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] Universite Joseph Fourier Grenoble 1

[4] ResilTech S.r.l.

[5] Thales Nederland Bv

[6] European Network For Cyber Security Cooperatief Ua

D2.2 – AMADEOS CONCEPTUAL MODEL

(01.06.2014-31.03.2015)

Revision of the document:	2.6
Responsible partner:	UNIFI
Contributing partner(s):	all
Reviewing partner(s):	TNL, ENCS
Document date:	31/03/2015
Dissemination level:	PU

Change Records

Revision	Date	Changes	Authors	Status[1]
0.1	04/09/2014	Tentative ToC and partners assignment	UNIFI	NV
0.2	14/11/2014	Draft contributions to section 2 and 3.2.	TUW	NV
0.3	19/11/2014	Draft contribution to section 3.1	UJF	NV
0.4	26/11/2014	Merged contributions with Section 4	UNIFI	NV
0.5	14/12/2014	Further contributions to section 2 and section 3.2. Incorporated section 2 feedback from UNIFI	TUW	NV
0.6	15/12/2014	Further contribution to section 3.1	UJF	NV
1.0	19/12/2014	Merged contribution with further contribution to section 4. Version for AB members	UNIFI	NV
1.1	20/01/2015	Revision to Section 4 after the advisory board meeting.	UNIFI	NV
1.2	10/02/2015	Revision of Section 3.1 (ongoing)	UJF	NV
1.3	16/02/2015	Contributions to section 2, 3.2.	TUW	NV
2.0	17/02/2015	A new integrated version with the contributions from UNIFI, TUW and UJF after the advisory board meeting	UNIFI, TUW, UJF	NV
2.1	18/02/2015	Updates to Section 3.1	UJF	NV
2.2	18/02/2015	Addition of Section 4.5	TNL	NV
2.3	17/02/2015	A new integrated version with the contributions from UNIFI (Section 4), TUW (Section 2 and 3.2), UJF (Section 3.1) and TNL (Section 4.5).	UNIFI, TUW, UJF, TNL	NV
2.4	24/02/2015	Version ready for first review round	UNIFI	NV
2.5	4/03/2015	Version revised according to the comments received from the internal reviewers (first review round)	UNIFI, TUW, UJF	NV
2.6	31/03/2015	Version revised according to minor comments from internal reviewers (second-review round)	UNIFI, TUW	V

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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
RUMI	Relied Upon Message Interface
RUPI	Relied upon Physical Interface
RW	Read/Write

SBR	Scenario-Based Reasoning
SOA	Service-Oriented Architecture
SoC	Sphere of Control
SoS	System-of-System
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

Executive Summary

This document contains the definition of the AMADEOS conceptual model that resulted from the joint work in WP2 achieved during M10-M18. The conceptual model will be further improved and finalized in the remaining 18 months of the AMADEOS project to present its final version in D2.3.

Conceptual modelling is an iterative process that will be consolidated parallel to the other AMADEOS activities related to the definition of the AMADEOS architecture in WP3. We expect a seamlessly integration between conceptual modelling and the AMADEOS architecture thus proving the relevance of basics concepts and their relationships for our cross-domain SoS approaches (see D1.1) and possibly for the SoS community in general.

This document will be used as reference in all the remaining AMADEOS activities, at the end of which we expect to deliver a mature conceptual model that has already passed a rigorous acceptance test.

In the previous version of the conceptual model (D2.1) concepts and relationships have been defined and grouped into 12 distinct categories (e.g., fundamental SoS concepts, time, emergence, dependability and security) which resulted from the achievement of WP1 on SoS SOTA analysis and SoS cross-domain findings.

As advised by EU reviewers we have improved the preliminary version of the conceptual model by focusing mainly on core AMADEOS issues like treatment of time, evolution, dynamicity and multi-criticality.

The document starts by discussing the rationale for conceptual modeling System-of-Systems (SoSs). Next, it contains a revision of the concepts and relationships from deliverable D2.1 including also a more detailed specification on interfaces, the physical part of an SoS and its design intents and goals.

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 also provides a detailed discussion on interfaces among constituent systems and its environment.

The document discusses a semi-formal UML-like model to represent the AMADEOS basic concepts and their relationships. In particular, the document defines a SysML profile to represent the conceptual model of SoSs according to the 7 previously identified views. Modeling an SoS by using such a profile it is possible to ease the understanding of critical SoS aspects (related to the views) and to enable 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 showing an application of the profile to support modeling and detection of emergent behaviors and a traceability matrix among basic SoS concepts and the SysML profile. In the annexes we also report on the differences of the conceptual model presented in this document and its previous version (i.e., deliverable D2.1).

It is worth noticing that the conceptual model presented in this document is amenable till to Collaborative SoSs, i.e., SoSs for which there is no a clear objective and the entailed CSs fulfil shared or common interests. Noteworthy, we are not considering Virtual SoSs where CSs act even without knowing one another and without having a common intent.

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 message-based 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.

In the DoW of AMADEOS it is stated in the first line of the abstract The objective of this research proposal 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 MODELING

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 architecture in WP3.

Starting from the definition of basics 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 7 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 analyses activities which can be carry 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, a 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), 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, an additional study of the literature and discussions with experts. We have considered the valuable reviews received as result of the review meeting 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)
- providing a more detailed definition of interfaces of CSs.

The rest of the document is structured as follows. Section 3 builds upon the basic SoS concepts of Section 2 a graphical representation divided according to the 7 different views by also eliciting the relationships among concepts. Section 4 presents a SysML profile for SoSs organized in different components which follow from the previously identified views. A possible application of the proposed profile to an SoS instance is also shown for better evidencing the applicability of the conceptual model. Section 5 concludes the document, Annex A shows how the SoS profile can leverage the detection of emergent behaviors by means of an interface analysis and Annex B presents a further illustrative example to represent an emergent behavior. Annex C contains a traceability matrix among basic SoS concepts and elements of the proposed SoS profile. Details of changes to the previous version of this deliverable (D2.1) can be found in Annex D.

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 an entity 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.

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 interfaces 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 attributes 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 a controlled objects and/or human role players that interact to provide a given service.

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

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 own goal.

An example for a prime mover could be a legitimate user or a malicious user that uses the system for his own advantage. A human who is a prime mover is 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.***

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 behaviour 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 behaviour 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 can 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 (distance) 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 followed 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 time standard 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 only 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 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.

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 behaviour.

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 behaviour 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 behaviour 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 of 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 BEHAVIOUR

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 our 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 behaviour 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 behaviour) 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.***

2.4.2 Behaviour

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

> ***Behaviour: 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 Behaviour: 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 behaviour or it may demonstrate a behaviour that is unintended (e.g. erroneous behaviour).

Service: The intended behaviour of a system.

Function: The intended behaviour of a stateless system.

The service specification must specify the intended behaviour of a system. In non real-time systems, the service specification focuses on the data aspect of the behaviour. 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 as depicted in Table 2. This classification has been taken from the DSOS project ([20], p19).

Table 2: Message Classification

Attribute	Description	Antonym
Valid	A message is valid if its checksum and contents are in agreement.	Invalid
Checked	A message is checked at the source (or, in short, checked) if it passes the output assertion.	Not Checked
Permitted	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.	Not Permitted
Timely	A message is timely if it is in agreement with the temporal specification.	Untimely
Value correct	A message is value-correct if it is in agreement with the value specification.	Not value correct
Correct	A message is correct if it is both timely and value correct.	Incorrect
Insidious	A message is insidious if it is permitted but incorrect.	Not insidious

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 3 presents the summary of our analysis.

Table 3: 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 behaviour 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

System-of-Systems (SoS) are built to realize novel services that go beyond the services that can be provided by any of the 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 4.1. 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. [40, 41, 42, 43, 44]. 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.

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 [24, p. 217]. 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.

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 level *n* is linked vertically to its controlling system at level *n+1*, the level above, 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* [24, p.196]. In a formal hierarchy there are no direct horizontal interactions among the subsystems of a given 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 level *n* that lead to the formation of a *whole* with its own characteristic properties at the level above, level *n+1*. Koestler [45, p.341] 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 externally (upwards and horizontally) acts as a whole but internally (downwards) is established by the interactions of its parts (i.e., the interactions among the holons of the level below).***

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. The lower level, called the *micro-level,* is often called the *level of the parts* and the higher level, the *macro-level*, is the *level of the whole* where the novel emergent phenomena appear. 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 taken from [32]:

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

According to the above definition of emergence, the emergent phenomenon must be *new* with respect to the non-relational phenomena of the parts, but not with respect to the *conceptual landscape* 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.

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, *weak emergence* and *strong emergence*. This classification can only be accomplished after a thorough analysis of the emergent phenomenon has been carried out.

> ***Weak emergence: An emergent phenomenon that is observed at a macro level is weakly 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).

Strong Emergence: An emergent phenomenon that is observed at the macro level is strongly 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 *weakly emergent* and *strongly 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 *strongly 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 *weakly emergent*.

There are some strongly 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 strongly emergent phenomena.

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 behavioural 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 behaviour of the transistors (the holons) is considered the *same* if the behavioural 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.

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

Downward Causation: The phenomenon that some novel higher-level properties cannot be reduced to mere physical phenomena and have causal powers to control the lower-level process from which they emerge.

There seems to be no universal agreement on the explanation of the phenomenon of downward causation.

2.6.2 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 (behaviour) 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 *weak 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 4 depicts four cases of emergent behaviour that must be distinguished in an SoS. *Expected and beneficial* emergent behaviour is the normal case. Un*expected and beneficial emergent* behaviour is a positive surprise. *Expected detrimental emergent* behaviour 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 behaviour*.

Table 4: Classification of Emergent Behaviour

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

2.7 PROBLEM SOLVING

This Section will be expanded in the final version of the conceptual model.

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.

In the next phase of the AMADEOS project we plan to investigate which phases of the problem solving process can be supported by an SoS and by what means an SoS can contribute to problem solving.

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].

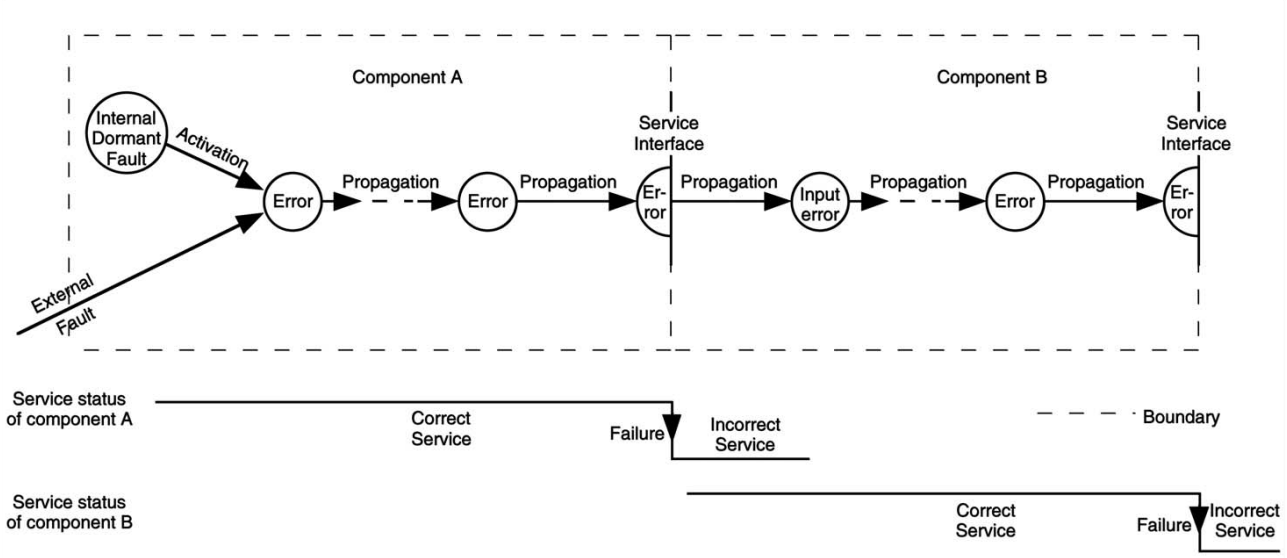

Figure 1: The chain of threats: a fault in component A activates and generates an error; errors propagate until component A fails; the failure of A appears as an external fault to B [4].

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 behaviour deviation from the intended system behaviour.***

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 behaviour that deviates from the intended behaviour, and may return to the intended behaviour at some later point in time:

> ***System outage: The interval during which the system has failed, i.e. where the behaviour of the system deviated from its intended one.***
>
> ***System restoration: The transition from system failure to intended system behaviour.***
>
> ***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 behaviour, 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 behaviour 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 authorisation (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.***
>
> ***Authorisation: Authorisation 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 “authorisation” 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.***

Authorisation 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 behaviour 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 behaviour, 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).

The goal of managed evolution in AMADEOS is maximizing business value:

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

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 behaviour as compared to the desired behaviour.***
>
> ***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: The property of an entity that is constantly changing in terms of offered services, built-in structure and interactions with other entities.***

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

> ***Reconfigurability: The ability to repeatedly change and rearrange the components of an entity in a cost-effective way.***

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.

[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

> ***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.

> ***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, behaviour 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

We start defining what is an architecture.

> ***Architecture: The manner in which the components of a system are organized and integrated. It defines the structure, behaviour, and more views of a system.***

The architecture 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 properties of a design object.***

There are several methodologies to design a system.

Hierarchical Design: An approach to design that leverages the natural logical hierarchy.

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 combining smaller subsystems.

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, e.g. Internet applications that are forward compatible given changes in business processes and strategies as well as in technology (digital, genetic, information-based and wireless). Principles of evolvability include modularity, updateability and extensibility. Design for evolution aims to achieve robust and/or flexible architectures.

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.

Starting from the item definition we can now define the concept of a tool:

Tool: A tool is any item that can be used to achieve a goal.

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 organisation.

> ***Administrative domain: An organisation 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 behaviour or configuration of another party, which is obliged to conform to these demands.***

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: Property or indicator used to quantitatively describe an aspect 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 I, 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].

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.

2.13 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 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 place over 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 properties can be characterized at different levels of abstraction:

- **Physical Level:** At the physical level information is represented as data items (e.g., a bit-pattern in cyberspace, or things/energy in the physical world) that are transferred among interacting systems during the IoD. The progression of time enables this transportation of information. Hence time is a fundamental property of interfaces and should not be abstracted away at any interaction abstraction level. In general, a physical channel can be bidirectional (e.g., Carrier Sense Multiple Access with Collision Avoidance - CSMA/CA). Some important properties at this level of abstraction are: signals (prearranged data representation), transmission medium, characteristics of connectors, frequencies, bit rates, energy levels, …
- **Message Level:** At this level of abstraction we are concerned with the timely exchange of messages by unidirectional channels across interfaces. Bidirectional channels can be expressed at the next higher abstraction level by using two unidirectional message channels in opposing directions. Messages carry information that is represented by data. A message channel is characterized by sender, recipients, temporal properties and the Itom(s) contained in the message.
- **Service Level:** At the service level, the interface exposes the behaviour of a system. Only the interdependencies of message channels are explicitly described at the service level, not each message channel to every connected system individually For example, at the message level a database service consists of a request and a response message channel where each request is followed by a timely response message.

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 behaviour can be reduced to the specification of the system interfaces [2].

Interface Specification: The interface specification defines at all appropriate levels of abstraction the interface properties, i.e., how information is exchanged at that interface.

Interface Physical Specification (P-Spec): Part of the interface specification at the physical abstraction level.

Interface Message Specification (M-Spec): Part of the interface specification at the message abstraction level.

Interface Service Specification (S-Spec): Part of the interface specification at the service abstraction level.

Depending on the system complexity and its underlying architecture the interface specification needs to be considered at appropriate levels of abstraction. 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 an—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 interaction among these two interface environments.***

2.13.1 Physical Interfaces

At the physical level interfaces are realized by energy transformers that are able to (1) translate observations from the physical environment to the cyber-space, and (2) cause effects that originate from the cyber-space in the physical environment.

> ***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.***
>
> ***Actuator: An actuator is an interface device that accepts data and trigger information from an interface component and realizes the intended physical effect in the controlled object.***
>
> ***Transducer: An interface device converting data to energy or vice versa. The device can either be a sensor or an actuator.***

Physical interfaces enable the interaction with the time-sensitive physical environment. Hence they are time-sensitive as well, i.e. the transducer latency and jitter have a profound impact on the timely accuracy of an observation or the timely effect on the physical environment.

Sensor-fusion and state estimation [55] are well researched techniques to improve on the faithfulness of sensor observations.

Interaction with the physical environment always occurs with the intent to directly influence or measure the physical attributes of things (e.g., increase the wheel rotation speed). Via prearrangement (signals, Section 2.2) we use this direct influence on things to transport messages.

2.13.2 Message-based Interfaces

The interface specification at the message level 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 type of a port specifies whether message handling should adhere to the periodic, sporadic or streaming semantics.
- **Temporal Properties:** The temporal properties determine the temporal behaviour 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) are 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 to define their explanation.

Interface Model: The interface model contains the explanation of each Itom sent or received over the interface.

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, .e. 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.

Compatible Interfaces: Interfaces are compatible if their interface specifications define the same syntactic- and semantic specifications, and the transport specifications agree w.r.t. port direction.

Formal methods on formally defined interface specifications can be used to verify whether interfaces are compatible.

2.13.3 Service-based Interfaces

Components 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.

The definition and evolution of service-based interface specifications can be formally described by contractual based Service Level Agreements (SLAs) among operators of services.

Service Level Agreement (SLA): A SLA is a document that defines the relationship between two parties: the provider and the recipient [The Service Level Agreement Zone[3]].

[3] http://www.sla-zone.co.uk/index.htm

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].

> ***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 System-of-Systems Interfaces

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).***

We distinguish two types of external CS interfaces: Relied Upon Interfaces (RUIs) (see Section 2.1 and 3.2) and *utility interfaces*.

> ***External Interface: A Constituent System (CS) is embedded in the physical environment by its external interfaces.***
>
> ***Utility Interface: An interface that does not need to be considered for the integration of SoSs.***

The purposes of the utility interfaces are to (1) configure and update the system, (2) diagnose the system, and (3) let the system interact with its remaining local physical environment which is unrelated to the 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).

> ***Configuration and Update 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 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: 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), to coordinate multicast communication, and/or to introduce emerging services.***

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 2 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*, *controlled objects* and/or *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 exhibit (from CS to RUMI) a *Relied Upon Message Interface (RUMI)* that allows the exchange of (from RUMI to information) information.

As it name indicates, RUMI relies on (from RUMI to message) messages, that are the ones that contain (from message to information) the information. A message follows (from message to Syntactic specification and Semantic specification) a Syntactic specification and a semantic specification.

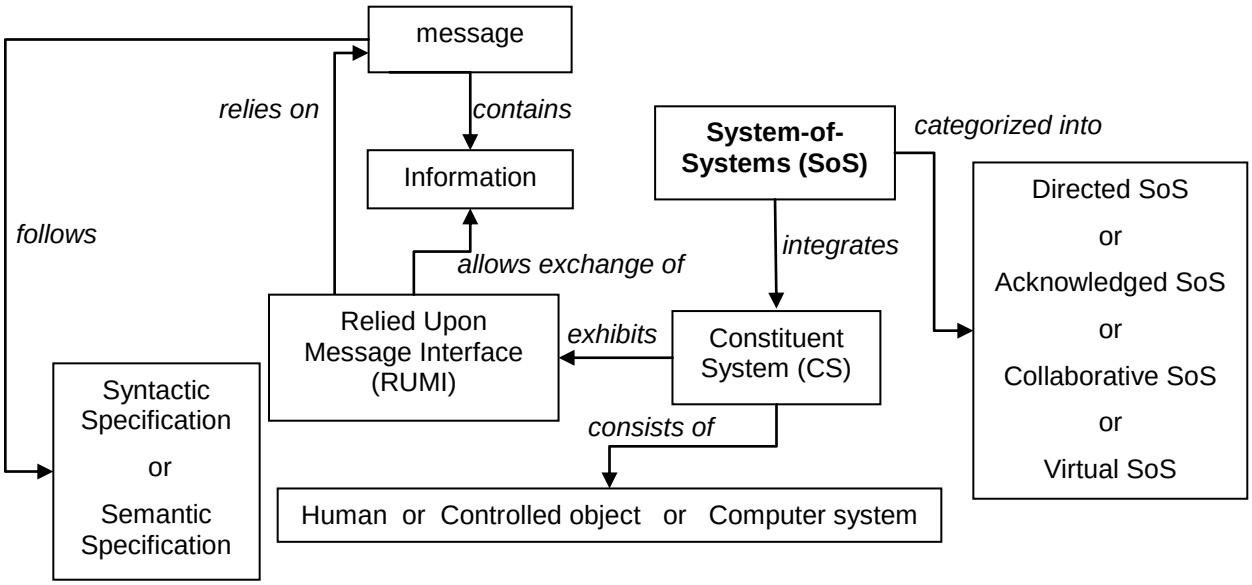

Figure 2 - Semantic relationships. Viewpoint of structure

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 3. 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.

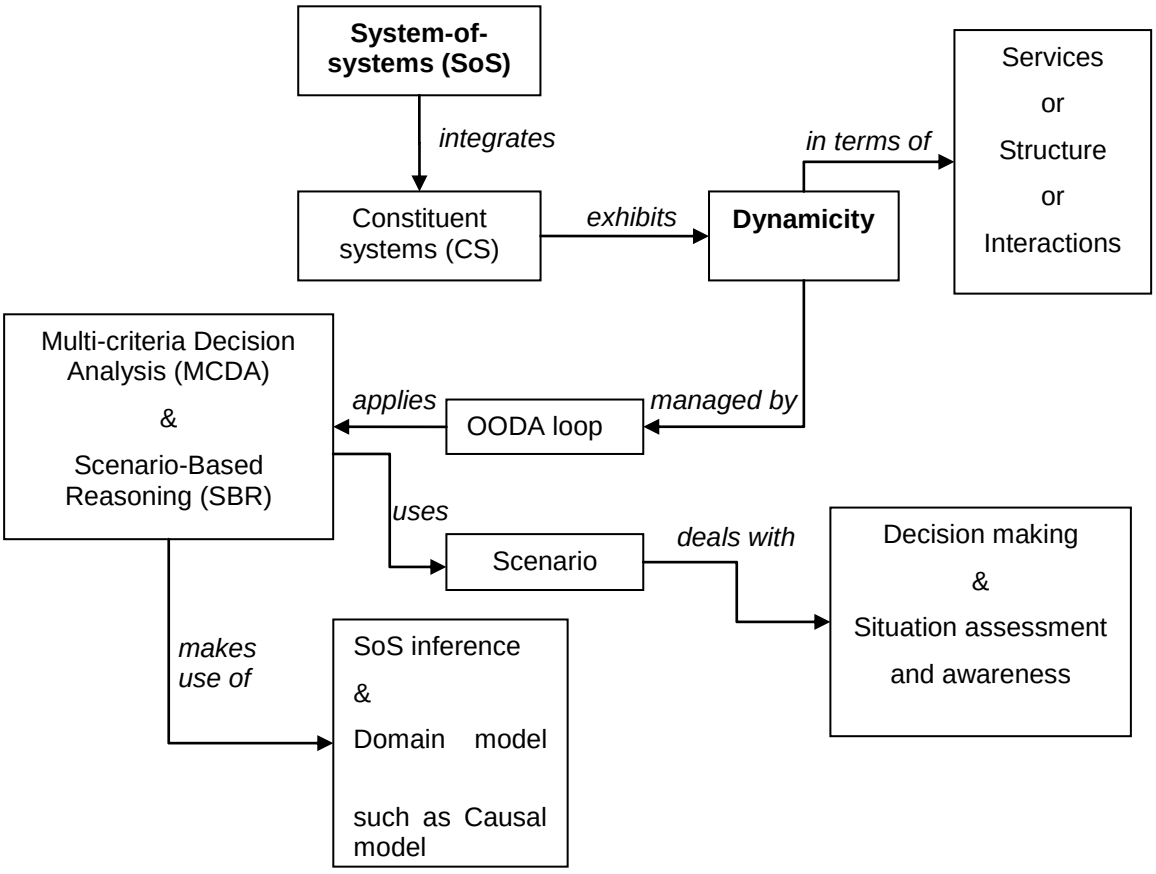

Figure 3 - Semantic relationships. Viewpoint of Dynamicity

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 4 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.

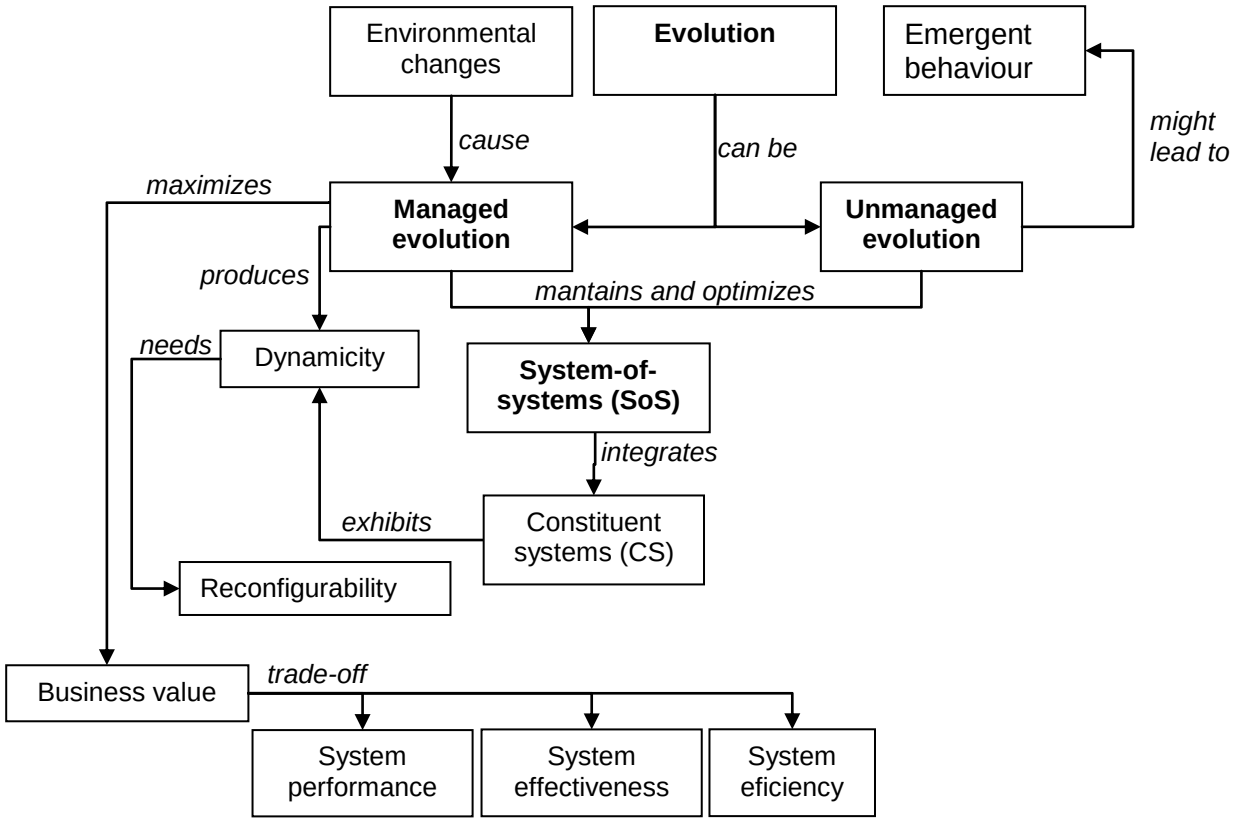

Figure 4 - Semi-formal model. Viewpoint of Evolution

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 5. 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.

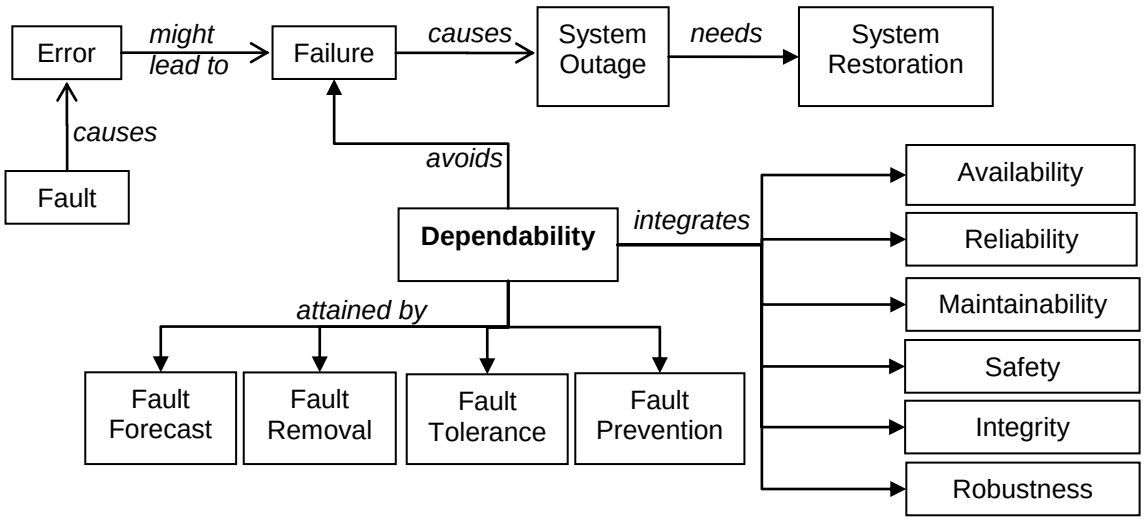

Figure 5 - Semantic relationships. Viewpoint of Dependability.

Semantic relationships among security concepts are shown in Figure 6. Security integrates all the dependability attributes (see Figure 5) 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 authorisation, which is provided by a reference monitor. Authorisation assigns permissions, which are defined in a security policy. A security policy relies on trusted systems, which encompass hardware, software or human components.

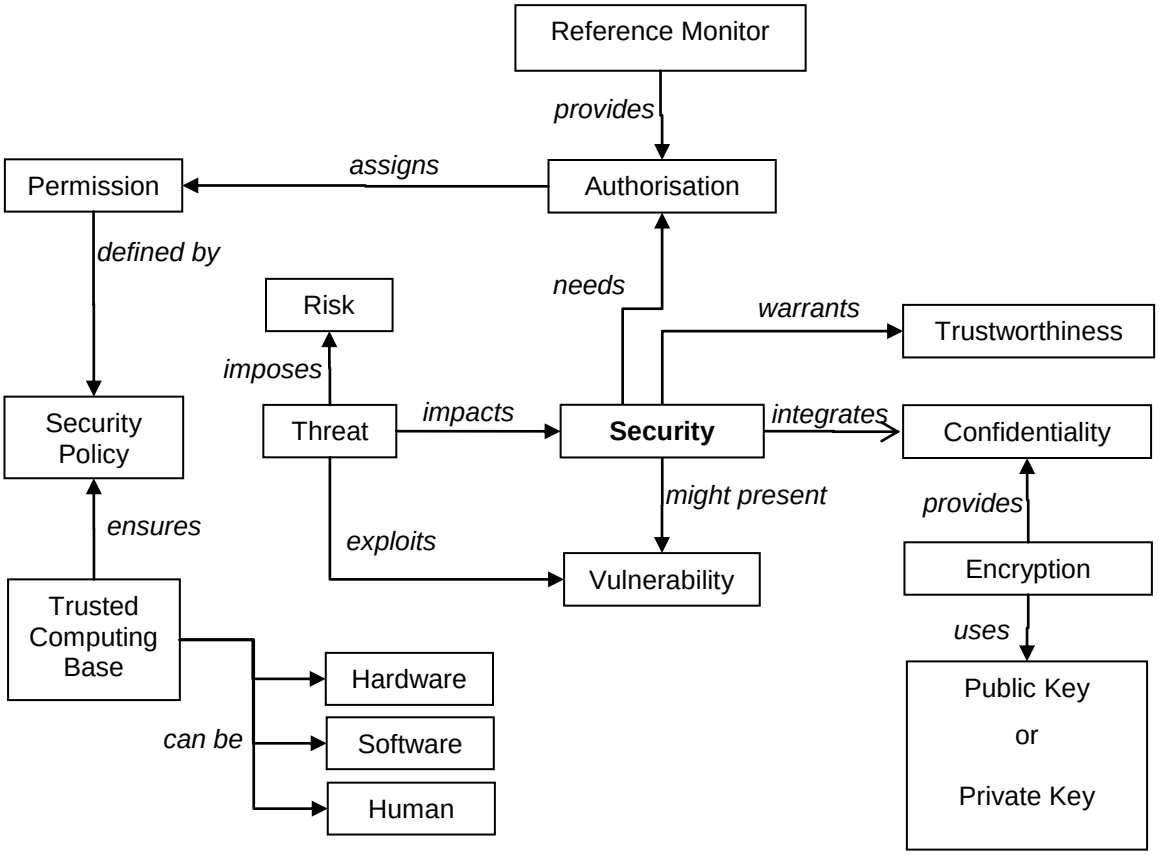

Figure 6 - Semantic relationships. Viewpoint of Security.

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 7 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*.

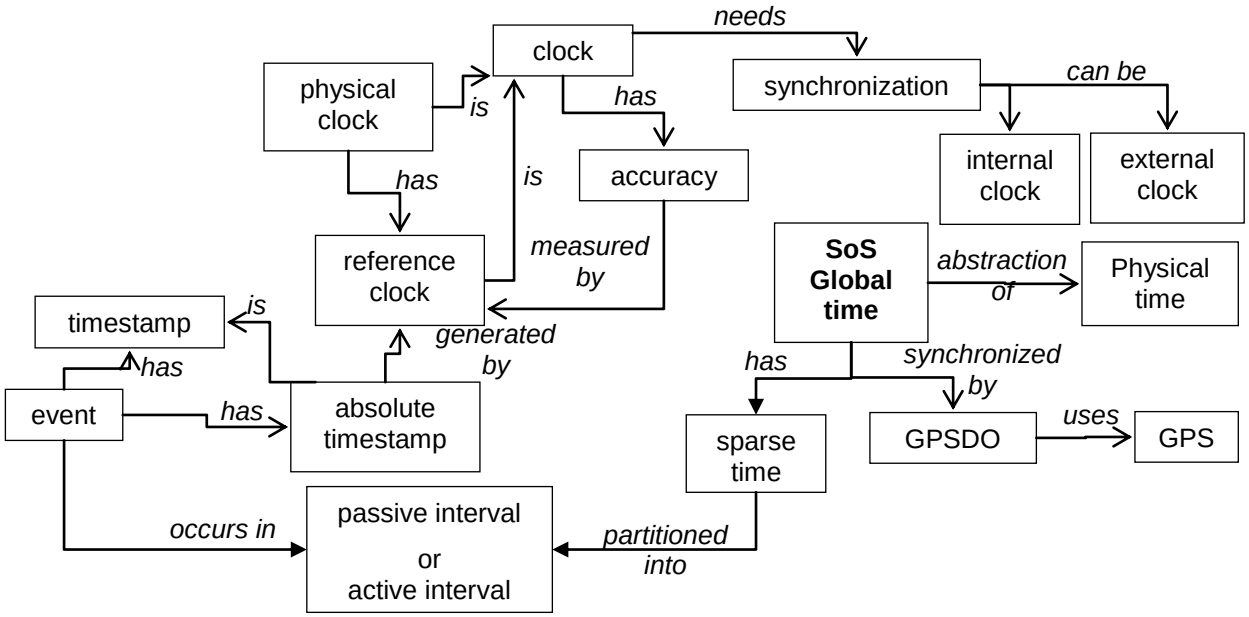

Figure 7 - Semantic relationships. Viewpoint of Time.

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 8. 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.

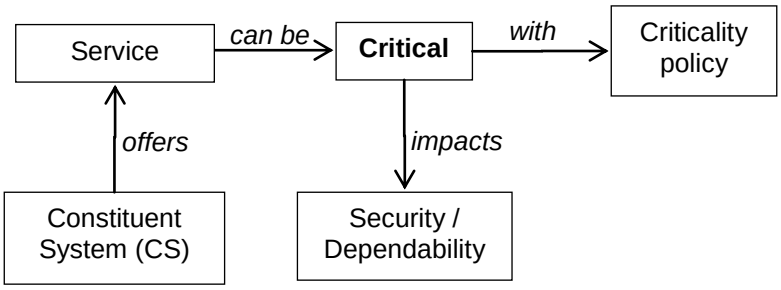

Figure 8 - Semantic relationships. Viewpoint of Multi-Criticality

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 9 presents the semantic relationships between emergence-related concepts. Emergence can be expected or unexpected, beneficial or detrimental, and weak or strong.

Emergence produces a resultant phenomenon that can appear in the macro level (i.e. the SoS) or the micro level (i.e. the CS).

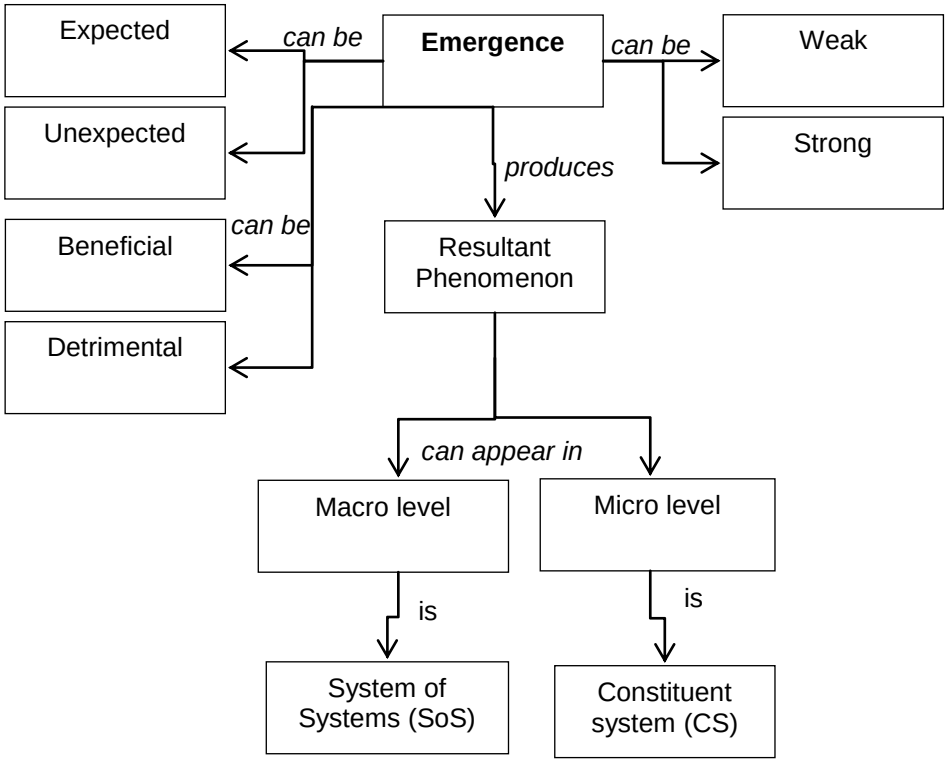

Figure 9. Semantic relationships. Viewpoint of emergence.

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 on. 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 *behaviour* 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 behaviour 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* 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 and will be investigated further in later phases 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 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 a direct implication for 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 directly among CSs over cyber channels, and communication that takes place indirectly among CSs over stigmergic channels in the environment of 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 10 gives a structural overview of information transfer in SoSs 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. CS A is the sender of the Itoms I_1 and I_3, while it is recipient of the Itoms I_2 and I_6. I_1 and I_2 are Itoms that are transferred over cyber channels realized by a computer network (e.g., the Internet). I_3 and I_5 are Itoms sent over a stigmergic channel under the control of a physical environment. The Itom I_4 is associated to the same property (of a thing) in the physical environment that CS A modifies by sending Itom I_3. The same applies to Itom I_6 which is associated to the transmission of Itom I_5. In the Figure we also separated the physical environment from the cyber space.

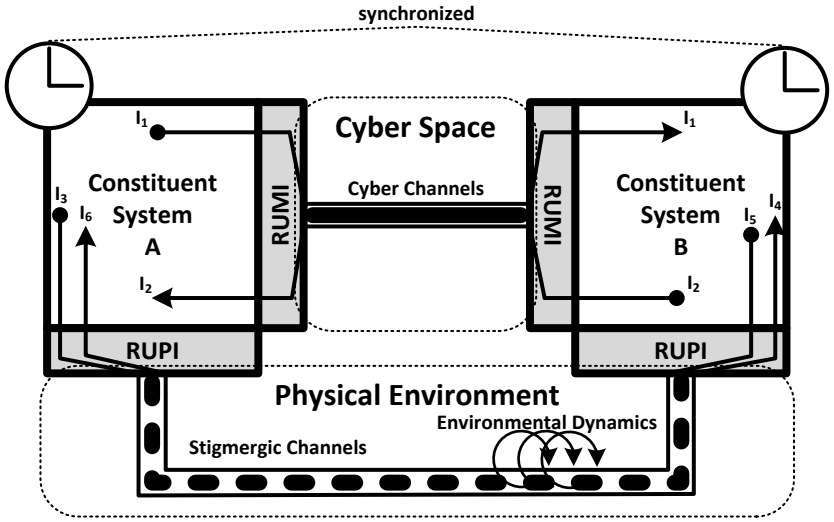

Figure 10: Information Transfer over Relied Upon Interfaces in Systems-of-Systems.

3.2.2 Cyber Space and Cyber Channels

A cyber channel describes direct 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 Itoms 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 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 (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 affects the state of the environment immediately. Similarly, any measurement taken by a receiving CS is also about the present state of the environment.
- **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.

Stigmergic channels do not directly transport an Itom 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 Itom usually is related to a sent Itom which justifies the use of the concept channel in the context of stigmergic CS interaction.

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[4] 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 clear 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 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,

[4] Mathworks Simulink, Maplesoft MapleSim, OpenModelica, Wolfram SystemModeler, xcos, …

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 that CSs shall be "*independent and operable"*. Hence, RUIs of a CS have to maintain a boundary for establishing a Fault Containment Region (FCR) and an Error Detection Region (ED R) such that CSs fail independently from each other and errors do not propagate undetected from one CS to another.

4 UML-LIKE 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 consider 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).

4.2 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 9 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.

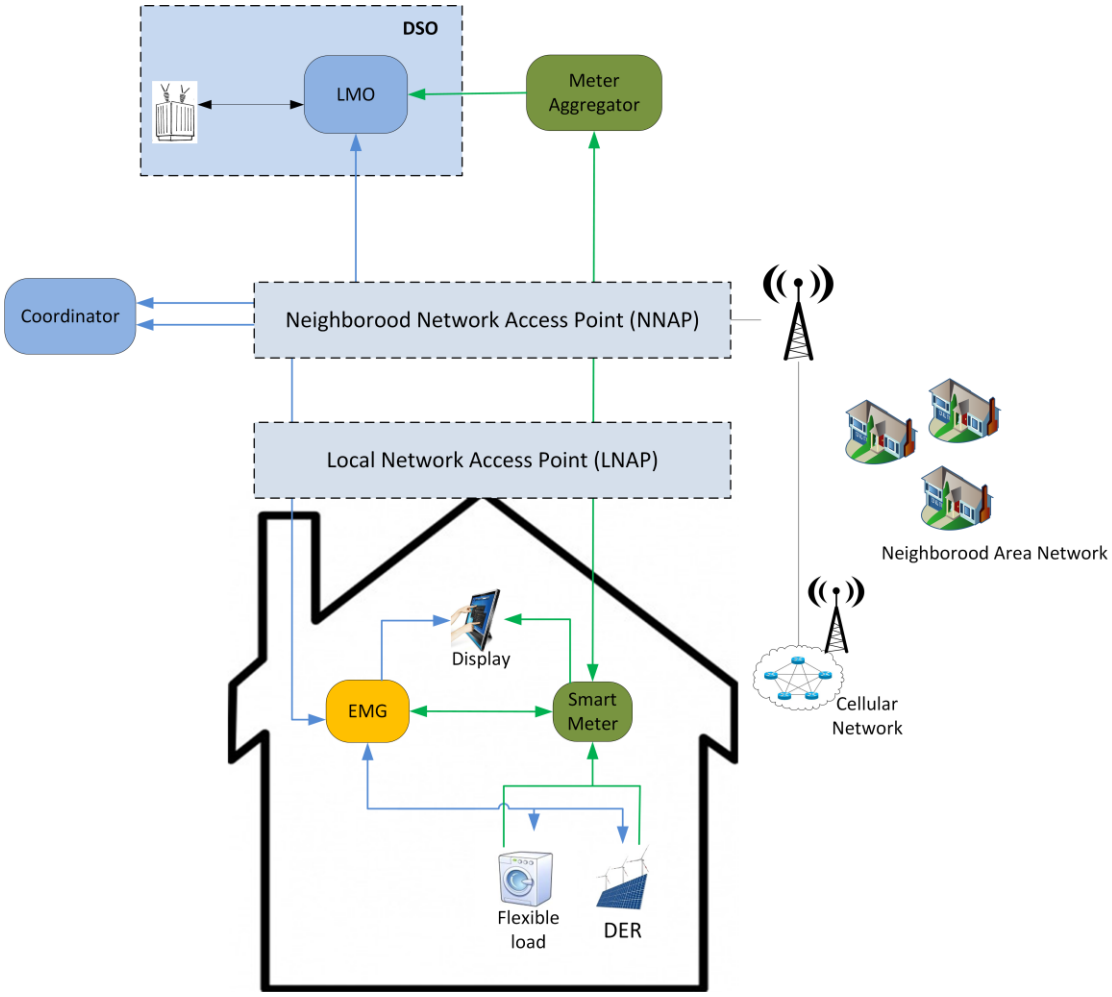

Figure 11: Smart Grid case study

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, MARTE [56] and CHESS profile [57]. MARTE is a UML profile to develop Real-Time and Embedded Systems and CHESS is UML conceptual model for dependability evaluation developed in the context of ARTEMIS-JU “CHESS” project.

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 defined in Section 2 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 Time*: provides the fundamental elements to describe time concepts.
- *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 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 comes from a direct mapping to the views defined in Section 3 except for *SoS Architecture, SoS Communication* and *SoS Interface* which all together implement the Structure view. Noteworthy, due to the nature of emergence and dynamicity concepts, it is not sufficient to represent their main characteristics by using static elements of a profile. To this end, beyond defining specific SysML profile packages (i.e., *SoS Dynamicity* and *SoS Emergence*), we also model dynamicity and emergence by instantiating the Structure view packages with an operational SoS, e.g., in the Smart Grid domain.

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 supports the editing of UML and SysML profile and offers a very advanced support to define SysML profiles.

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 12). 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 metadata "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, and periodic. A system can be influenced by an "**Architectural style**" and can provide communication "**Interfaces**". A "**Subsystem**" is a subordinate system that is part of a system and is related to "System" by a composite relation.

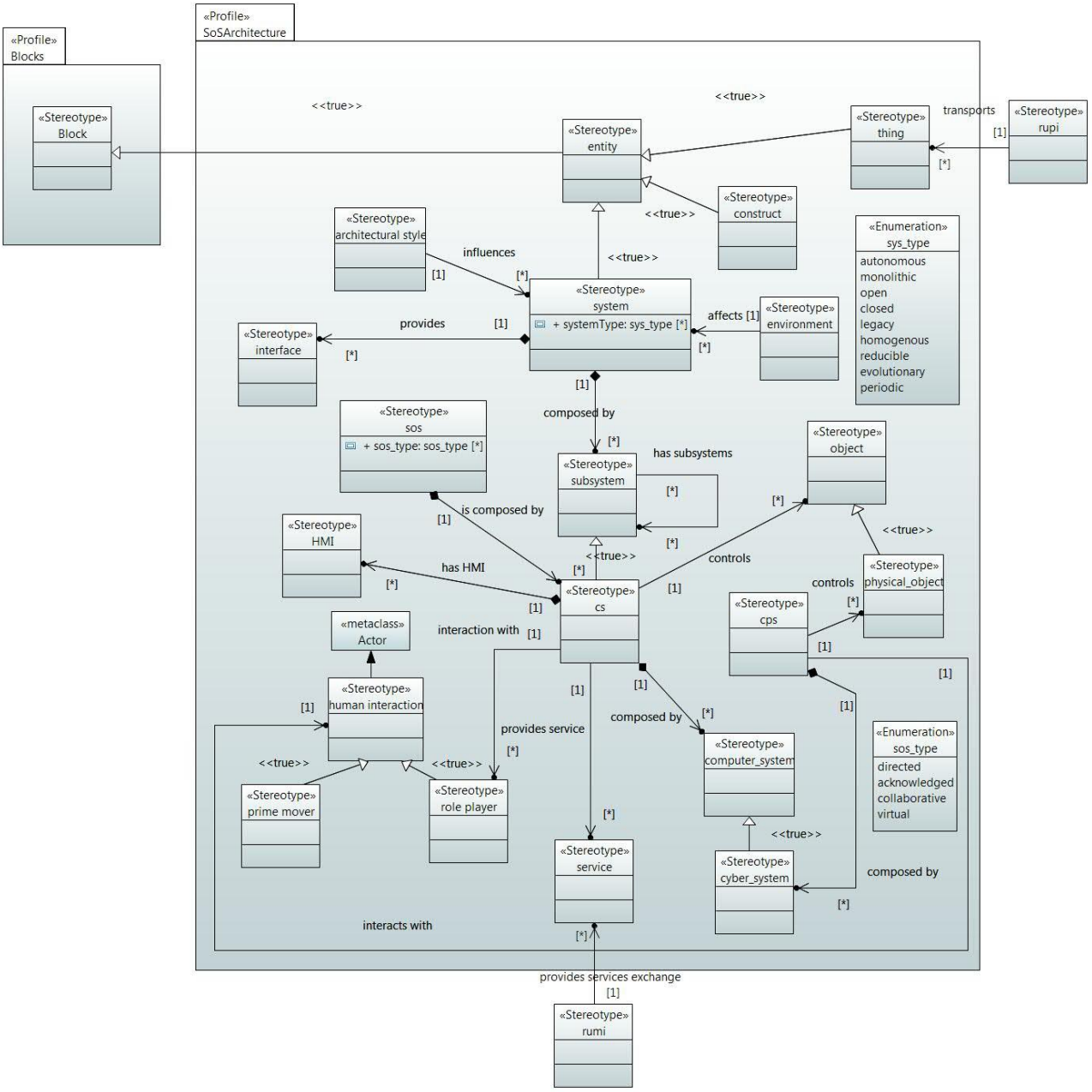

Figure 12: SoS Architecture package

As defined in Section 2, a Constituent System or "**cs**" is an autonomous subsystem of an SoS, consisting of computer systems ("**computer_system**") and possibly of controlled objects and/or human role players that interact to provide a given service. In this profile "cs" is a Stereotype that extends the property of "Subsystem", is part of an SoS, controls "**objects**" and has interactions with "**role players**" (all defined as a Block Stereotypes). In addition, a CS can have a component that realizes the human-machine interface, or "HMI", and can provide "**services**". Otherwise "**rumi**" Stereotype (that is introducing in SoS Communication package) 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.

"**sos**" Stereotype represent the integration of a number of constituent systems 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**") consists of a "**cyber_system**", a "**physical_system**" and the possibility of interacting humans ("**human_interaction**").

Using this package it is possible to represent the topology of any System of Systems. Now we show how to use 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 13 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.

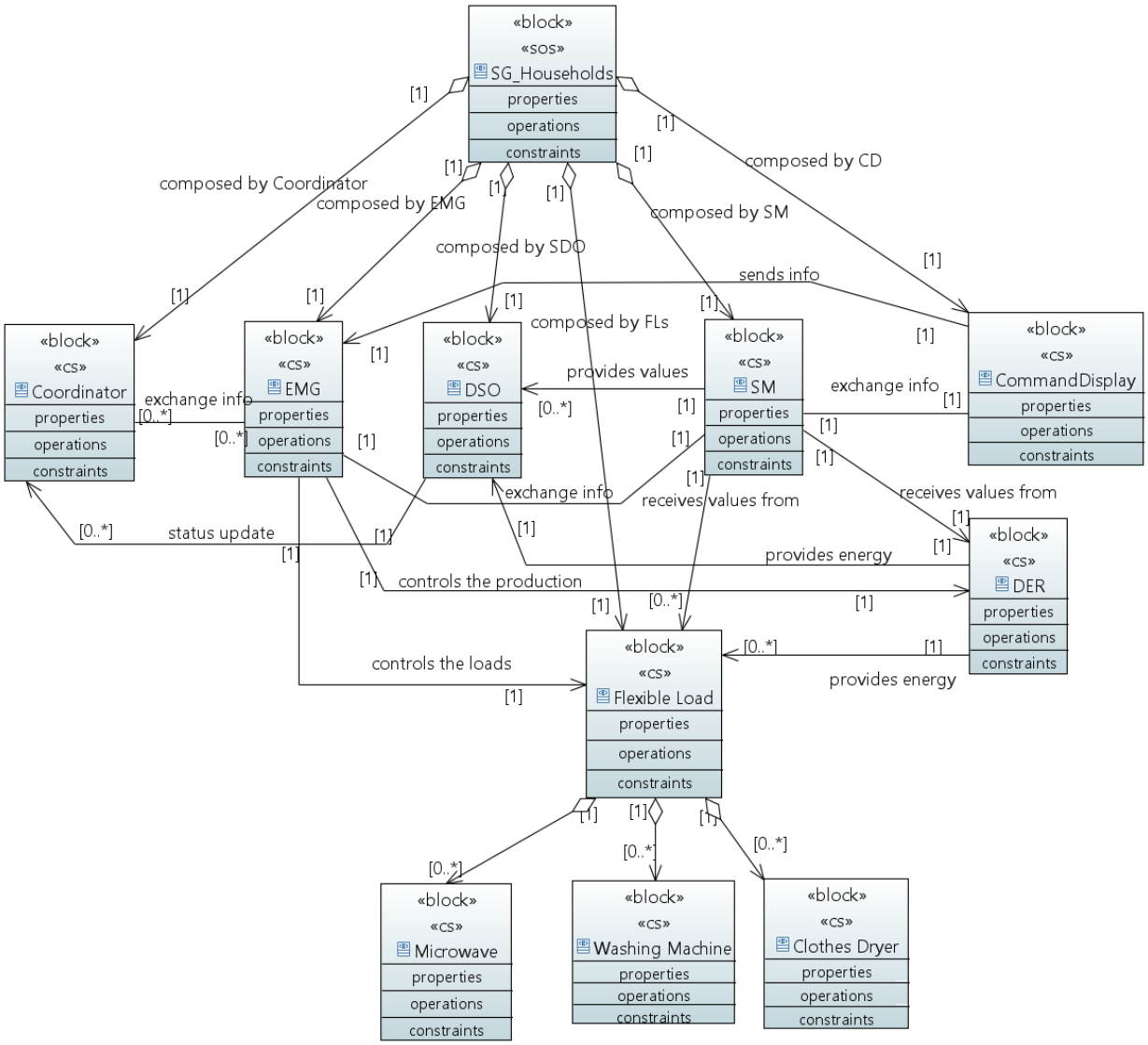

Figure 13: Smart Grid Household Block Definition Diagram

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.

"*SoS Communication*" package (see Figure 14) 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 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 the lifelines that exchange information.

A "**rumi**" represents a message interface for the exchange of information among two or more CSs and extends the "**rui**" Stereotype. If the messages are exchanged through the RUMI, physical elements are exchanged among the CSs of an SoS through the "**rupi**", where physical elements are things or energy. Furthermore "**monitoring_CS**" represent a CS that monitors the information exchange across the RUMI.

In this package we introduce the concepts of hidden channel also called 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 following Stereotypes: environment (already shown in *SoS Architecture*) and *state_space*. RUI can affect the system environment and the environment can have effect on the system state space.

As defined in Section 2, a message is a data structure that is composed by a "**data_field**", a "**header**" and a "**trailer**". 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.

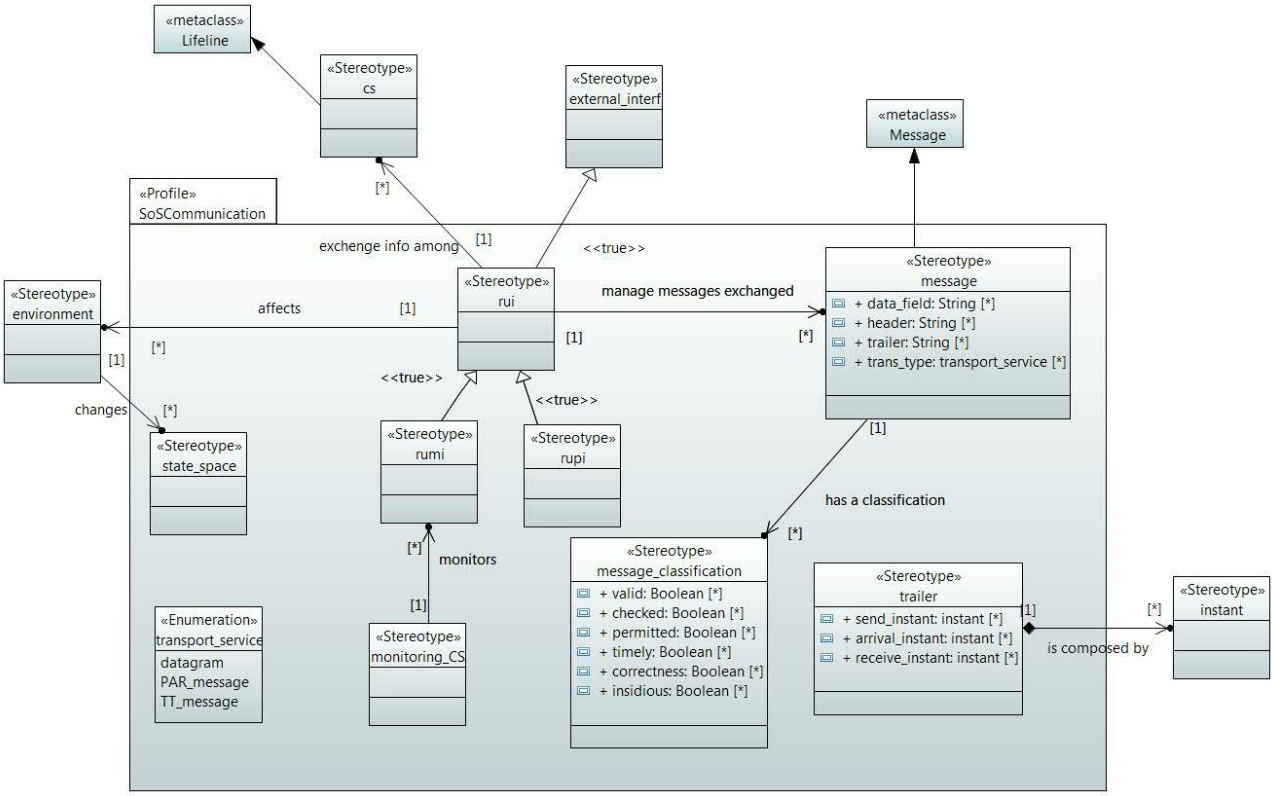

Figure 14: SoS Communication package

An application example of the main SoS communication concepts are shown in Figure 15. 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 14 and they can be displayed using a constraint or a comment box. Figure 15 shows the message properties using a comment box (e.g., "data_field"=2kW, "header"=wm to EMG, "trailer"=t1, "trans_type"=PAR_message).

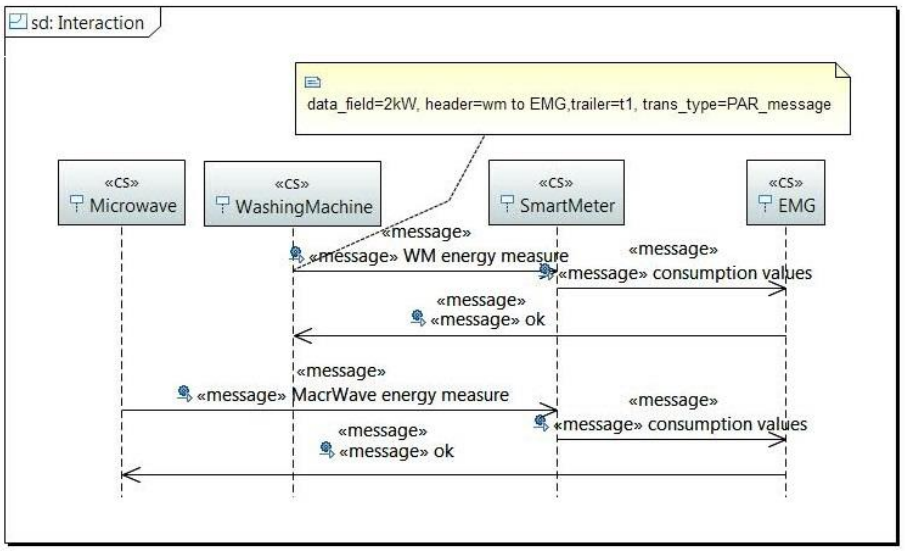

Figure 15: Message exchange between constituent systems of a Smart Grid

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 16 shows the SoS Interface Package.

An interface (described within the *SoS Architecture* package) can be an "**internal_interface**" or 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 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**", Diagnosis Interface "**d-interface**" and Local I/O Interface "**local_IO_Interface**". For the Local I/O Interface we show three different types of interfaces that do not need to be considered for the integration of SoSs because they are hidden behind RUIs: "**sensor**", "**actuator**" and "**transducer**".

An interface can have a specification ("**interface_specification**") with different kind of levels: Interface Physical Specification ("**P-Spec**"), Interface Message Specification ("**M-Spec**"), Interface Service Specification ("**S-Spec**").

A "**channel**" is a link that transports information among the systems, it is represented by a Stereotype and it can be physical or logical ("**physical_channel**", "**logical_channel**"). The information are exchanged through an "**interaction**" and an interaction has the following attributes described in Section 2: "**transferred_info**", "**temporal_property**" and "**dependability_req**". Interaction is represented by a Stereotype and the attributes are represented by properties.

A channel can transport physical entities, services, or messages. A "**channel_model**" describes the effects of the channel on the transferred information.

If the information are transmitted through messages we have identified, for each message a syntactic specification (the named syntactic units of the message payload) , a semantic specification (the explanation) and a name. The "**semantic_specification**" and the "**name**" establish the "**message_variable**" and an "**interface_model**" contains the explanation of the semantic interface. For this type of interface we define "**transport_specification**" as a Stereotype that extends "**m-spec**". 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).

When an interface transport messages uses ports that have some properties. A Message-based Interface Port ("**interface_port**") is represented as a Stereotype that extends FlowPort Stereotype of MARTE (contained in Generic Component Model Package). A MARTE FlowPort has FlowProperties that are extended from our "**property**". This Stereotype represents the properties of a FlowPort with the addition of the following properties: direction ("**FlowDirectionKind**"), size ("**size**"), type ("**type**"), temporal properties ("**temporal_prop**"), dependability properties ("**dependability_prop**").

If the interfaces are service-based, this means that the system provides many services. We have introduced the Stereotype "**sla**" Service Level Agreement in order to represent the document, which defines the relationship between two parties: the "**provider**" and the "**recipient**". In addition we have create 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 ("**request_instant**") at the reservation allocation instant ("**allocation_instant**") will remain allocated until the reservation end instant ("**end_instant**").

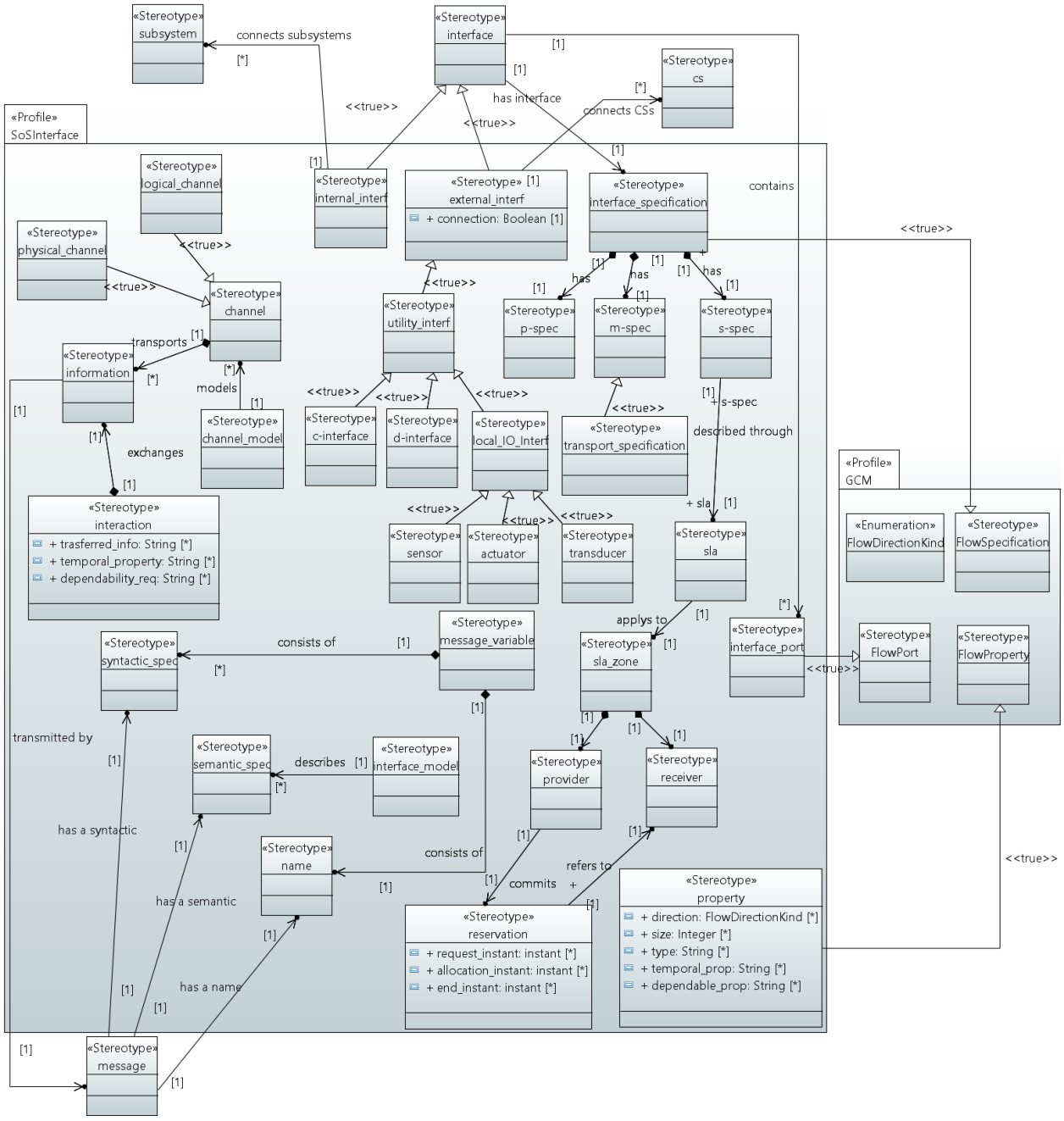

Figure 16: SoS Interface Package

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. All the Stereotypes involved in the definition of this package are defined as blocks and for a better understanding we have chosen to not represent the extension relation between all the stereotypes and “Block” (“Evolution” Stereotype is the only one to show this kind of relation).

The main SoS concepts are modelled within the “*SoS Evolution*” package of our SoS profile. Figure 17 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.

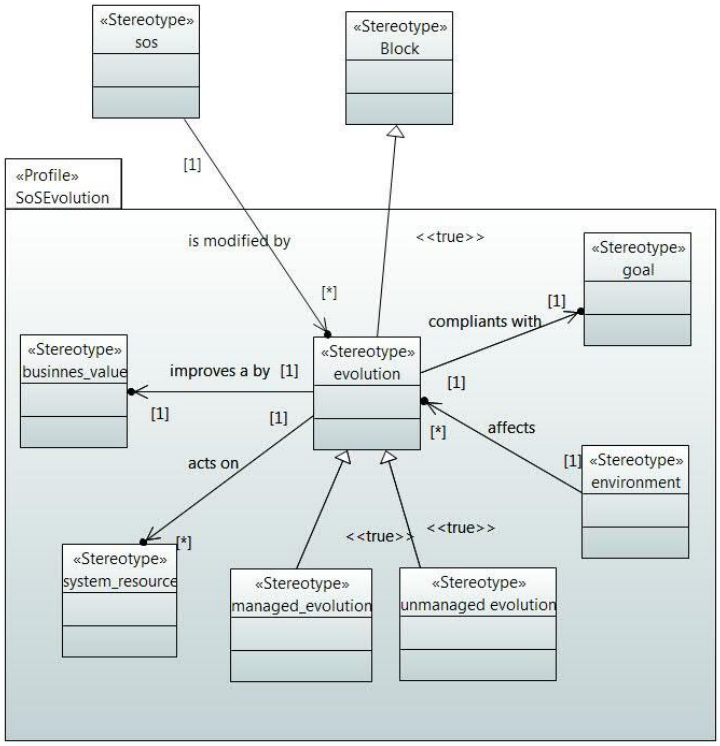

Figure 17: SoS Evolution package

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 18) with a new device (e.g. a Smartphone) by which it is possible showing consumption values and providing new interactive actions.

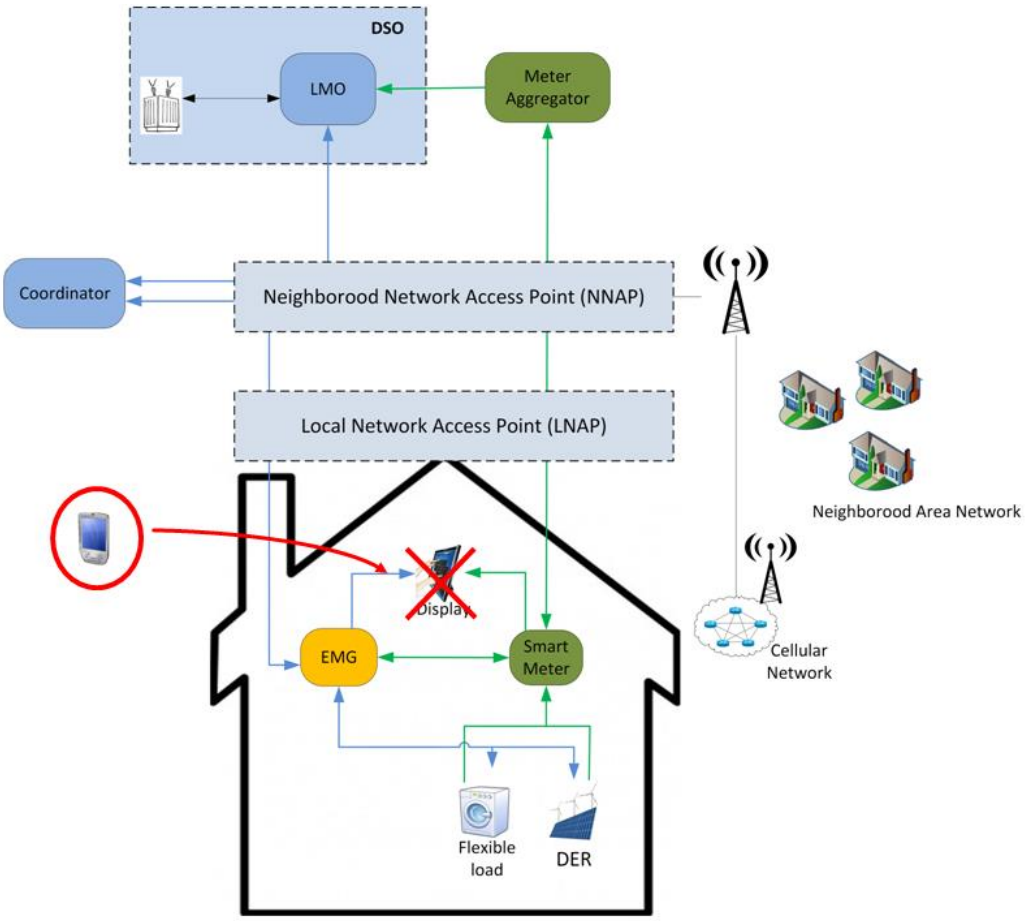

Figure 18: Example of system evolution in a Smart Grid Household

Figure 19 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".

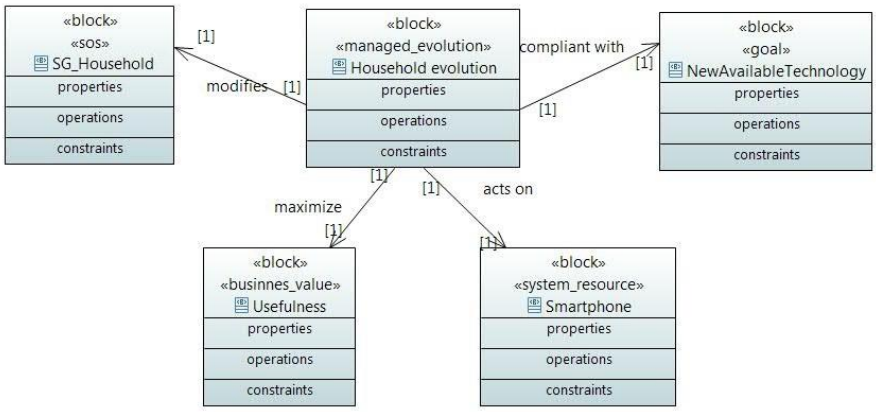

Figure 19: Smart Grid evolution modeling

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 20 we have introduced the concept of **"dynamicity"** which belongs to the already defined stereotype **"entity"**, thus applying 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.

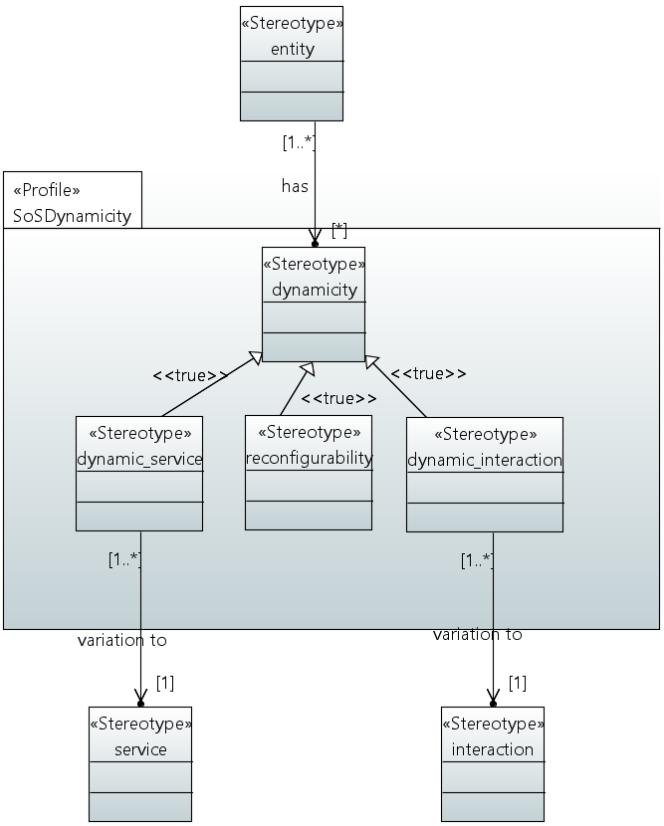

Figure 20 - SoS Dynamicity package

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 21). 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.

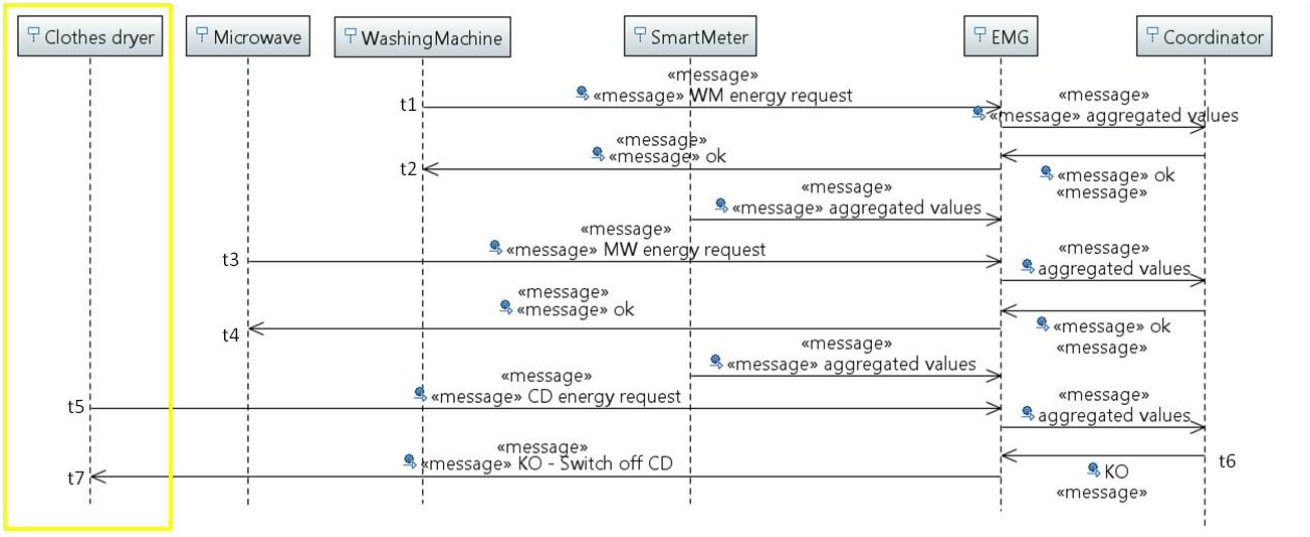

Figure 21: Dynamicity behavior of a Smart Grid

The illustrative example in Figure 21 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 21 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 component 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 22, 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.

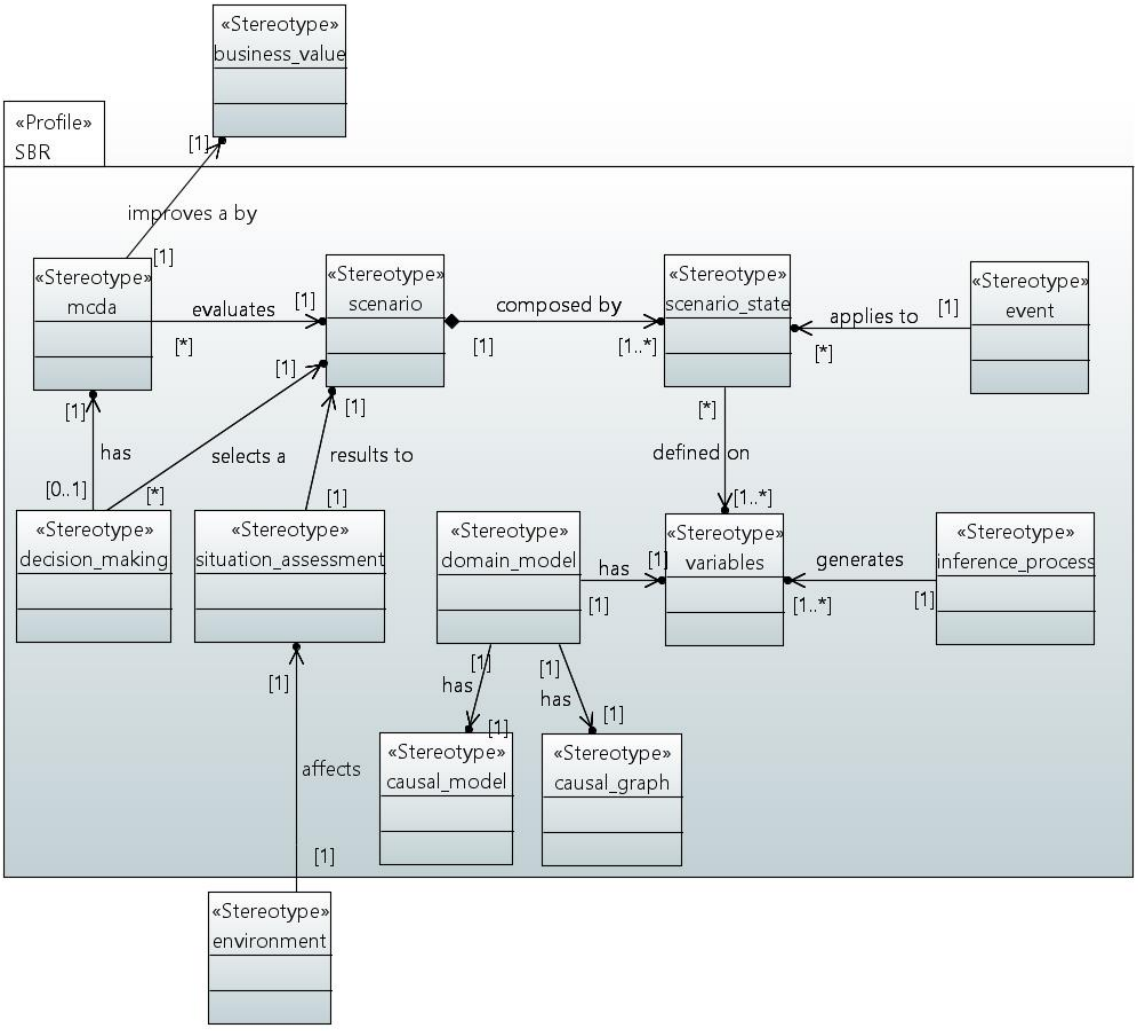

Figure 22 - SoS Scenario-Based Reasoning (SBR) package

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/continuity of operation, 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 takes inspiration from the profile already defined into the CHESS Project [57]. The latter define Stereotypes in order to describe security concepts of an SoS. The main aim of CHESS Project is defining, developing and assessing a methodology for the component-based design and development of embedded systems, using model-driven engineering techniques [57]. Among the activities performed for the CHESS project there is the definition of a methodology that aims at properly supporting the representation of non-functional properties like dependability aspects.

4.3.3.1 DEPENDABILITY COMPONENTS

In order to describe dependability concepts we propose to select the main components defined in CHESS profile and to integrate them in our SoS profile.

Figure 23 shows the dependability components followed by a detailed description of its entailed elements.

In our profile we define the "*SoS Dependability*" package. We select the main concepts defined in CHESS profile that we consider useful for a system designer and for the AMADEOS objective. Indeed this profile has several additional concepts regard to the viewpoints described in Section 2, however we consider they could be very useful during the system analysis and design.

We consider a constituent system as an element that could fail. In our profile failure classifications are associated with component ports. In a Block Definition Diagram, a block can have a flow port normally used to describe interaction points for items flowing in or out of a block. Through this port we specify which are the failures that we assume can occur on that port. Ports connected together by a connector should use the same classification with respect to failures. We have introduced the concept of system state that could be nominal state, error state ("**errorState**") or failure state ("**failureState**"). When a system is in a failure state its behavior is deviated from the intended behavior and during this interval of time there is system outage ("**outageState**"). The transition from a system failure to intended system behavior represents the system restoration ("**restorationState**"). We will define several error states, which define a behavior of the component different from the nominal healthy state (or initial state).

In the following we list the set of concepts we adopted for the Dependability conceptual model as represented in Figure 23:

- **ElementWithFailures**: this concept extends the Constituent System stereotype and BDD Port concept, by specifying the failure classification to be used for that specific port.
- **FailureType**: represents a type of failure (i.e., a failure mode) that is explicitly considered for the involved system. Different failure types may have different consequences on failure propagation.
- **FailuresClassification**: defines a partition of the failure modes associated with a component or service of the system architecture. Different failure classifications can be used for different parts/components of the system, in order to analyze in details their specificities.
- **ErrorModel**: this element extends the SysML StateMachine element, with the purpose of allowing users to describe detailed error models for system components and connectors.
- **SystemState**: this element represents the state of the involved system and extends the "state" concept defined in a State Machine Diagram. It is described within an error model.
- **NominalState**: this element represents the system nominal state.

- **ErrorState**: this element represents state of the component with respect to failures. ErrorState can only appear in ErrorModel StateMachines.
- **FailureState**:this represents the state of failure
- **OutageState**: this elements represents the transition state during which the system has failed.
- **RestorationState**:it represents the transition state from system failure to intended system behavior.
- **ThreatEvent**: this abstract element represents an event occurring within a component or connector, and modifying its behavior with respect to failures. It is described within an error model.
- **Fault**: this element represents a fault; a fault is active when it causes an error, otherwise is dormant.
- **Technique**: this element represents a technique used to reduce the faults occurrence.
- **FaultPrevention**: this element extends technique element and represents a particular technique aiming to prevent the introduction of faults during the development phase of the system.
- **FaultTolerance**: this element extends technique element and represents a particular technique aiming to avoid the occurrence of failures by performing error detection and system recovery over time.
- **faultRemoval:** this element extends technique element and 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.
- **FaultForecast:** this element extends technique element and represents a particular technique aiming to forecast faults by performing an evaluation of system behavior.
- **InternalFault**: this element represents an internal fault occurring within the component to which the error model is associated.
- **ExternalFaultReaction**: this element represents a reaction of the component to an external fault occurring on one of its ports. The reaction is modeled by a change of "**errorState**" within the "**errorModel**".
- **RecoveryEvent**: this element represents a recover event, for which the component changes its "**errorState**" (within the "**errorModel**") to a "**nominalState**". The recover requires a certain amount of time to be performed that could be provided by the a Recover Delay attribute.
- **HardwareFault**: this element represents a failure of the hardware on which a software component is allocated. The purpose of this element is to be able to refer, within the error model, a failure of the hardware.
- **Measure**: this element represents a property expected from a dependable system strictly connected with the failure type.
- **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.

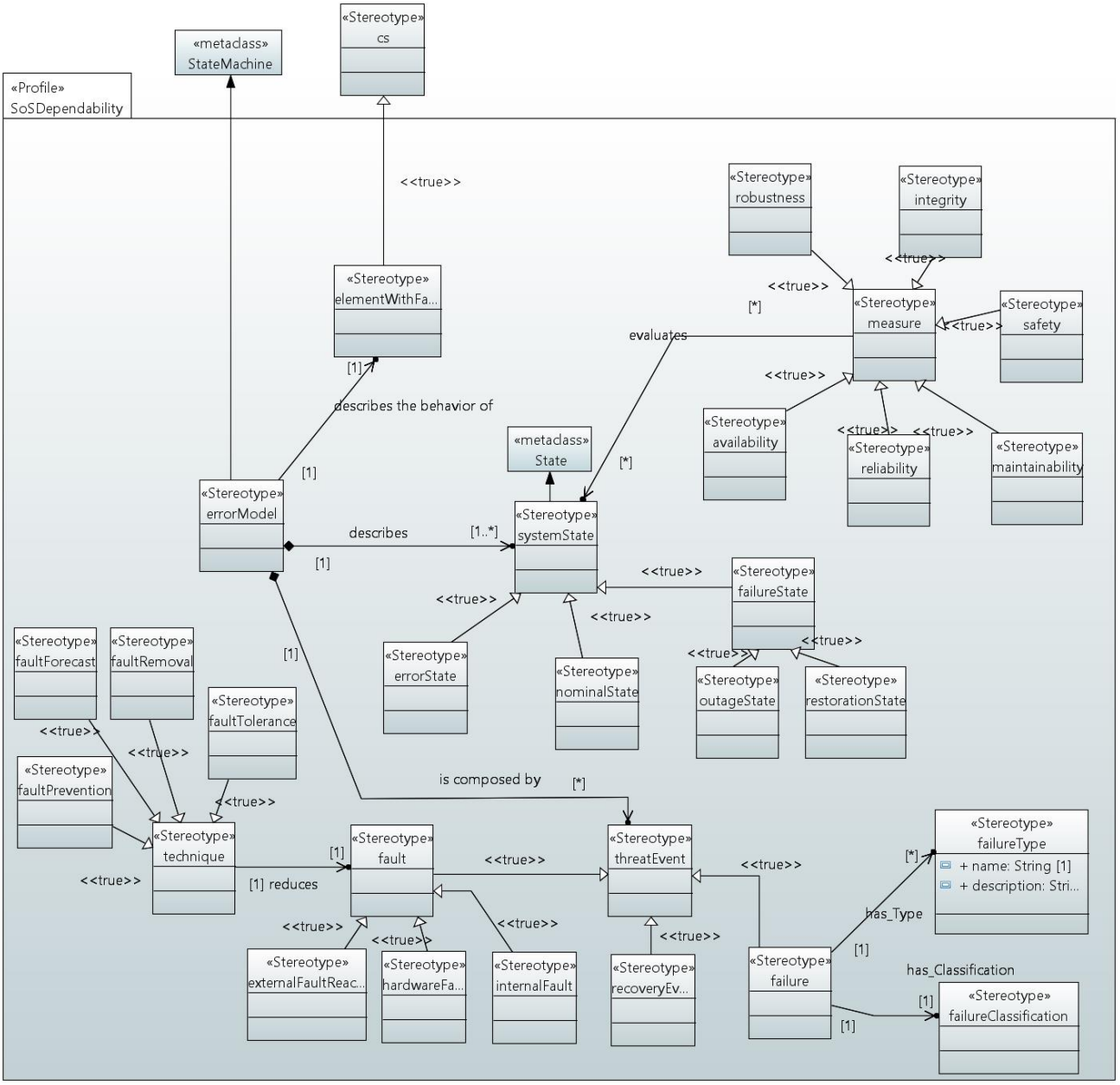

Figure 23. Dependability conceptual model.

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 24, we connect the Stereotype "**sos**" to "**security**" Stereotype, in order to satisfy the condition of security of an SoS. To this end we use "**cryptography**" based on symmetric ("**symmetric_cryptography**") or public key ("**public_key_cryptography**") infrastructure.

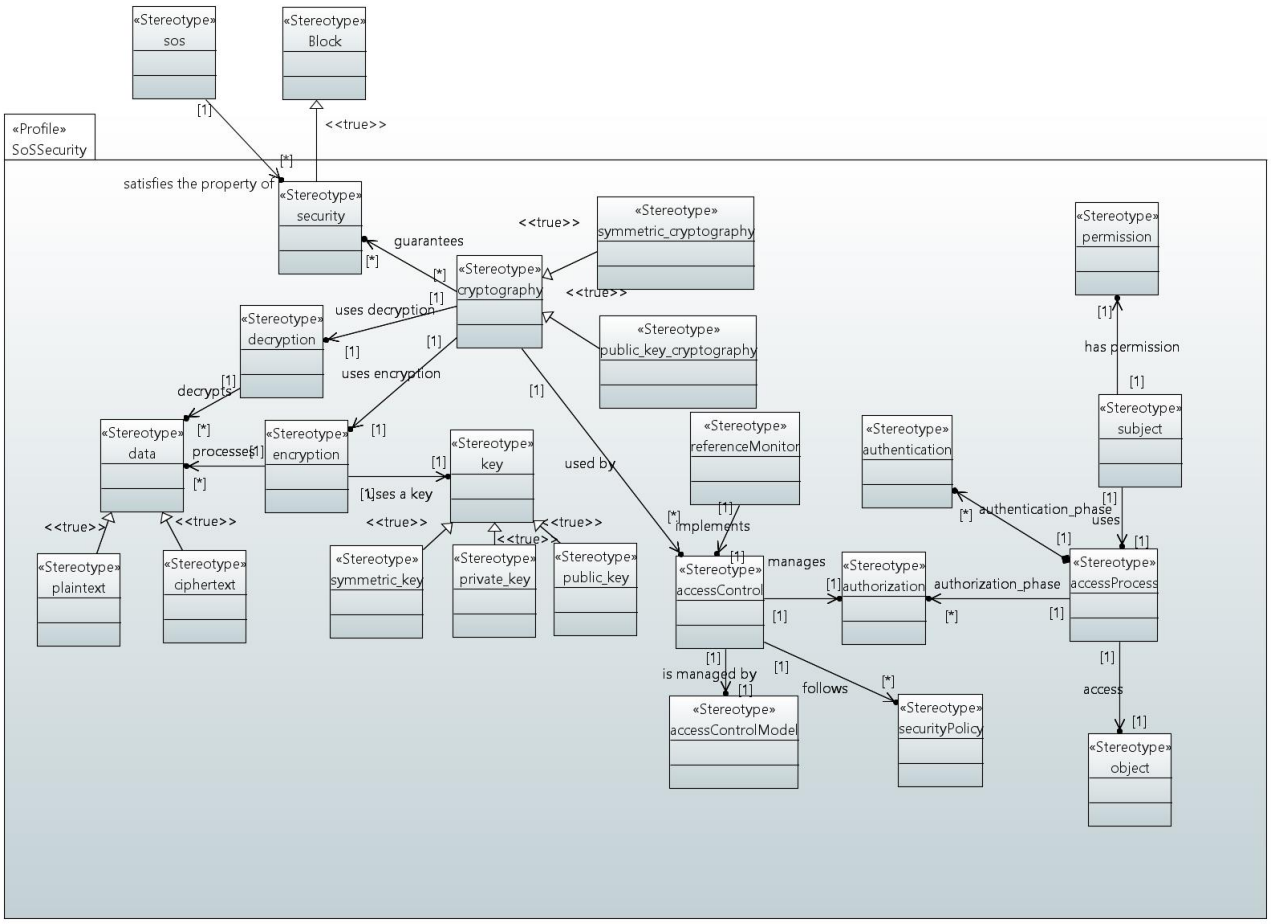

Figure 24: SoS Security package

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**").

The information exchanged (also called "**data**") can be either "**ciphertext**" if encrypted "**plaintext**" otherwise and "**decryption**" Stereotype represents the process of turning ciphertext to plaintext.

During the cryptography phase the access control ("**accessControl**") consists in a set of action in order to define the actions that are permitted or not allowed by the system. Figure 24 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 could have attributes ("**permission**") that describe how the subject can access to objects. An "**object**" is a passive system-related devices, files, records tables, processor, or domain containing or receiving information. 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 as a policy within a system. 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 25 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 extend the metaclass "Lifeline" of a Sequence Diagram. The "timeline" is composed by an infinitive 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.

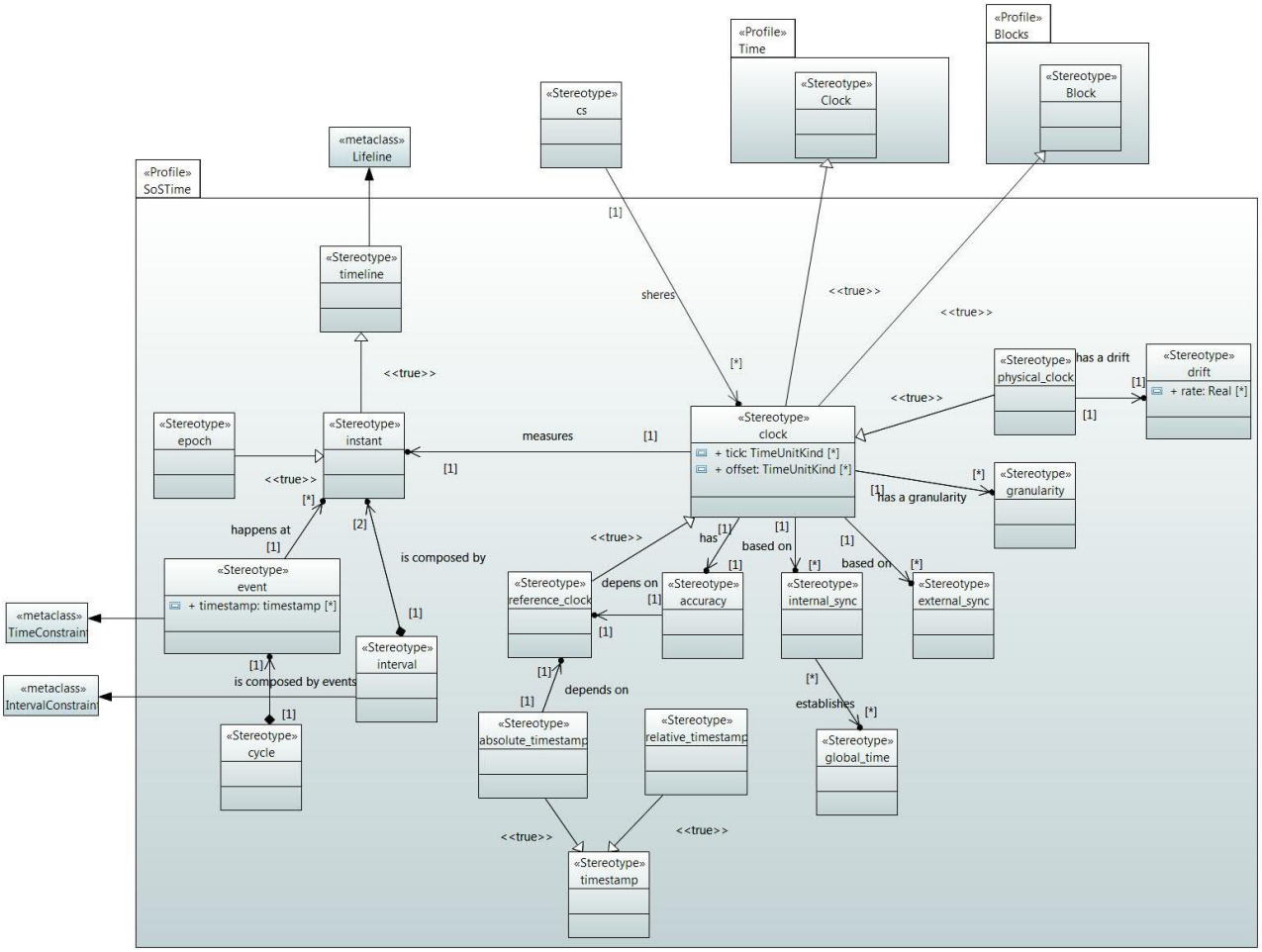

Figure 25: SoS Time package

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 26). 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".

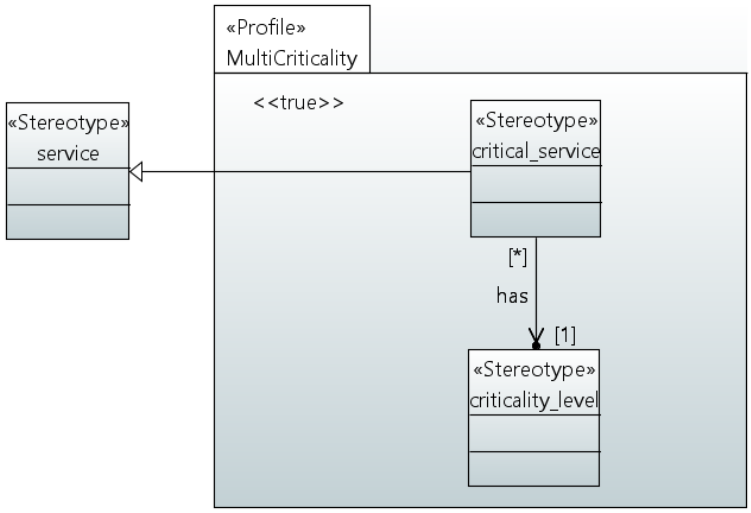

Figure 26: Multi-criticality package

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 27 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 weak ("**weak_emerg_phenomenon**") or strong ("**strong_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"

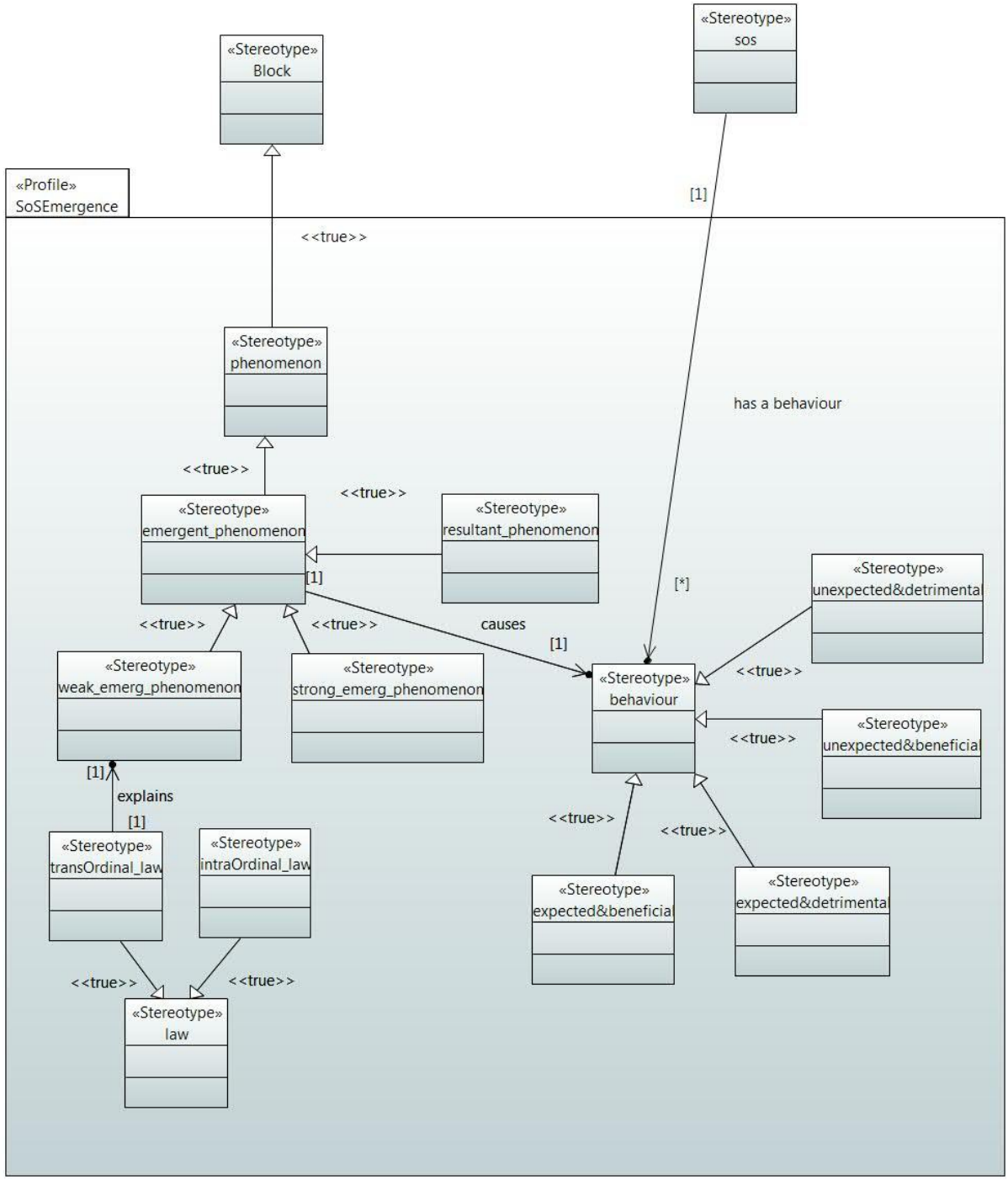

Figure 27: SoS Emergence package

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.

In order to show the two different representations 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 pick

of request of energy which comes from the neighborhood. 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 have it if we only look at the interactions of the internal household CSs without considering the neighborhood CSs.

Figure 28 shows how, through a BDD, the public event lighting is represented as a weak 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.

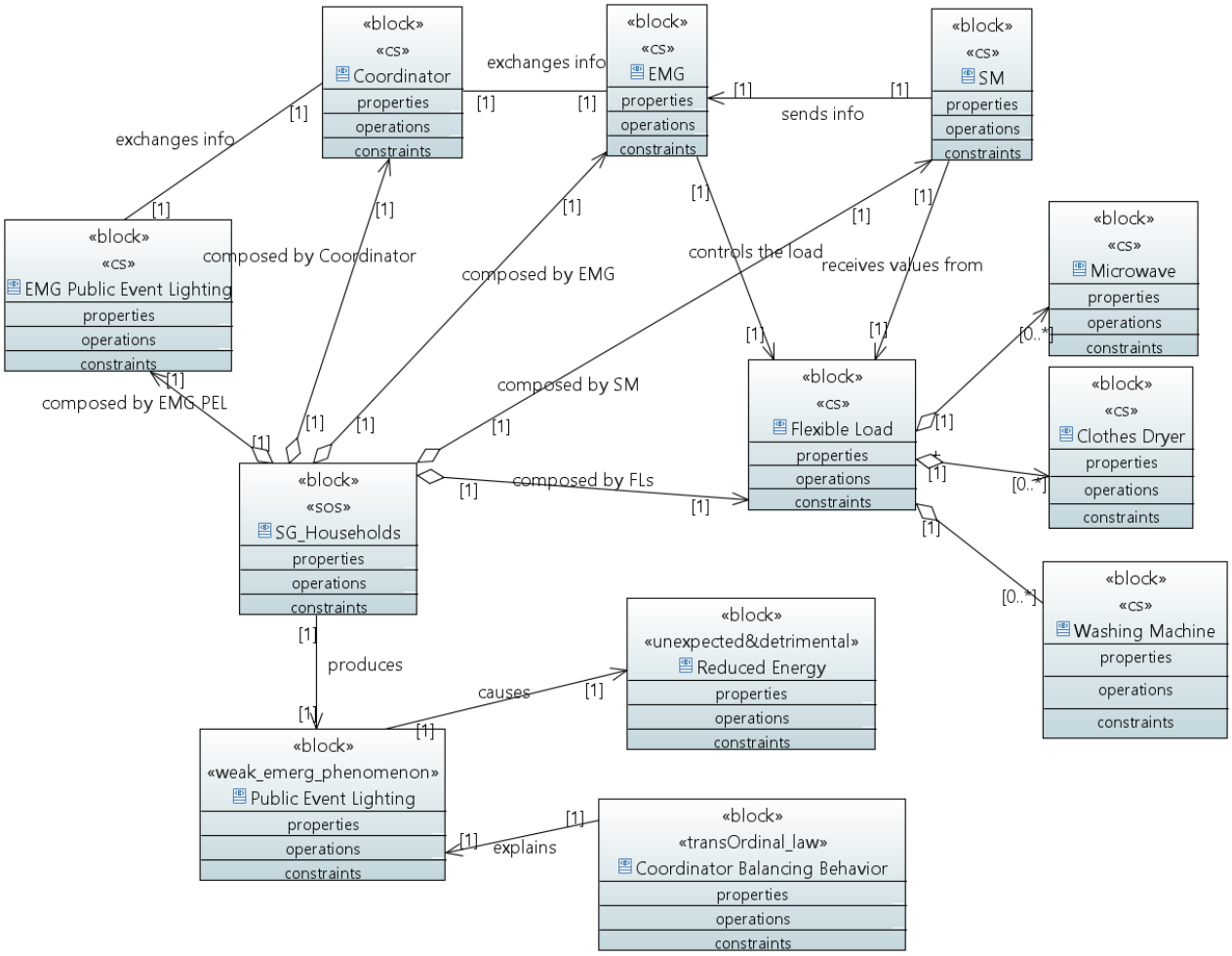

Figure 28: Smart Grid Household – Emergent behavior

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 pick 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.

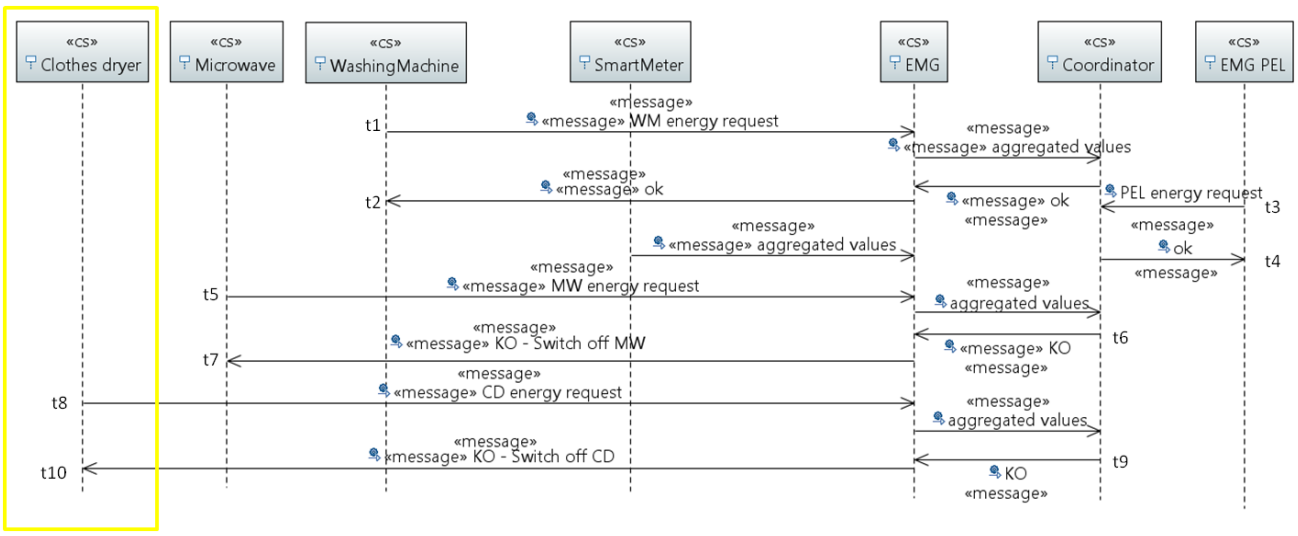

Figure 29: Smart Grid Household SysML Model – Emergent Behavior description

As shown in Figure 29, “electronic appliances” CSs are represented as “Lifeline” and their interactions are represented through directed labeled arrows. As shown in Figure 29, 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 from the Public Event Lighting (PEL) EMG, which is external to the household. At time t5, microwave issues its request to be connected at 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 pick 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.

Annex A shows a simple example related to how our profile can become useful to analyze the SoS behavior and find hazardous behaviors. In particular we focus on how modeling systems helps a system designer to execute an interface hazard analysis activity. This represents only an example of application of the SoS profile but it already shows its usefulness for our Smart Grid scenario.

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 behaviour 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 behaviour) within the SoS.

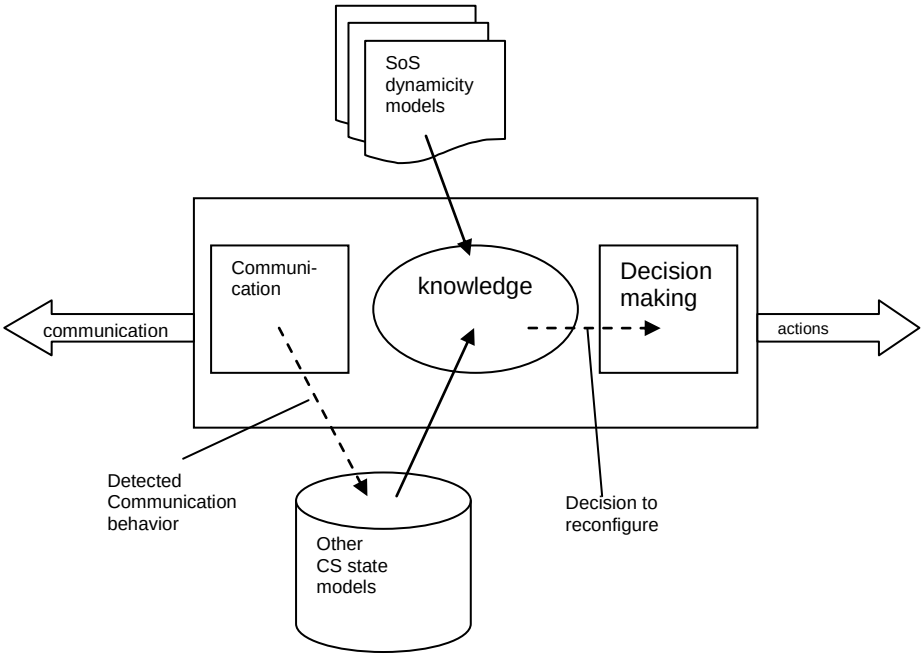

Figure 30: Internal representation of dynamicity viewpoint in CS agent model

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.

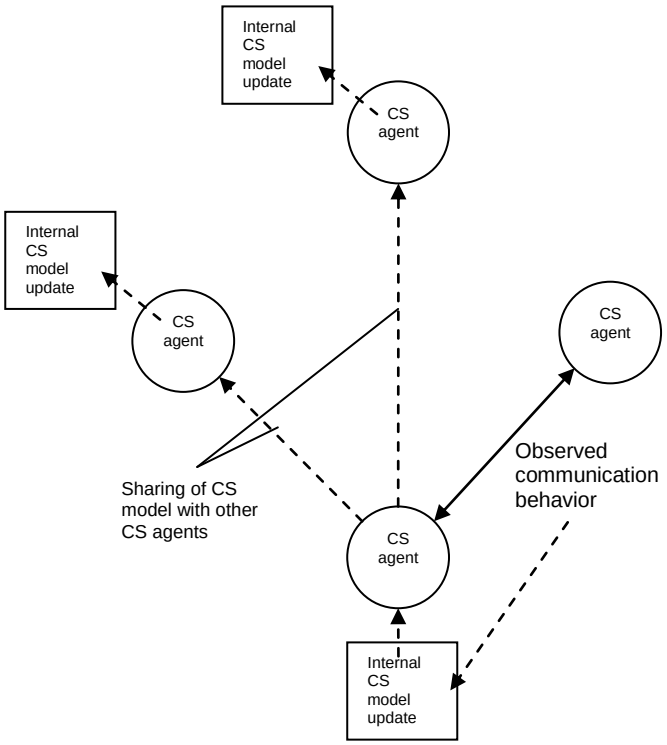

Figure 31: Sharing of Dynamicity viewpoint information between agents representing CSs

4.5.2 Viewpoint of Emergence in multi-agent based SoS domain

This viewpoint addresses the emergence of global SoS behaviour 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 behaviour.

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 behaviour.

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.

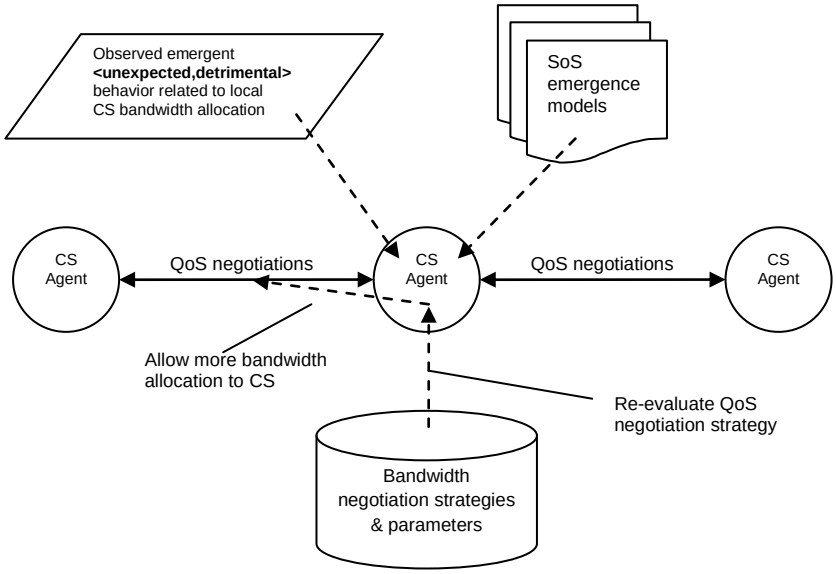

Figure 32: Applying emergence models and observations to adapt local QoS negotiation behavior

5 CONCLUSION

The document discussed the rationale for conceptual modeling System-of-Systems (SoSs) starting from the achievements of deliverable D2.1, which contains preliminary basic SoS concepts and relationships. A revision to the starting concepts has been carried out in order to align the conceptual modeling to the parallel activities in the others WPs and to focus on core AMADEOS issues. Comments from reviewers have been also considered for the revision of SoS basic concepts.

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.* A detailed description on interfaces has been included in order to determine interactions within an SoS.

The document showed how the basics SoS concepts and relationships have been exploited to define an SoS profile which is aligned with mainstream languages in SoS engineering. In particular, the document described a SysML SoS profile defined in different components related to the 7 views of analysis. Applicability of the profile has been proved by illustrative SoS instances and by further enabled analysis over the application of the SoS profile.

The conceptual model in this document builds a common basis for the remaining activities of AMADEOS, which we expect to reciprocally evolve according to the conceptual model. This process is expected to be concluded by the end of the project with the delivery of the final version of the AMADEOS conceptual model (deliverable D2.3 at month 36).

6 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

ANNEX A - INTERFACE ANALYSIS TO DETECT EMERGENT BEHAVIORS OF SOSS

This section describes an example of how the SoS profile can be used in order to detect emergent behaviors of an SoS by means of an interface analysis.

As already described in Section 2, SoSs are often subject to emergence behaviors belonging to a different nature:

- Expected or Unexpected emergent behavior
- Beneficial or Detrimental emergent behavior

In this section, through an example, we clarify how we can detect, avoid and reduce a detrimental and emergent behavior starting from an instantiation of our proposed SoS profile. The idea consists in analyzing the interfaces of CSs as they can be formalized through our SoS architectural component.

In the following, we provide a sequence of activities to be performed in order to carry out the SoS interface analysis, which achieves a preliminary assessment (identification/prediction) of possible emergent behaviors. These steps (summarized in Figure 33) are:

Step 1. Defining an SoS architectural model using the elements of the SoS profile detailed in previous sections

Step 2. Highlighting the types of connections each constituent system may have with other constituent systems and univocally Identifying all the SoS internal interfaces

Step 3. Identifying a set of events that could lead to emergent behaviors, be it beneficial or detrimental

Step 4. Mapping each event with some guidewords in order to explore particular circumstances that lead the system to an emergent behavior.

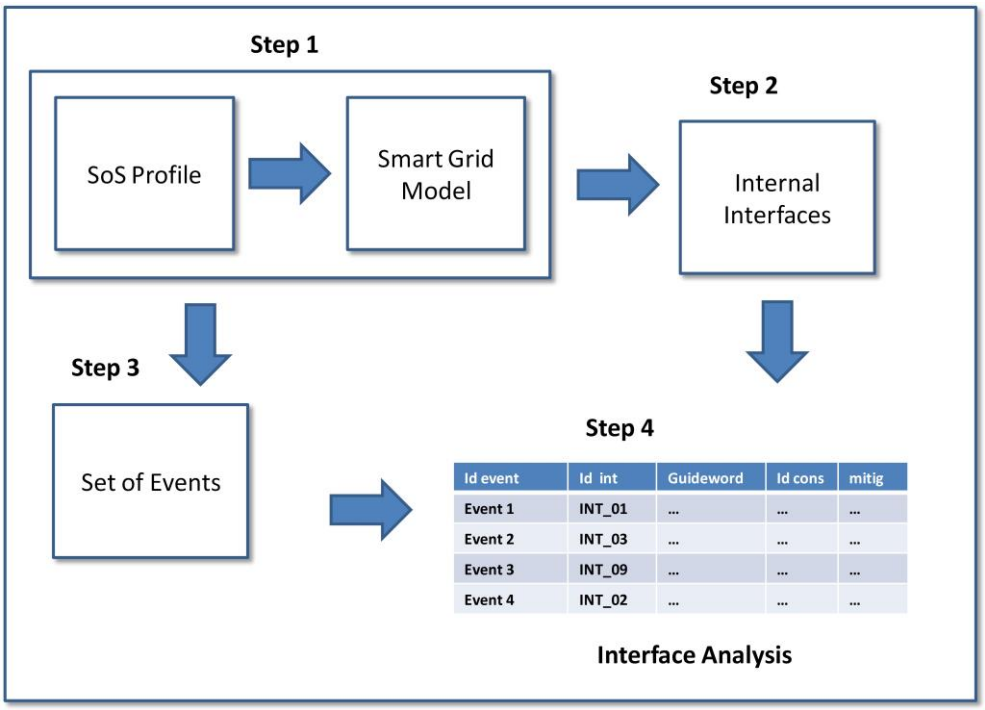

Figure 33: Example of SoS profile application

To illustrate how it is possible to detect emergent behaviors, we exploit the Smart Grid household example as defined in Section 4.2 . According to Step 1 we define the SoS model for the Smart Grid. Figure 13 shows how the SoS profile has been exploited to model the Smart Grid architecture. Using this representation we have identified all the internal interfaces as listed in Table 5 (Step 2)

Interface among Constituent Systems	ID Interface
EMG and Coordinator	INT_01
Coordinator and DSO	INT_02
Smart meter and Meter Aggregator	INT_03
Meter Aggregator and DSO	INT_04
Command Display and EMG	INT_05
Command Display and Smart meter	INT_06
Smart Meter and Flexible load	INT_07
Smart Meter and EMG	INT_08
EMG and Flexible load	INT_09
PEL EMG and Coordinator	INT_10

Table 5: Internal Interfaces of a Smart Grid Household scenario

Following Step 3 we have identified a small set of events that could originate emergent behaviors, as follows:

Event 1. modifications of constituent systems (change of a physic aspects or system requirements)
Event 2. introduction of new components that interact with the constituent systems
Event 3. constituent systems removal

This represents the most difficult step because it is strictly related to the SoS characteristics. In order to identify a set of events that conduct the system to emergent behaviors, it is possible to leverage previous experiences and common practices related to the applicative domain. For instance, for our Smart Grid example, we have basically identified introduction/modification and removal of components.

Once we have identified relevant events, it is possible to start with a system interface analysis (Step 4). To this end, we exploit the interface hazard analysis technique, as defined in [61], which identifies and mitigate hazards leading to detrimental situations. Our aim is to adopt the hazard analysis technique with a different objective, i.e., finding emergent conditions (positive and negative) related to the information exchanged through the interfaces.

Our hazard-based analysis takes as input event types (Id Event) and internal interfaces (Id Interface) (see Table 5) and it produces as output the identification of consequences (Consequences) and emergent behaviors (Emergent Behavior). An extract of this analysis is shown in Table 6. Each rows represents an event with an associated guideword (in our example *not*, *more* and *early/late*) [63] which adds semantic information to the event, i.e., "not" indicates the non-occurrence of an event, "more" indicates the occurring or something additional, "early/late" means an early/late occurrence of an event. In canonical hazard analyses, guidewords are exploited to help designers to detect hazardous events of the system. In our specific case, we exploit guidewords to identify both hazards and emergence behaviors.

Let us consider the following events:

- Event 1: A new functionality is added to the command display: the electrical appliances can be switched on and off through the command display HMI. When an electrical appliance is switched on or off, the command display sends an information message to EMG containing the name/type of an electrical appliance involved. In this case EMG can distinguish the device that requires/release energy and forward the request/notification to the Coordinator.
- Event 2: A new EMG is connected to the Smart Grid to support the provision of energy for a public event lighting.

Id Event	Id Interface	Guideword	Hazard	Emergent Behavior	Consequence	Mitigation
Event 1	INT_05	Not	EMG does not receive information from the new Command Display	NO	EMG cannot forward the request/notification to the Coordinator	MIT_01: Command Display has to wait the acknowledgment from EMG
Event 1	INT_05	Not/More	EMG receives wrong information from the new Command Display	NO	EMG makes a mistake in requesting/releasing energy	MIT_02: EMG has to verify the information sent from Command Display
Event 1	INT_05	Late	EMG receives information from the new Command Display with a delay	NO	EMG cannot control flexible load and does not optimize the energy consumption	MIT_01: Command Display has to wait the acknowledgment from EMG within a fixed interval
Event 1	INT_05	More	EMG receives additional information from the new Command Display on the electrical appliance switched on	YES - Beneficial	EMG can forward additional information to the Coordinator for better balancing the Smart Grid	
Event 2	INT_10	More	The Coordinator receives a request for a very high amount of energy to support the public event lighting	YES - Detrimental	Coordinator, in order to keep balanced production and consumption values, decides to limit the provision of energy for the electrical appliance	MIT_03: EMG can communicate the energy decrease to the common Display (INT_05). The latter supports the reconfiguration of the electrical appliances

Table 6: Example of Smart Grid Interface Analysis

Table 6 shows a little extract of an Interface Analysis with the purpose of detect emergent behaviors. The last two lines identify two types of emergent behaviors. The former represent a beneficial emergent behavior caused by the new functionality of the command display: EMG receives additional information on the electrical appliance willing to be switched on. In this case EMG can forward additional information to the Coordinator. For instance the type of electrical appliance could be communicated to the Coordinator, which can optimize the energy consumption accordingly (i.e., the energy requested by a washing machine lasts longer than the energy requested by a microwave).

The latter represents a detrimental emergent behavior caused by the request of a high amount of energy to support the public event lighting. This information is received by the Coordinator from the exceptionally connected EMG. In this case commonly granted electrical appliance cannot be switched on. Mitigation to this case consists in making the user aware of this situation by communicating the limited energy provision to the household through the common display. The user may accordingly re-schedules its request of energy, e.g., may decide to turn off for a few minutes the currently working washing machine in order to warm his launch with the microwave (given that both electrical appliances cannot be connected simultaneously).

ANNEX B - 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.

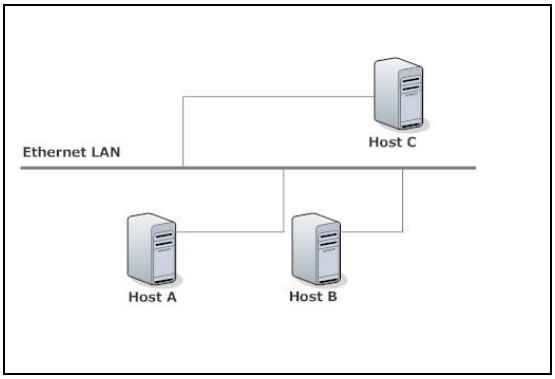

Figure 34: Ethernet capture effect

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.

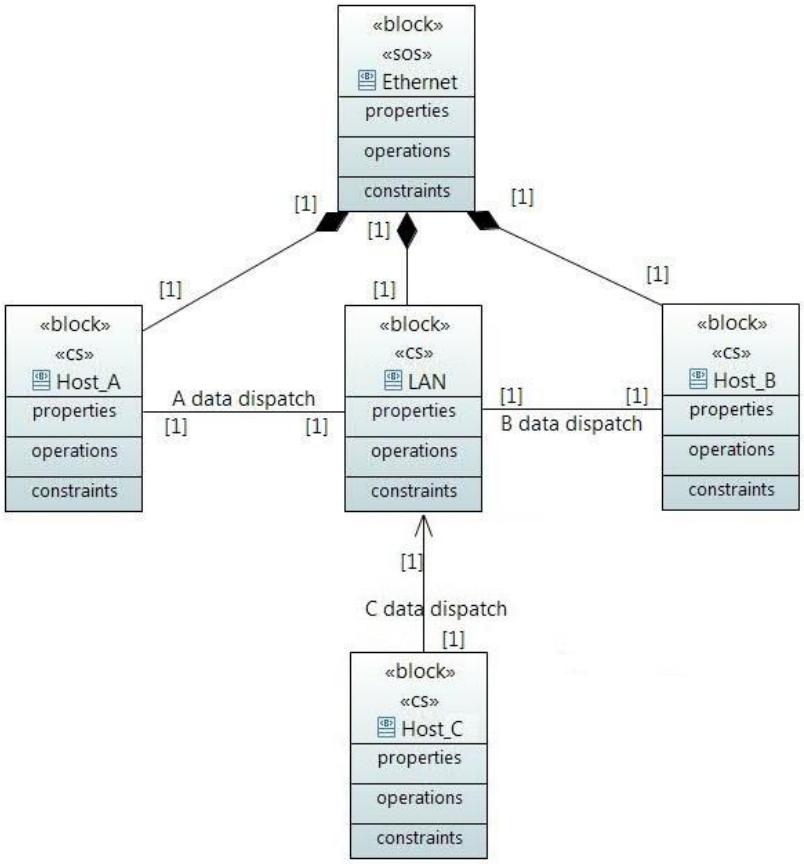

Figure 35: Ethernet Capture Effect SysML Model – Topology description

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 a weak emergent phenomenon explained by the random backoff computation. This phenomenon causes an unexpected and detrimental behavior of the SoS.

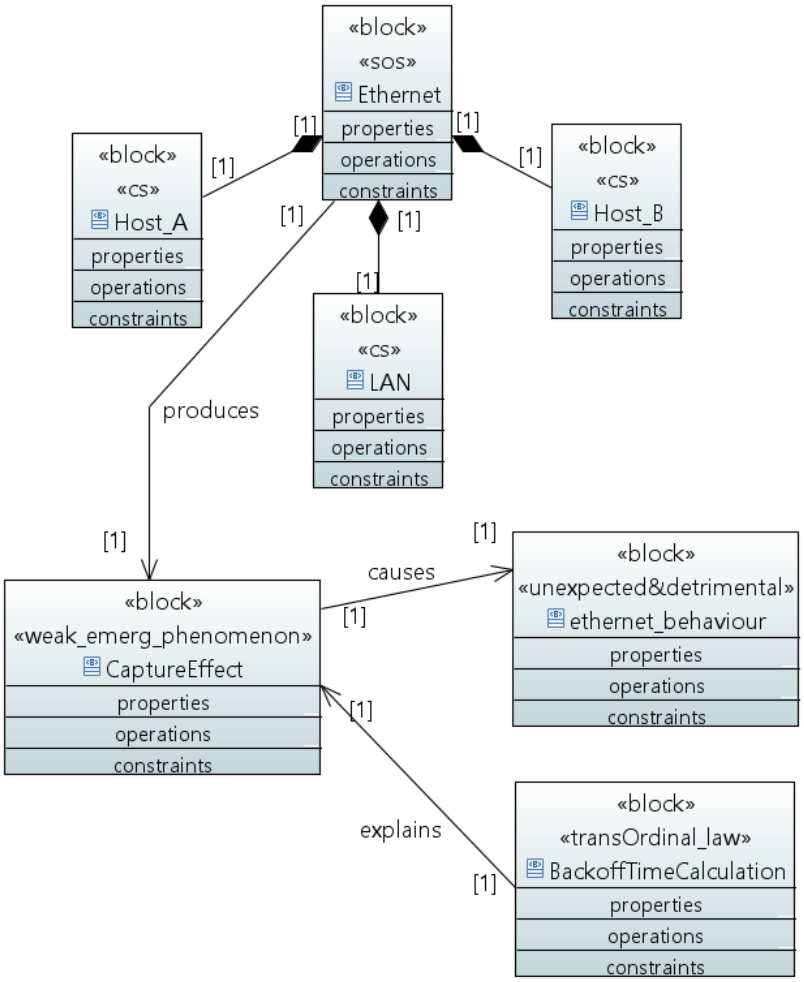

Figure 36: Ethernet Capture Effect – Emergent behavior

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.

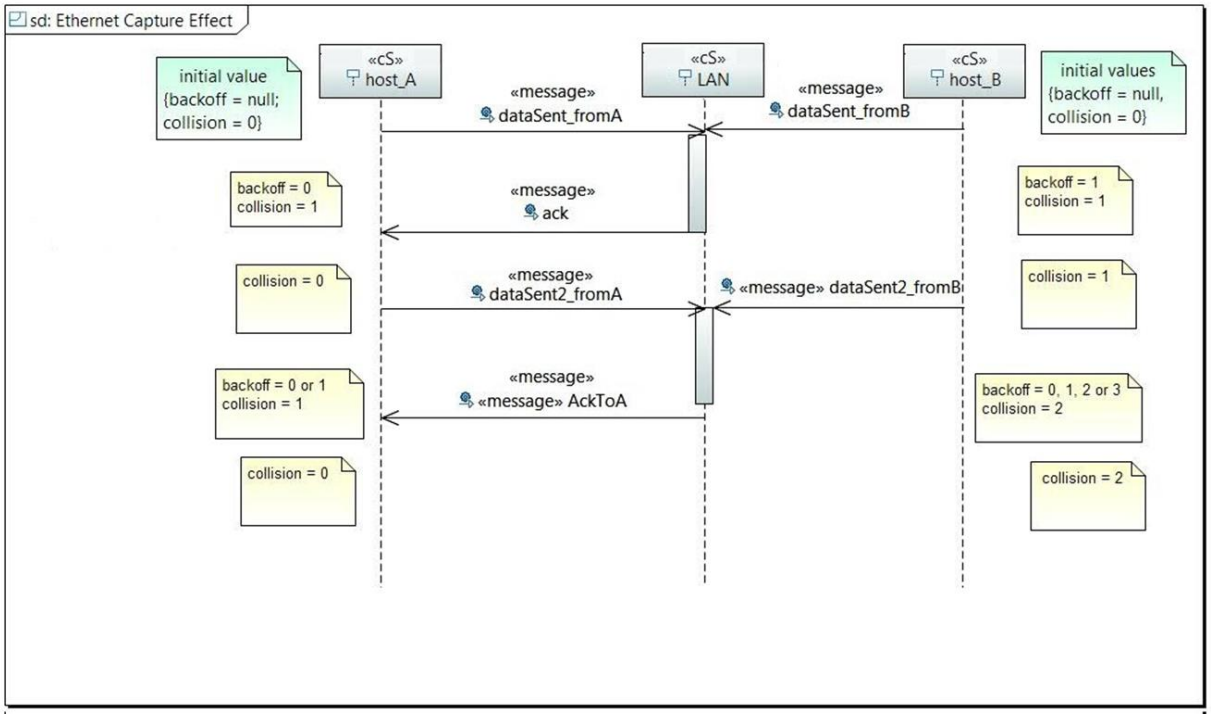

Figure 37: Ethernet Capture Effect SysML Model – Emergent Behavior description

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 7 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 7 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 7 - 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

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Open System	SoSArchitecture	Enumeration	sys_type- open	Block Definition Diagram (SysML)	Enumeration literal of 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)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Periodic System	SoSArchitecture	Enumeration	sys_type - periodc	Block Definition Diagram (SysML)	
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

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Receive Instant	SoSCommunication	Property	receive_instant	Sequence Diagram (UML2)	Property of trailer
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)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Weak emergence	SoSEmergence	Stereotype - Block	weak_emerg_phenomenon	Block Definition Diagram (SysML)	
Strong Emergence	SoSEmergence	Stereotype - Block	strong_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)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Public key	SoSSecurity	Stereotype - Block	public_key	Block Definition Diagram (SysML)	
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)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
Scenario pruning	SoSSBR	Stereotype - Block	scenario	Block Definition Diagram (SysML)	
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 Physical Specification	SoSInterface	Stereotype - Block	p-spec	Block Definition Diagram (SysML)	
Interface Message Specification	SoSInterface	Stereotype - Block	m-spec	Block Definition Diagram (SysML)	
Interface Service Specification	SoSInterface	Stereotype - Block	s-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)	
Temporal Properties	SoSInterface	property	property – temporal_prop	Block Definition Diagram (SysML)	

Set of Main Concepts	Profile Package Name	SysML metadata representation Type	Profile Element Name	Diagram Type	Description / Notes
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)	
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 - DEFINITION DIFFERENCES WRT D2.1

This annex lists the definitions or sections that have been modified or added compared to deliverable D2.1.

Location	Concepts	Rationale
2.1.1	Attribute, Value, Property, Static Property, Dynamic Property, Observation an Entity	Alignment with concepts in literature.
2.1.2	Constituent System	Result of discussion of AMADEOS consortium at plenary meeting.
2.1.2	Reducible System	Concept contrasts emergence.
2.1.2	Evolutionary System	SoSes investigated in D1.1 are evolutionary and motivate the introduction of this concept.
2.1.3	Human-Machine Interface (HMI) Component	Human-System interaction justifies the introduction of this interface.
2.2.1	Time	Explicit definition of the formerly implicit concept.
2.2.1	Now	Explicit definition of the formerly implicit concept.
2.2.1	Signal	Connecting concept: time and information
2.2.1	Time code	Alignment with ITU [49].
2.2.1	Time scale	Alignment with ITU [49].
2.2.1	Offset of events	Explicit definition of the formerly implicit concept.
2.2.1	Phase shift	Alignment with ITU [49].
2.2.1	Phase jump	Alignment with ITU [49].
2.2.2	Syntonization	Alignment with ITU [49].
2.2.2	Nominal Frequency	Alignment with ITU [49].
2.2.2	Frequency drift	Alignment with ITU [49].
2.2.2	Frequency offset	Alignment with ITU [49].
2.2.2	Stability	Alignment with ITU [49].
2.2.2	Wander	Alignment with ITU [49].
2.2.2	Jitter	Alignment with ITU [49].
2.2.2	Reference clock	Refinement of the concept.
2.2.2	Coordinated clock	Alignment with ITU [49].
2.2.2	Path delay	Explicit definition of the formerly implicit concept.
2.2.2	Clock Ensemble	Alignment with ITU [49].
2.2.2	Primary clock	Explicit definition of the formerly implicit concept.
2.3.2	State Space	Refinement of the concept.
2.3.2	Instantaneous State Space	Refinement of the concept ‘state space’ w.r.t. time.
2.4.2	Capability	Explicit definition of the formerly implicit concept.
2.5.4	Stigmergy, Environmental Dynamics, Stigmergic Information Flow	Added concepts to address review comments concerning physical interactions in SoSes.
2.8.4	Criticality level, Criticality, Multi-criticality system, Critical service	Added concepts to address review comments concerning AMADEOS core topics.