Basic Concepts on Systems of Systems

Andrea Ceccarelli[1(✉)], Andrea Bondavalli[1], Bernhard Froemel[2], Oliver Hoeftberger[2], and Hermann Kopetz[2]

[1] Department of Mathematics and Informatics, University of Florence, Florence, Italy
{andrea.ceccarelli,andrea.bondavalli}@unifi.it
[2] Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria
{froemel,oliver}@vmars.tuwien.ac.at, h.kopetz@gmail.com

1 Introduction

A System of System (SoS) stems from the integration of existing systems (legacy systems), normally operated by different organizations, and new systems that have been designed to take advantage of this integration. 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. This is well represented by Table 1.

In this chapter we present the fundamental concepts for Systems of Systems engineering established within the AMADEOS[1] project, with the objective of proposing a shared System of Systems *vocabulary* and define an *implicit theory* about the SoS domain.

The overarching concern of our work is to target the reduction of the *cognitive complexity* needed to comprehend the behaviour of a SoS by the application of appropriate *simplification strategies* [29]. In fact, the considerable cognitive effort needed to understand the operation of a large SoS is the main cause for the substantial engineering (and monetary) effort required to design and maintain many of today's Systems of Systems.

Our position is that the first important step of achieving simplicity is the development of an understandable set of concepts that describes the SoS domain. In fact, if the language used to talk about a domain contains terms with clearly defined meanings that are shared by a wide part of the domain community, then the communication of ideas among experts is simplified. Consequently, starting from a detailed analysis of the existing concepts for the SoS domain (e.g., from the projects DANSE [35], DSoS [7],

This work has been partially supported by the FP7-610535-AMADEOS project.

[1] FP7-ICT-2013-10-610535 AMADEOS: Architecture for Multi-criticality Agile Dependable Evolutionary Open System-of-Systems, http://amadeos-project.eu/.

A. Bondavalli et al. (Eds.): Cyber-Physical Systems of Systems, LNCS 10099, pp. 1–39, 2016.
DOI: 10.1007/978-3-319-47590-5_1

Table 1. Comparison of an SoS compared to a monolithic system [22]

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	???

COMPASS [36]), in this work we reuse main existing concepts, and formulate new ones when needed, to propose a *shared vocabulary for Systems of Systems* that can be used in a coherent and consistent manner.

The concepts devised are divided in ten viewpoints, summarized below. Noteworthy, an extended version of the conceptual model is freely available at [33].

Fundamental System Concepts. Discussed in Sect. 2, the first group focuses on the static structure of systems and their parts. It starts from the presentation of the universe and time of discourse of an SoS, to finally define an SoS and its related parts.

Time. Discussed in Sect. 3, this group explains the progression of time and its role in an SoS. The role of time and clocks in SoSes is further debated in Chap. 6.

Data and state. Discussed in Sect. 4, this groups defines the data and information that are exchanged between the Constituent Systems that form an SoS. These concepts are further investigated in Chap. 2.

Actions and Behaviour. Discussed in Sect. 5, this group illustrates the dynamics of an SoS, that consists of discrete variables by an *event-based view* or by a *state-based view*.

Communications. Discussed in Sect. 6, the focus of this group is on the role and properties of a communication system in an SoS. These concepts are further elaborated in Chap. 2.

Interfaces. Discussed in Sect. 7, this group presents the fundamentals definition for the interfaces i.e., the points of interaction of SoS components with each other and with the environment over time. These concepts are further debated in Chap. 2.

Evolution and Dynamicity. Discussed in Sect. 8, this group explains SoS dynamicity, intended as short term changes, and evolution, intended as long term changes. These concepts are also largely applied in Chap. 7.

System design and tool. Discussed in Sect. 9, this group sets the foundational concepts to define design methodologies to engineer SoSes.

Dependability and Security. Discussed in Sect. 10, this group presents dependability and security concepts, in compliance with the taxonomies presented in [1, 3, 4].

Emergence. The phenomenon of emergence in Cyber-Physical Systems of Systems and its main concepts are largely discussed in Chap. 3. Consequently, this group of concepts is only briefly introduced in this Chapter. The definition of emergence reported in Chap. 3 states that "a phenomenon of a whole at the macro-level is emergent if and only if it is of a new kind with respect to the non-relational phenomena of any of its proper parts at the micro level". Emergent phenomena can be of a different nature either beneficial or detrimental and either expected or unexpected. Managing emergence is essential to avoid undesired, possibly unexpected situations generated from CSs interactions and to realize desired emergent phenomena being usually the higher goal of an SoS [30]. For example, system safety has been acknowledged as an emerging property [31], because its meaning at the SoS level does not have the same meaning for the individual CS, and obviously it cannot be expressed just as the sum of the individual parts. Additionally, to further strengthen on the relevance of emergence in an SoS, we remark that it is acknowledged that the SoS may be exposed to new security threats when novel phenomena arise [32, 34].

2 Fundamental System Concepts

2.1 Universe and Time of Discourse

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* [8] 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, such as an idea.

2.2 Systems

We use the definition of the term *system* introduced in the EU Project DSOS (Dependable System-of-systems IST-1999-11585 [9]):

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 [9]. 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, 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 [22, 29].

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

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

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

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

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

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 [11], p. 184.*

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

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

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

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*.

Some 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 [25].

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

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

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

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

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

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

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

2.3 System-of-Systems

We decided to select the following definition of Jamishidi as the starting point of our work [10].

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.

Dahmann and Baldwin have introduced the following four categories of SoSs [11]:

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" with respect to the SoS,* since the service of the SoS as a whole relies on the services provided by the respective CSs across the RUIs.

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

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

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

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

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

There may be other interfaces to systems external to a CS which we investigate in Sect. 6, together with the issues of *information exchange* and *interface specification*.

3 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* [12] 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.

3.1 Basics on Time

Time: A continuous measurable 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.

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.

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

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 [2], 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 [14], pp. 19–4. Note that the difference between *cycle* and *period* is the constant duration of the period during the IoD.

3.2 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.

Nominal Frequency: The desired frequency of an oscillator [24].

Frequency drift: A systematic undesired change in frequency of an oscillator over time [24].

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 [24].

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) [24].

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) [24].

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 [24].

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.

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 [24].

Clock synchronization establishes a *global notion of time* in a clock ensemble. A global notion of time is required in an SoS if the timestamps generated in one CS must be interpreted in another CS. *Global time* is an abstraction of physical time in a distributed computer system. It is approximated by a properly selected subset *of the ticks* of each synchronized local clock of an ensemble. A *selected tick* of a local *clock* is called a *tick* of the *global time*. For more information on the global notion of time, see [2], pp. 58–64.

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

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

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

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

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

There are a number of different internal synchronization algorithms, both non-fault tolerant or fault-tolerant, published in the literature (see e.g., [13], and many others). These algorithms require the cooperation of all involved clocks.

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

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

The term *master clock* is often used synonymously to the term *primary clock.* If the clocks of an ensemble are externally synchronized, they are also internally synchronized with a precision of $|2A|$, where A is the *accuracy*.

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

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

3.3 Time in an SoS

In a recent report from the GAO to the US Congress [15] it is noted that *a global notion of time is required in nearly all infrastructure SoSs,* such as telecommunication, transportation, energy, etc. 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 [14]. 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. 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 [16], 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 1 μsec, a holdover of up to one hour is achievable using a good GPSDO.

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

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

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

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

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

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

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

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

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

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

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

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

4 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 [16]. In this Section we will elaborate on the concepts of *data* and *information* along the lines of reasoning expressed in [17].

4.1 Data and Information

Let us start by defining the *fundamental* concepts of *data* and *information* [17]:

Data: A data item is an artefact, 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 [8]. 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 [18]. 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* [19]), 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* [17].

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*.

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 [20], 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.

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 [21].

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.

4.2 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.

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.

5 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 the trace. Events that happen within a duration that is shorter than the duration between the equidistant observation instants may get lost.

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

Most control systems use *sampling* to acquire information about the controlled object. The choice of the *duration between two observation instants*, called *the sampling interval,* is critical for acquiring a satisfying image of the controlled object. This issue is discussed extensively in the literature about control engineering [23].

5.1 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 depends on the performance of the available hardware and is also 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.

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°* 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.

Real-Time (RT) Transaction: A time-bounded transaction.

In a control system the duration of the RT-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.

5.2 Behaviour

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

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

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.

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.

6 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 data from a sender to one or more receivers. The higher-level protocols build on these basic transport protocols to provide more sophisticated services.

6.1 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.

6.2 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 where the send instant is derived from the progression of the global time.

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 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.

Table 2 reports on the main characteristics of the transport services surveyed above. 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.

Table 2. Characteristics of 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

6.3 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.

6.4 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 [26].

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 [27, 28]. 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.

A good example for a stigmergic information flow is the exchange of information among the drivers of cars on a busy intersection.

7 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.

Interface Properties: The valued attributes associated with an interface.

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

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

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

Note that abstraction is always associated with a given specified purpose.
Interface properties can be characterized at different interface layers:

- **Cyber-Physical Layer:** At the cyber-physical layer information is represented as data items (e.g., a bit-pattern in cyberspace, or properties of things/energy in the physical world) that are transferred among interacting systems during the IoD.
- **Itom Layer:** In this layer we are concerned with the timely exchange of Itoms by unidirectional channels across CS interfaces.
- **Service Layer:** At the service layer, the interface exposes the system behavior structured as *capabilities*. In contrast to the informational layer, Itom channels are not individually described at the service layer, but only the interdependencies between the exchanged Itoms are specified.

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 interface layers the interface properties, i.e., what type of, how, and for what purpose information is exchanged at that interface.

7.1 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).

External Interface: A Constituent System (CS) is embedded in the physical environment by its external interfaces.

We distinguish three types of external CS interfaces: Time-Synchronization Interface (TSI), Relied Upon Interfaces (RUIs) and *utility interfaces*. RUIs have been defined in Sect. 2.3.

Time-Synchronization Interface (TSI): The TSI enables external time-synchronization to establish a global timebase for time-aware CPSoSs.

Utility Interface: An interface of a CS that is used for the configuration, the control, or the observation of the behaviour of the CS.

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. 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 support 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 or other CSs that is not accessible over any other external interface.

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.

One important class of faults that might occur at connected interfaces is related to compatibility.

Property Mismatch: A disagreement among connected interfaces in one or more of their interface properties.

Connection System/Gateway Component/Wrapper: A new system with at least two interfaces that is introduced between interfaces of the connected component systems in order to resolve property mismatches among these systems (which will typically be legacy systems), to coordinate multicast communication, and/or to introduce emerging services.

8 Evolution and Dynamicity

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 Sect. 2.3).

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

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

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

Evolutionary Step: An evolutionary change of limited scope.

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

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

Managed SoS evolution is due to changes in the environment. The goal of managed evolution in AMADEOS is maximizing business value, while maintaining high SoS agility:

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

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

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

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

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

System effectiveness: The system's 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 of a system: The capability of a system to react promptly to changes in the environment.

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

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

We conclude the discussion presenting fundamentals on governance, because governance-related facts may have impact on SoS evolution.

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.

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

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

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

9 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.

9.1 Architecture

The architecture of a system can have some variants or even can vary during its operation. We recognize this adaptability of a system architecture by the following three 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.

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

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

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

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

There are several methodologies to design a system.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Then we have two methodologies to perform design verification:

Design inspection: Examination of the design and determination of its conformity with specific requirements.

Design walkthrough: Quality practice where the design is validated through peer review.

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

10 Dependability and Security

We report 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" [5] and Bruce Schneier's "Applied Cryptography" [6].

10.1 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.

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 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.

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

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

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

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.

10.3 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" [6], have been adapted to incorporate our definitions of *data* and *information*.

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. Data that has been encrypted is called ciphertext.

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.

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

11 Conclusions

The need of understanding and explaining the SoS in a clear way is a relevant requirement in order to reduce the cognitive complexity in SoS engineering. The scope of the set of concepts presented in this Chapter is to provide a common basis for the understanding and the description of Systems of Systems; we hope that such SoS concepts contribute towards such needed clarification.

Finally, we conclude mentioning that *SoS engineering* is closely related to many other IT domains by looking at the *glue* that is needed to integrate the diverse systems, for example *Cyber-Physical Systems (CPSs)*, *embedded systems, Internet of Things (IoT), Big Data.* The concepts introduced in this Chapter and in the AMADEOS project to describe the properties of an SoS can thus form a foundation of a conceptual model in these other domains as well.

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 edn. Springer, NewYork (2011)
3. Laprie J.: Resilience for the scalability of dependability. In: Proceedings ISNCA 2005, pp. 5–6 (2005)
4. Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Comput. **1**(1), 11–33 (2004)
5. Anderson, R.: Security Engineering, 2nd edn. Wiley, New York (2008)
6. Schneier, B.: Applied Cryptography, 2nd edn. Wiley, New York (1996)
7. Jones, C., et al.: Final version of the DSoS conceptual model. DSoS Project (IST-1999-11585) (2002)
8. Popper, K.: Three Worlds: The Tanner Lecture on Human Values. University of Michigan (1978)

9. Simon, H.: The Science of the Artificial. MIT Press, Cambridge (1969)
10. Jamshidi, M.: Systems of Systems Engineering—Innovations for the 21st Century. Wiley, Cambridge (2009)
11. Dahman, J.S., Baldwin, K.J.: Understanding the current state of US defense systems of systems and the implications for systems engineering. In: Proceedings of 2nd Annual IEEE Systems Conference, Montreal. IEEE Press (2008)
12. Withrow, G.J.: The Natural Philosophy of Time. Oxford Science Publications, New York (1990)
13. Kopetz, H., Ochsenreiter, W.: Clock synchronization in distributed real-time systems. IEEE Trans. Comput. **36**(8), 933–940 (1987)
14. Lombardi, M.A.: The use of GPS disciplines oscillators as primary frequency standards for calibration and metrology laboratories. Measure J. Measur. Sci. **3**, 56–65 (2008)
15. 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)
16. Zins, C.: Conceptual approaches for defining data, information and knowledge. J. Am. Soc. Inf. Sci. Technol. **58**(4), 479–493 (2007)
17. Kopetz, H.A.: Conceptual model for the information transfer in systems of systems. In: Proceedings of ISORC 2014, Reno, Nevada. IEEE Press (2014)
18. Floridi, L.: Is semantic information meaningful data? Philos. Phenomenological Res. **60**(2), 351–370 (2005)
19. Glanzberg, M.: Truth: Stanford Encyclopedia on Philosophy (2013)
20. World Wide Web Consortium: Extensible Markup Language. http://www.w3.org/XML/. Accesssed 18 April 2013
21. Aviation Safety Network: Accident Description. http://aviationsafety.net/database/record.php?id=19920120-0 Accessed 10 June 2013
22. Maier, M.W.: Architecting principles for systems-of-systems. Syst. Eng. **1**(4), 267–284 (1998)
23. Zak, S.H.: Systems and Control. Oxford University Press, New York (2003)
24. International Telecommunication Union (ITU): Glossary and Definition of Time and Frequency Terms. Recommendation ITU-R TF686-3, December 2013
25. Frei, R., Di Marzo Serugendo, G.: Concepts in complexity engineering. Int. J. Bio-Inspired Comput. **3**(2), 123–139 (2011)
26. Kopetz, H.: Direct versus stigmeric information flow in systems-of-systems. TU Wien, November 2014 (not yet published)
27. Camazine, S., et al.: Self-Organization in Biological Systems. Princeton University Press, Princeton (2001)
28. Grasse, P.P.: La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La theorie de la stigmergie. Insectes Soc. **6**, 41–83 (1959)
29. Kopetz, H.: Real-Time Systems, 2nd edn. Springer, New York (2011)
30. Mogul, J.C.: Emergent (mis)behavior vs complex software systems. ACM SIGOPS Oper. Syst. Rev. **40**(4), 293–304 (2006)
31. Black, J., Koopman, P.: System safety as an emergent property in composite systems. In: 2009 IEEE/IFIP International Conference on Dependable Systems & Networks. IEEE (2009)
32. Mori, M., Ceccarelli, A., Zoppi, T., Bondavalli, A.: On the impact of emergent properties on SoS security. In: 7th International Conference on System of Systems Engineering (SoSE) (2016)
33. AMADEOS Consortium: D2.3 – AMADEOS Conceptual Model Revised (2016). http://amadeos-project.eu/documents/public-deliverables/

34. Gligor, V.: Security of emergent properties in ad-hoc networks (transcript of discussion). In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 2004. LNCS, vol. 3957, pp. 256–266. Springer, Heidelberg (2006). doi:10.1007/11861386_30
35. DANSE Consortium: DANSE Methodology V2-D_4.3. https://www.danse-ip.eu
36. COMPASS, Guidelines for Architectural Modelling of SoS. Technical Note Number: D21.5a Version: 1.0, September 2014. http://www.compass-research.eu