profile	Stereotype - Block	availability	State Machine Diagram (SysML)	
Reliability	SoSDependability Profile	Stereotype - Block	reliability	State Machine Diagram (SysML)	
Maintainability	SoSDependability Profile	Stereotype - Block	maintainability	State Machine Diagram (SysML)	
Safety	SoSDependability Profile	Stereotype - Block	safety	State Machine Diagram (SysML)	
Integrity	SoSDependability Profile	Stereotype - Block	integrity	State Machine Diagram (SysML)	
Robustness	SoSDependability Profile	Stereotype - Block	robustness	State Machine Diagram (SysML)	
Fault prevention	SoSDependability Profile	Stereotype - Block	faultPrevention	State Machine Diagram (SysML)	
Fault tolerance	SoSDependability Profile	Stereotype - Block	faultTolerance	State Machine Diagram (SysML)	
Fault removal	SoSDependability Profile	Stereotype - Block	faultRemoval	State Machine Diagram (SysML)	
Fault forecasting	SoSDependability Profile	Stereotype - Block	faultForecasting	State Machine Diagram (SysML)	
Security	SoSSecurity	Stereotype - Block	security	Block Definition Diagram (SysML)	
Encryption	SoSSecurity	Stereotype - Block	encryption	Block Definition Diagram (SysML)	
Cryptography	SoSSecurity	Stereotype - Block	cryptography	Block Definition Diagram (SysML)	
Plaintext	SoSSecurity	Stereotype - Block	plaintext	Block Definition Diagram (SysML)	
Ciphertext	SoSSecurity	Stereotype - Block	cipertext	Block Definition Diagram (SysML)	
Decryption	SoSSecurity	Stereotype - Block	decription	Block Definition Diagram (SysML)	
Key	SoSSecurity	Stereotype - Block	key	Block Definition Diagram (SysML)	
Symmetric Cryptography	SoSSecurity	Stereotype - Block	symmetric_cryptography	Block Definition Diagram (SysML)	
Public key Cryptography	SoSSecurity	Stereotype - Block	public_key_cryptography	Block Definition Diagram (SysML)	
Symmetric key	SoSSecurity	Stereotype - Block	symmetric_key	Block Definition Diagram (SysML)	
Private key	SoSSecurity	Stereotype - Block	private_key	Block Definition Diagram (SysML)	
Public key	SoSSecurity	Stereotype - Block	public_key	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Subject	SoSSecurity	Stereotype - Block	subject	Block Definition Diagram (SysML)	
Object	SoSSecurity	Stereotype - Block	object	Block Definition Diagram (SysML)	
Authentication	SoSSecurity	Stereotype - Block	authentication	Block Definition Diagram (SysML)	
Authorization	SoSSecurity	Stereotype - Block	authorization	Block Definition Diagram (SysML)	
Access control	SoSSecurity	Stereotype - Block	accessControl	Block Definition Diagram (SysML)	
Access control model	SoSSecurity	Stereotype - Block	accessControlModel	Block Definition Diagram (SysML)	
Permission	SoSSecurity	Stereotype - Block	permission	Block Definition Diagram (SysML)	
Security policy	SoSSecurity	Stereotype - Block	securityPolicy	Block Definition Diagram (SysML)	
Reference monitor	SoSSecurity	Stereotype - Block	referenceMonitor	Block Definition Diagram (SysML)	
Evolution	SoSEvolution	Stereotype - Block	evolution	Block Definition Diagram (SysML)	
Managed evolution	SoSEvolution	Stereotype - Block	managed_evolution	Block Definition Diagram (SysML)	
Managed SoS evolution	SoSEvolution	Stereotype - Block	managed_evolution	Block Definition Diagram (SysML)	
Unmanaged SoS evolution	SoSEvolution	Stereotype - Block	unmanaged_evolution	Block Definition Diagram (SysML)	
Business value	SoSEvolution	Stereotype - Block	businnes_value	Block Definition Diagram (SysML)	
System resources	SoSEvolution	Stereotype - Block	system_resource	Block Definition Diagram (SysML)	
Dynamicity	SoSDynamicity	Stereotype - Block	dynamicity	Block Definition Diagram (SysML)	
Dynamicity	SoSDynamicity	Stereotype - Block	dynamic_service	Block Definition Diagram (SysML)	
Dynamicity	SoSDynamicity	Stereotype - Block	reconfigurability	Block Definition Diagram (SysML)	
Dynamicity	SoSDynamicity	Stereotype - Block	dynamic_interaction	Block Definition Diagram (SysML)	
Scenario-based reasoning	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	
Scenario	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	
Scenario	SoSSBR	Stereotype - Block	event	Block Definition Diagram (SysML)	
Domain models	SoSSBR	Stereotype - Block	domain_model	Block Definition Diagram (SysML)	
Domain models	SoSSBR	Stereotype - Block	variables	Block Definition Diagram (SysML)	
Causal model	SoSSBR	Stereotype - Block	causal_model	Block Definition Diagram (SysML)	
Causal graphs	SoSSBR	Stereotype - Block	causal_graph	Block Definition Diagram (SysML)	
SoS inference processes	SoSSBR	Stereotype - Block	inference_process	Block Definition Diagram (SysML)	
SoS inference processes	SoSSBR	Stereotype - Block	scenario_state	Block Definition Diagram (SysML)	
Scenario pruning	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Scenario updating	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	
Situation assessment	SoSSBR	Stereotype - Block	situation_assessment	Block Definition Diagram (SysML)	
Situation awareness	SoSSBR	Stereotype - Block	situation_assessment	Block Definition Diagram (SysML)	
Decision making	SoSSBR	Stereotype - Block	decision_making	Block Definition Diagram (SysML)	
Decision alternative	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	
Decision alternative	SoSSBR	Stereotype - Block	decision_making	Block Definition Diagram (SysML)	
Multi-Criteria Decision Analysis	SOSSBR	Stereotype - Block	mcda	Block Definition Diagram (SysML)	
Multi-Criteria Decision Analysis	SOSSBR	Stereotype - Block	decision_making	Block Definition Diagram (SysML)	
Criticality level	SoSMultiCriticality	Stereotype - Block	Critical_level	Block Definition Diagram (SysML)	
Criticality	SoSMultiCriticality	Stereotype - Block	Critical_level	Block Definition Diagram (SysML)	
Critical service	SoSMultiCriticality	Stereotype -Block	Critical_service	Block Definition Diagram (SysML)	
Interaction	SoSInterface	Stereotype - Block	interaction	Block Definition Diagram (SysML)	
Channel	SoSInterface	Stereotype - Block	channel	Block Definition Diagram (SysML)	
Channel model	SoSInterface	Stereotype - Block	channel_model	Block Definition Diagram (SysML)	
Transferred Information	SoSInterface	Stereotype - Block	interaction - transferred_info	Block Definition Diagram (SysML)	
Temporal Properties	SoSInterface	Stereotype - Block	interaction - temporal_properties	Block Definition Diagram (SysML)	
Dependability Requirements	SoSInterface	Stereotype - Block	interaction - Dependability_req	Block Definition Diagram (SysML)	
Interface properties	SoSInterface	Stereotype - Block	property	Block Definition Diagram (SysML)	
Interface Specification	SoSInterface	Stereotype - Block	interface_specification	Block Definition Diagram (SysML)	
Interface Cyber-Physical Specification (CP-Spec)	SoSInterface	Stereotype - Block	cp-spec	Block Definition Diagram (SysML)	
Interface Itom Specification (I-Spec)	SoSInterface	Stereotype - Block	i-spec	Block Definition Diagram (SysML)	
Interface Service Specification (S-Spec)	SoSInterface	Stereotype - Block	s-spec	Block Definition Diagram (SysML)	
Interface Message Specification (M-Spec)	SoSInterface	Stereotype - Block	m-spec	Block Definition Diagram (SysML)	
Interface Physical Specification (P-Spec)	SoSInterface	Stereotype - Block	p-spec	Block Definition Diagram (SysML)	
Transport Specification	SoSInterface	Stereotype - Block	Transport_specification	Block Definition Diagram (SysML)	
Message-based Interface Port	SoSInterface	Stereotype - Block	interface_port	Block Definition Diagram (SysML)	
Direction	SoSInterface	Stereotype - Block	FlowDirectionKind	Block Definition Diagram (SysML)	
Size	SoSInterface	property	property - size	Block Definition Diagram (SysML)	
Type	SoSInterface	property	property - type	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Temporal Properties	SoSInterface	property	property – temporal_prop	Block Definition Diagram (SysML)	
Dependability Properties	SoSInterface	Stereotype - Block	property – dependable_prop	Block Definition Diagram (SysML)	
Message Variable	SoSInterface	Stereotype - Block	message_variable	Block Definition Diagram (SysML)	
Interface Model	SoSInterface	Stereotype - Block	interface_model	Block Definition Diagram (SysML)	
Service Level Agreement (SLA)	SoSInterface	Stereotype - Block	SLA	Block Definition Diagram (SysML)	
Service Level Objective (SLO)	SoSInterface	Stereotype - Block	SLO	Block Definition Diagram (SysML)	
Reservation	SoSInterface	Stereotype - Block	reservation	Block Definition Diagram (SysML)	
Reservation Request Instant	SoSInterface	property	reservation – request_instant	Block Definition Diagram (SysML)	
Reservation Allocation Instant	SoSInterface	property	reservation – allocation_instant	Block Definition Diagram (SysML)	
Reservation End Instant	SoSInterface	property	reservation – end_instant	Block Definition Diagram (SysML)	
Internal Interface	SoSInterface	Stereotype - Block	internal_interf	Block Definition Diagram (SysML)	
External Interface	SoSInterface	Stereotype - Block	external_interf	Block Definition Diagram (SysML)	
Utility Interface	SoSInterface	Stereotype - Block	utility_interf	Block Definition Diagram (SysML)	
Configuration and update Interface (C-Interface)	SoSInterface	Stereotype - Block	c-interface	Block Definition Diagram (SysML)	
Diagnosis Interface (D-Interface)	SoSInterface	Stereotype - Block	d-interface	Block Definition Diagram (SysML)	
Local I/O Interface (L-Interface)	SoSInterface	Stereotype - Block	l-interface	Block Definition Diagram (SysML)	
Sensor	SoSInterface	Stereotype - Block	sensor	Block Definition Diagram (SysML)	
Actuator	SoSInterface	Stereotype - Block	actuator	Block Definition Diagram (SysML)	
Transducer	SoSInterface	Stereotype - Block	transducer	Block Definition Diagram (SysML)	

ANNEX D - Comparison of conceptual model with related EU SoS Cluster Projects

Key terms of AMADEOS conceptual model, based on the viewpoints, have been selected to compare AMADEOS with other projects. In order to compare these terms, we checked the general description in the web pages of Local4Global, Dymasos and CPSOS and then we looked into the deliverables which contained the most of their conceptual definitions.

Of course, differently from AMADEOS, not all the other projects contain a set of definitions as we provided. So, information has been extrapolated from the deliverables as an attempt in understanding to what extent each AMADEOS viewpoint term has been considered.

In the following we report the results of this analysis. When we found a definition of the term we put it into the table, otherwise we put the use of the term from our understanding of the analyzed deliverables (see references). Within the table, definitions which are reported from deliverables are double-quoted while our understanding on the use of terms is not double-quoted.

AMADEOS Terminology and definition	**Local4Global**	**Dymasos**	**CPSOS**
Multi-criticality *Multi-criticality system: A multi-criticality system has at least two components that have a different criticality.*	Checking safety-critical threshold but not different level of criticality	Not addressed	No addressed
Dependability The ability to avoid failures that are more frequent and more severe than is acceptable.	Not addresses. Evaluating the efficiency of control problems solutions. "Development of control methodologies by using self-learning and situation awareness mechanisms extension of standard control-theoretical approaches"	Exploiting design / optimization and validation models to assess the suitability of control mechanisms for certain classes of systems of systems. There is no specific reference to dependability properties.	Dependability properties are guaranteed with the support of devised modeling and simulation tools. Nevertheless there is no a clear list of attributes (availability, etc...) which are supported but it is only mentioned about the assessment of functional and non-functional (performance) properties.
Security The composition of confidentiality, integrity, and availability; security requires in effect the concurrent existence of availability for authorized actions only, confidentiality, and	Not found	Not found. "SoS engineering tools have to manage the ownership of information, responsibility for correctness and access rights. This is not an important issue for security, but also to trace conclusions drawn from an SoS engineering tool back to the original real systems."	"Cyber-security is a very important element in cyber-physical systems of systems. A specific CPSoS challenge is the recognition of obstructive injections of signals or takeovers of components in order to cause malfunctions, suboptimal performance, shutdowns or accidents, e.g. power outages. The detection of

integrity (with "improper" meaning "unauthorized").			such attacks requires taking into account both the behavior of the physical elements and the computerized monitoring, control and management systems."
Evolution Process of gradual and progressive change or development, resulting from changes in its environment (primary) or in itself (secondary).	Not found.	Physical process evolution	"Cyber-physical systems of systems are systems that evolve continuously over long periods of time both in terms of the purpose they serve as the means they have for achieving those. As a consequence the engineering of such a system has to be performed at run-time. The waterfall paradigm "Requirements – modelling – model-based design – verification – commissioning – operation - dismantling" is not applicable to systems of systems where the requirements change during operation." Focus specific to the runtime dimension
Dynamicity The property of an entity that is constantly changing in terms of offered services, built-in structure and interactions with other entities.	"Dynamics of interaction among CSs but also changing structure, topology and hierarchy"	"Dynamic interactions of the locally managed components." "Dynamic lower modeling and simulation technology." "Dynamic (i.e. time thermodynamics"	"Dynamic reconfiguration refers to the addition or removal of components on different time scales, depending on the nature of the system and the reasons for the changes of the structure and changes of the way the system is operated."
Time A continuous measureable physical quantity in which events occur in a sequence proceeding from the past to the present to the future.	"Real-time decision making methods for generating decisions in a worst- case or robust manner."	All modeled information is bound by a time interval that determines when the information is valid. "Time Discretization / Sampling Time". No clear if a global time is provided	No mentioning of a global time base. Just handling large amounts of data in real time to monitor the system performance and to detect faults and degradation
Emergence	"Emergent behavior:	Not addressed	"The point of view adopted

A phenomenon of a whole at the macro-level is emergent if and only if it is of a new kind with respect to the non-relational phenomena of any of its proper parts at the micro level.	Through a SoS operation, new properties may appear, which cannot be anticipated beforehand and cannot be deduced from understanding the constituent systems and their properties."		in the CPSoS project is that the emerging behavior is to be restricted to the occurrence of patterns, oscillations or instabilities on a systems level and the formation of structures of interaction in a way that had not been anticipated in the construction of the subsystems and in the design of their interactions." It seems that they do not consider positive expected emergent phenomena.
Architecture System Architecture: The blueprint of a design that establishes the overall structure, the major building blocks and the interfaces among these major building blocks and the environment.	"Developing, evolving and maintaining an architecture for the SoS"	High (Software)(HLA) / Run (RTI). The latter represent the simulation environment	It is mentioned an "Hierarchical Management and Control" for the management and control of SoS. It seems comparable to our management infrastructure, though phases like monitoring, planning, analysis and execution are not explicitly differentiated.

D.1 References within Annex D

- CPSOS
  - Project web page - http://www.cpsos.eu/project/what-are-cyber-physical-systems-of-systems/
  - Deliverable D2.4 - http://www.cpsos.eu/wp-content/uploads/2015/02/D2-4-State-of-the-art-and-future-challenges-in-cyber-physical-systems-of-.pdf
- DYMASOS
  - Project web page - http://www.dymasos.eu/
  - Deliverable D4.1 - http://www.dymasos.eu/wp-content/uploads/2013/12/D4_1-DYMASOS-Engineering-Concept-Specification.pdf
- Local4Global
  - Project web page - http://local4global-fp7.eu/

ANNEX E - DEFINITION DIFFERENCES WRT D2.2

This annex lists the definitions or sections that have been modified or added compared to deliverable D2.2.

Location	Concepts	Rationale/Changes/Comments
2.1.2	Constituent System (CS)	Clarified that the environment of a CS can be modelled as another CS.
2.1.3	Relied upon Service (RUS)	Introduced new, previously implicit, concept.
2.2.3	Time-aware SoS	Introduced new, previously implicit, concept.
2.6		Revision of subsection addressing 2nd review recommendations and reflecting results of the AMADEOS workshop about emergence (e.g., renamed weak emergence to explained emergence, renamed strong emergence to unexplained emergence; Better alignment with actual semantics, Added concept of cascading effects).
2.9.1	Local Evolution, Global Evolution, Evolutionary Performance, Evolutionary Step, Minor Evolutionary Step, Major Evolutionary Step, Agility (of a system), Dynamicity (of a system), Reconfigurability	Introduced new concepts and refined a few during the work on D3.2, Section 4 (SoS Evolution).
2.10.1	Designer	Refined to also include behavioral properties.
2.11	(Collaborative) SoS Authority, Incentive	Introduced new concepts developed during the work on D3.2, Section 4 (SoS Evolution).
2.13	Cyber Space, Environmental Model, Interface Specification, CP-Spec, I-Spec, S-Spec, M-Spec, P-Spec, Interface layers, Itom and service level refinements, Service Level Objective (SLO), Service Level Agreement (SLA), Time-Synchronization Interface (TSI)	Clearer separation of interface abstraction levels: cyber-physical, Itom, and service level. Added a figure visualizing interfaces of a CS. Introduced, previously implicitly assumed, interface related concepts.
2.13.4	connecting system / gateway component / wrapper	Added ‘wrapper’ as another synonym of connecting system.