Interfaces in Evolving Cyber-Physical Systems-of-Systems

Bernhard Frömel(✉) and Hermann Kopetz

Institute of Computer Engineering,
Vienna University of Technology, Vienna, Austria
froemel@vmars.tuwien.ac.at, h.kopetz@gmail.com

1 Introduction

In the past twenty years the view on how we engineer, operate and evolve independently owned and managed *Cyber-Physical Systems (CPSs)* in order to realize and optimize complex economical processes has started to change. Advances in telecommunications and automation accompanied by standardization efforts resulted in sophisticated cross-domain information and communication technologies (e.g., the Internet of Things (IoT) [2, 9], elastic processing and storage clouds, Web Services) that allow for the integration of more and more existing and previously technologically isolated CPSs. These *legacy systems* became cooperating *Constituent Systems (CSs)* of evolving Cyber-Physical Systems-of-Systems (CPSoSs) and – by their physical and cyber interaction – give rise to new emergent services that cannot be realized by any single or small number of CSs alone.

One prototypical example of a CPSoS is a smart grid [14] where the interacting CPSs (producers, consumers, and prosumers where, for example, electricity consuming households are equipped with electricity producing photovoltaic power plants) cooperate to optimize energy distribution with respect to stability, dependability, and costs. A smart grid handles high *dynamicity* as it constantly reconfigures in order to react to changed energy production and demand conditions. Further they need to support *evolution* during runtime as the *service* of the smart grid is adapted or extended towards new *requirements* or technological advances. Finally, smart grids represent critical infrastructure that may in the event of failure cost human lives or cause high economical costs. Hence a smart grid needs to fulfill high expectations concerning its *dependability*, including *security* and *safety*.

Central to the integration of CPSs as CSs of evolving CPSoSs are their *interfaces*, i.e., their points of interaction with each other (direct interaction) and with their common *environment* (indirect interaction) over *time*. The identification, proper specification, standardization, and managed modification of these interfaces are of paramount importance in order to tackle CPSoS key challenges related to emergence, dynamicity, evolution and dependability. Specifically, time-sensitive physical interactions and the role of delays in *emergence* impose the requirement of properly taking

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

A. Bondavalli et al. (Eds.): Cyber-Physical Systems of Systems, LNCS 10099, pp. 40–72, 2016.
DOI: 10.1007/978-3-319-47590-5_2

time for all kinds of interactions in CPSoSs into account. To this end this work assumes the availability of a *sparse global timebase* [25, 26] that can be used by all involved CSs to temporally coordinate *interactions* at their interfaces. We call an SoS where its CSs have access to such a global timebase a *time-aware SoS*.

The objective of this chapter is twofold: First, we conceptualize time-sensitive interactions in CPSoSs at appropriate *interface layers* and propose a CS interface design that simplifies engineering, operating and evolving such CPSoSs. Second, we discuss evolution of CPSoSs and how to manage it by applying our proposed interface design.

The following section gives a brief overview of related work. Section 3 conceptualizes interfaces in CPSoSs and introduces the *Relied Upon Interface (RUI)* of a CS which is an interface the operational service of the overall CPSoS relies upon. Section 4 discusses the design of RUIs. Section 5 suggests how evolution can be managed at the RUI. Section 6 concludes this chapter.

2 Related Work

In the domain of safety-critical real-time systems several simplification principles concerning the integration of models of distributed computer systems with models of physical processes have been suggested [25]: abstraction, separation of concerns, causality by determinism, temporal segmentation, independence of *entities*, observability, and consistent time. In accordance with these simplification strategies the design of linking interfaces [24] which realize cyber-interactions among nodes of a distributed real-time system, and the design of *sensor/actuator* interfaces [10] that interact with the physical process of a Cyber-Physical System (CPS) has been proposed.

Maier outlines in [30] fundamental differences in developing monolithic systems compared to a System-of-Systems (SoS) where for example the single Constituent Systems (CSs) are operationally and managerially independent, the SoS has an evolutionary nature, and there are emergent behaviors. Maier postulates that SoS architecting might rely entirely on interface design and the specification of communication standards at multiple abstraction levels.

The World Wide Web (WWW) running on top of the Internet is often considered as one of the first examples of a human engineered SoS, also satisfying Maier's definition of SoSs. Fielding [15] suggests the concept of an *architectural style*, i.e., in his thesis the *"named, coordinated set of architectural constraints"*, and uses it to obtain an appropriate architectural design for networked software components. Fielding further introduces the Representational State Transfer (REST) architectural style which he applied in the definition of the Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifier (URI) specifications. Together they describe the generic interface which is in its essential form still used in all interactions of the WWW.

Web Services (WSs) [3, 8] are a prominent web-inspired implementation of the Service-oriented-Architecture (SoA). They are based on machine-interpretable interface descriptions (Web Service Description Language (WSDL), and other WS-* specifications) and give support for platform-independent machine-to-machine interaction over large-scale networks like the Internet. Highly distributed applications can

be integrated by means of Web Service (WS) that are provided and owned by possibly many different entities. However, WSs are subject to the limitations of underlying technologies and the expressiveness of the used description languages. For example, the end-to-end *delay* of *messages* is in the scope of underlying technologies. Currently, temporal and semantic interoperability is not part of the *interface specification*, hence – while an agent might syntactically be able to interact with a WS – there might be a temporal and/or semantic mismatch leading to undesired effects.

OPC Unified Architecture (OPC UA) [29] is a SoA machine-to-machine communication stack intended to tackle challenges related to semantic interoperability. The OPC UA specifications are maintained by the Open Platform Communications (OPC) Foundation and have been in part standardized in IEC 62541. The extensible information model allows the description of arbitrary object structures where *object data* and its *meta data* is managed.

Caffall and Michael propose in [7] the concept of service-oriented contract interfaces for an architectural framework for SoSs. Contract interfaces are based on the principles of Design-by-Contract (DbC) and demand an explicit definition of interfaces among CSs that formalize assumptions about the services provided by a CS to achieve goals of the SoS.

3 Interfaces in Cyber-Physical Systems-of-Systems

This section introduces interfaces in the architectural context of time-aware Cyber-Physical Systems-of-Systems (CPSoSs), discusses interface abstraction layers, and presents different interface classes of Constituent Systems (CSs) that are part of a CPSoS.

3.1 Architectural Elements of Cyber-Physical Systems-of-Systems

A CPSoS is a System-of-Systems (SoS) whose interacting Constituent Systems (CSs) are Cyber-Physical Systems (CPSs). The architecture of a CPSoS defines the boundaries of its elements (major building blocks), the relationships among them, and the relationship between elements and their environment at abstraction levels that are useful for the discussion of CPSoS *attributes* of interest. There are many important attributes (e.g., business, societal, legal) to investigate in CPSoSs, but this chapter focuses on behavioral attributes, i.e., on interaction relations among architectural elements of CPSoSs. The architectural elements in CPSoSs are: the *Itom* [27] as the unit of interaction, the CS as a computing component that interacts physically and digitally with its environment, and the environment of CSs that enables the interactions among CSs. An Itom is an information atom comprising of data and explanation in an atomic unit. The environment of a CS consists of all entities that have the capability to interact with the CS.

CSs process Itoms which they exchange with their environment at their interfaces. There are two kinds of interactions a CS can have with its environment: message-based

interactions with *cyber space* and physical or *stigmergic* interactions [28]. In order to model stigmergic interactions it is useful to define the concept of an *entourage* of a CS, i.e. all entities that are part of a CS, but are external of its computer system. It consists of humans and *things* and may change over time. The entourage of two or more CPSs may – not necessarily simultaneously – overlap during the *Interval of Discourse (IoD)*. Overlapping entourages provide a common environment which enables stigmergic interactions among involved CPSs.

Itom
An Itom [27] is the basic *information* item exchanged in interactions of CSs. It is defined as "*a timed proposition about some state or behavior in the world"* in some representation (data) together with the explanation of this representation. For processing in computer systems the representation and explanation can be both coded as bit strings, i.e., digital data. The representation part is called object data, while the explanation of the object data is called meta data. Depending on the Itom's purpose, the meta data might be based on an ontology (e.g., a conceptual model of Newtonian physics), or can be a machine program for a (virtual) computer system which explains the object data (cf. code mobility [17]). Interacting CSs may adhere to different *contexts*, i.e., often have implicit assumptions about the actual meaning and temporal *properties* of exchanged object data. Conceptually, Itoms solve this context dependency problem by tying information representation and explanation together. Hence object data is interpreted consistently across all CSs that need to access and process the information contained in the Itom.

The definition of Itoms is recursive and allows that an Itom contains other Itoms. Take for example the object- and meta data of two or more Itoms and regard them as object data of a higher level Itom. The explanation of this higher-level Itom describes how to extract the contained Itoms. Consequently, Itoms can be – in principle – arbitrarily complex constructs that might describe (virtual) computer machines and thus (virtual) CSs.

To avoid misinterpretation of interactions the explicit definition of Itoms as the basic unit of interaction is essential in modeling CPSoSs. Explicitly defined Itoms also enable automatic Itom *transformation systems* which are able to map object data in compliance to one explanation into object data conforming to another explanation and vice-versa, provided the two meta data explanations can be put formally into relation, i.e., a bijective function exists that maps one explanation (described by meta data) to the other.

Constituent System
A Constituent System (CS) is a Cyber-Physical System (CPS) which consists of a computer system (cyber part), optionally an influenced and/or observed object (physical part) and possibly humans. The object, i.e., a thing obeying to physical laws in the dense physical time of our reality, can be observed by sensors and/or influenced by actuators of the cyber part (for example for the purpose to control it). The *behavior* of the computer system and its ability to conduct *actions* in the physical environment adheres to a discrete progression of time. Consequently, a sufficient alignment of the dense physical time and

the discrete computer time is essential for an influence or *observation* of the physical part of a CPS that is precise enough for the application at hand.

The purpose of a CS regarding its integration in a *collaborative SoS* is the realization of a service that allows the CS to benefit from the emergent CPSoS service. The service of a CS is only provided at its interface thus an interface specification sufficiently defines the CS's interaction *capabilities* (all possible interactions) that the CS internals must deliver. In CPSoSs a precise temporal coordination of the individual CSs is required across the whole CPSoS. Hence, CSs have access to a sparse global timebase [25], and are able to *timestamp input actions* (messages, sensor observations) and timely generate *output actions* (messages, actuations) at the CS interface within the *precision* of the global timebase.

The computer system of a CS processes Itoms that are received at the CS interface by sensors observing a property of the physical *state* in the entourage of the CS, or are received by messages. Further a CS generates new Itoms that can be implemented as influences on the physical state in the entourage by actuators, or sent as messages into cyber-space, possibly to other CSs.

In summary, a CS implements a time-aware computational element that operates on Itoms which it exchanges with its environment according to its interface specification.

Environment

A CS interacts with two kinds of environments at its interface: cyber space which allows message-based communication and the physical environment which enables physical interactions among CSs.

Cyber Space

Cyber space is a distributed information processing system that enables message-based interactions among CSs by means of direct and indirect cyber *channels*. For instance, the IP-based Internet is a prominent example of a planet-scale cyber space where in principle any two systems with a unique IP address can establish cyber channels and exchange Itoms. The concept of (physical) proximity and neighborhood of CSs does not necessarily play a significant role in cyber space. The physical distance between two CSs exchanging messages via cyber space might only affect the delay of the message, but has no other impact on the communication (e.g., does not change message contents in the absence of *faults*).

A direct cyber channel conveys messages from one sending CS to one or more receiving CS without any modifications.

An indirect cyber channel is established over the state of a shared memory which is located somewhere in cyber space. The sending CSs modify the shared memory by means of state messages. The shared memory is also possibly affected by cyber dynamics a form of *environmental dynamics*. Cyber dynamics are autonomous processes inherent within the cyber space and/or cyber interactions of other not explicitly modeled systems. Finally, receiving CSs obtain the (partial) state of the shared memory. For example, the publish-subscribe [13] communication paradigm is supported by indirect cyber channels where cyber-dynamics (that are in this example part of the system architecture) take care about queueing published messages and notifying subscribers. Many popular

message-based middleware platforms support publish-subscribe, e.g., the Robot Operating System (ROS) [34], or Eclipse paho[1], based on MQTT [5].

Physical Environment

The physical environment consists of things and physical fields (energy) whose properties we model as a dynamic network of physical state variables. Such a network of *state variables* can be described in an *environmental model* (for example, see [21] about an ontology-based environmental model) which captures the interrelationships (e.g., transfer delays, functional dependencies, location) of the state variables. For example, the temperature and the pressure of air are physically related and if one is affected by an actuation, so is the other one. Our networked view on the relations of physical state variables allows for a simple composition of environmental models, the consideration of different levels of detail, and taking interfacing effects into account. Note that one can reduce this network to a single set of physical state variables where all relations among the variables need to be described explicitly.

In the physical environment the concept of proximity is essential, because many physical interactions depend on distance (e.g., force fields). In case the entourages of two or more CSs overlap during the IoD, a stigmergic information flow, i.e., a physical interaction, can take place. Overlapping entourages help to limit the size of the environmental model that needs to be considered for the interaction of CSs. Figure 1 shows the network of physical state variables which is under the influence of *environmental dynamics*, but also part of the entourage of multiple CSs. Environmental dynamics are the time-sensitive effects of autonomous processes occurring in the environment.

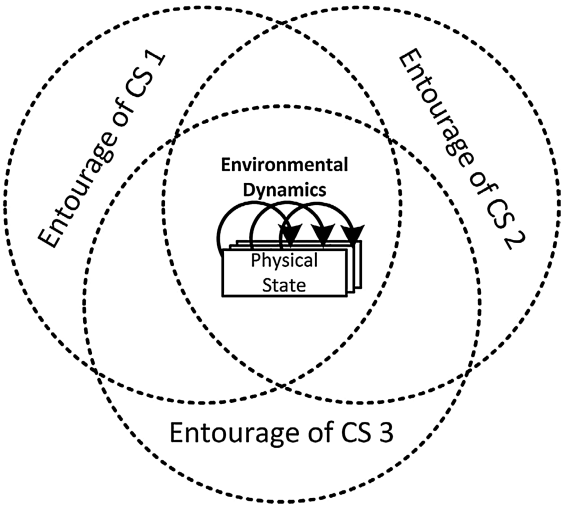

Fig. 1. Overlapping entourage of CPSs enabling physical interaction

[1] https://eclipse.org/paho/.

An actuator is an interface device of a CS which allows the CS to apply changes to one or more state variables of the physical environment that is currently part of the CS's entourage. Besides this actuation also other environmental dynamics may act on the network of state variables. For example, a heating actuator might increase the state variable 'room temperature', while environmental dynamics (heat dissipation) additionally affect the state variable over time. Concerning our environmental model, actuators are connected to the environmental model such that their influences affect the state variables appropriately by considering their placement, actuation delays, and effect propagation through the environmental model.

Sensors within the interface of a CS can observe a state variable of the physical environment. Usually these observations are limited with respect to measurement resolution, temporal *accuracy*, and rate limits. Consequently, such an observation is only partial and noisy. Similar to actuators, also the sensors need to be appropriately connected with the environmental model in order to take their placement and their capability to make observations (measurement delay) into account.

3.2 Interface Layers

Interface layers allow the discussion of system *interface properties* and their definition in *interface specifications* at different abstraction levels and modeling viewpoints. In the following, three interface layers are introduced: the cyber-physical, the informational, and the service layer. The informational layer is an abstraction over the cyber-physical one, while the service layer structures the behavior of a system in a set of capabilities.

Cyber-Physical Layer
At the cyber-physical layer information is represented by data items (e.g., a bit-pattern in cyber space, or properties of things/energy in the physical world) that are transferred among interacting systems during the Interval of Discourse (IoD). While in this layer there is a distinction between cyber- and physical channels, both share many properties, because cyber channels are implemented by physical channels. Consequently, any interaction over cyber-physical channels is ruled by the progression of time and fundamentally constrained by the speed of light and distance among communicating systems. Time is an elemental property of cyber- and physical interfaces and must be considered at all interaction abstractions. Important properties of the cyber-physical layer are: *signals* (i.e., prearranged representation of information), transmission medium, characteristics of connectors, frequencies, bit rates, energy levels.

Interface properties at the cyber-physical layer are defined in the *Cyber-Physical Interface Specification (CP-Spec)* which consists of the two disjoint specifications: *Interface Physical Specification (P-Spec)* and the *Interface Message Specification (M-Spec)*.

Physical Interfaces
Physical interfaces of CSs are realized by energy transformers that are able to *(1)* take observations from the physical environment, and *(2)* set actions initiated from the

computer system of the CS in the physical environment. An observation in time-aware CPSoSs is a time-stamped measurement of a physical state variable (a property of a thing). A sensor is an interface device that measures the physical environment and produces observations in the form of digital data (a bit pattern), whereas the sensor design determines which property of the physical environment is observed. Sensor-fusion and state estimation [22] are well researched techniques to improve the fidelity of sensor observations. An actuator is an interface device that accepts digital data and control information (e.g., an actuation deadline) from an interface component, and realizes the intended effect in the physical environment (influences physical state variables).

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

The design of sensors and actuators as well as their placement in the physical environment determines the semantics of the digital in-, and/or, output bit-pattern, i.e., effectively form a basic Itom that contains the observation or actuation. In regard to information theory, we often have that some property of a thing (e.g., voltage level) has additional meaning to its receivers (e.g., digital zero and one), while the actual property of the thing becomes irrelevant, after the measured value has been properly abstracted. This refinement process takes place according to a receiver/sender shared conceptual context. It removes intrinsic information and produced a higher level Itom that contains only the extrinsic information. In computer systems, such overlay-meaning needs to be added as meta data until an Itom is formed which the target CSs can interpret and access correctly. The refinement process also removes unnecessary data that is not needed to convey the intended information, i.e., transforms *raw object data* to *refined object data*.

Take for example a speed limit sign at the side of the road which should be interpreted as such by an autonomous car CS. First, a camera sensor of the CS produces a bitmap Itom where the road side including the speed limit is contained as a large array of pixels. Then, for instance machine learning-based methods take this bitmap Itom, segment the image, and finally extract an Itom describing the speed limit sign. At this point the bitmap Itom including its large object data becomes irrelevant and can be removed from the computer system memory. The new Itom is then further contextualized with surrounding/implicit information (e.g., which metric or non-metric system the country where the road is located uses). Finally, it is possible to construct the speed limit Itom consisting of object data that represents a numeric value and meta data which explains (e.g., decimal, km/h, speed limit) the object data to be usable by the CS.

The Interface Physical Specification (P-Spec) describes the properties of sensors and actuators (e.g., sample rates, value/time uncertainties, observation granularity) in order to exchange Itoms with the physical environment according to a specified purpose. On the input side the P-Spec specifies the formation of basic level Itoms from sensor observations. On the output side the P-Spec defines how basic level Itoms are implemented by actuators as influences on physical state variables. The P-Spec together with an environmental model allows for the description of stigmergic channels.

Cyber Interfaces
Cyber interfaces produce and/or consume messages, i.e., bit-patterns in cyber space (e.g., an email) according to the Interface Message Specification (M-Spec). The M-Spec consists of three parts [25]: *(1)* the *transport specification*, *(2)* the *syntactic specification*, and *(3)* the *semantic specification*. The transport specification describes all properties of a message that are needed by the communication system to correctly deliver the message from the sender to the receiver(s). A correctly transported message adheres to all temporal and dependability specifications. The cyber interface consists of ports (channel endpoints) where messages are placed for sending, or received messages are read from. A port has the following properties:

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

The named syntactic units of a message are called *message variables* [25] and are defined in the syntactic specification. Additionally, the semantic specification links the name of message variable to its explanation, i.e., syntactic and semantic specification define the Itom contained in a cyber message.

For cyber interfaces we can differentiate among several types of *compatibility*:

- **Context compatibility:** The same data (bit pattern) is explained in the same way at the sender and at the receiver.
- **Context incompatibility:** The same data (bit pattern) is explained differently at the sender and at the receiver.
- **Syntactic Compatibility:** The syntactic chunks sent by the sender are received by the receiver without any modification.
- **Full Compatibility:** The Itom that is sent by the sender is received by the receiver without modification.

In case of context compatibility, syntactic compatibility suffices to realize full compatibility. In case of context incompatibility a *gateway* is required to translate the data representation of the sender to a data representation that is compatible with the context of the receiver.

Informational Layer
This interface layer concerns the timely exchange of Itoms by unidirectional channels across interfaces. It provides an abstraction over cyber-physical channels to context-independent [27], direct and indirect information flows among systems and

their environment [28]. The abstraction over cyber-physical channels removes any lower-level details of the interactions that is not relevant for describing the information processing behavior of CSs. Itoms at this layer are maximally refined and explicitly specified, i.e., their meta data is available to the extent necessary for all CSs that are possibly involved with these Itoms. Their realization at the lower-level cyber-physical layer must adhere to the semantics specified at the informational layer, otherwise the abstraction is invalid and there is risk of *property mismatch* among interacting CSs. All for the CPSoS service relevant cyber-physical interactions must be taken into account at the informational layer. Otherwise there are *hidden channels* at the informational layer which might compromise security, safety, or may lead to unexpected behavioral detrimental emergence.

Further, the informational layer focuses on modeling the direct and indirect communication among CSs. There are cases where it is beneficial not to model every system involved in an interaction explicitly, but regard them as anonymous common environment of a smaller set of systems of interest which indirectly interact over this common environment. Indirect communication also allows for decentralized coordination of systems [36] and the description of cascading effects [16].

- **Direct Communication:** Itoms are transferred directly and unmodified from one sending to one or more receiving CSs. Consequently, we model a direct channel by a system that simply forwards Itoms from its input to its outputs according to given temporal and dependability properties.
- **Indirect Communication:** The Itoms of a sending CS affects the state of the common environment of one or more CSs. Additionally, the state of this common environment is possibly affected by environmental dynamics, i.e., time-sensitive processes that act autonomously and independently of the explicitly modeled systems. Finally, receiving CSs read Itoms from the common environment by taking observations. The received Itoms represent a superposition of all influences carried out by other CSs or environmental dynamics. In contrast to direct communication, not all CSs participating in an interaction need to be modeled explicitly, as long as their effects are appropriately considered in the model of the environmental dynamics. We model an indirect channel by instantiating an additional Environmental CS (ECS) which incorporates the behavior of the common environment of indirectly interacting CSs.

An Itom channel is characterized by what kind of Itoms the channel can transport, the sender, one or more recipients, temporal properties, and dependability properties. The *Interface Itom Specification (I-Spec)* describes the Itoms exchanged at the system interface, independently of how the information transfer is actually realized. For example: a system ‘car’ notifies cars behind about its sudden change of velocity to an immediate stop by ‘emergency brake’ Itoms. In the cyber-physical layer these Itoms might be implemented by: *(1)* a stigmergic channel between the braking car and the cars behind who observe that the car in front suddenly slows down, *(2)* a stigmergic channel realized by the brake light of the sender and the human operators of the cars behind, and *(3)* a wireless car2car cyber channel between the braking car and the cars behind.

Service Layer
At the service layer, the interface exposes the system behavior structured as capabilities. In contrast to the informational layer, Itom channels are not individually described at the service layer, but only the interdependencies between the exchanged Itoms are specified. If a system with a need is matched with a system that offers the needed capability, the interdependencies must be resolved in the information interface layer with concrete Itom channels. Hence, at the service interface layer there is an instantiable collection of Itom channels per offered capability where generic properties of the Itom channels and their interaction pattern are described.

Systems may provide many services through their interfaces provided that their internal structure can rely on required services. This concept is a fundamental principle in the Service-oriented Architecture (SoA) [12] where components in need of capabilities and components that offer capabilities are brought together by means of a *service registry*, *service discovery*, and *service composition*. A *service provider* is a component that provides a service, while a *service consumer* is a component that uses a service. The service registry is a repository of *Interface Service Specifications (S-Specs)* of capabilities that can be provided by a service provider. Service discovery is the process where service consumers match their service requirements against the available S-Specs in a service registry. Finally, service composition is the integration of multiple services into a new service. The benefits of this service-based view are twofold:

First, there is an immediate reduction of complexity, because one does not need to regard component relations on the basis of single Itom channels anymore. Service consumers can discover services they depend on, and a scheduler can instantiate the necessary unidirectional Itom channels automatically. Further service composition enables the formation of higher-level services based on low-level services. Take the example of a service that provides a humanoid robot the capability to open doors. Such a service would need to implement planning and re-planning of the complex movements realized by lower-level actuation services while constantly taking into account observations (e.g., position of arms, state of the door) from lower-level sensing services.

Second, the coupling of components (integrated in one system) is loose, because the actual constituents of composed services are unimportant background details in the service-based view. This freedom in service composition allows for self-organized system reconfiguration such that the system is able to perform optimally in case new services become available and previously active services become un-available. For example, a service consumer does not depend on a single component to provide the required service, but the service consumer can lookup multiple suitable services from the service registry and choose the optimal regarding computational, communicational, or other costs.

These benefits have been originally observed in the context of free market economy where trading among buyers and sellers led to efficiency in the production and distribution of products.

An S-Spec also includes a set of *quality metrics* that are available for an independent observer to determine the quality of a provided service. Based on these quality metrics, service providers can offer their service under a *Service Level Agreement (SLA)* which consists of *Service Level Objectives (SLOs)* together with the price of a service,

and compensation actions in case an SLO of a committed service was not achieved. An SLO describes a quantifiable service objective based on measurable *quality metrics* that can be monitored independently of the service provider. Service providers can publish their SLA with a reference to the S-Spec of an offered service at the service registry, such that prospective service consumers can find and choose an appropriate service provider.

3.3 Interfaces of a Constituent System

Interfaces within Constituent Systems (CSs) that are not exposed to other CSs or the CS's environment are called *internal interfaces*. A CS is embedded in its environment by its *external interfaces*. When applying the principle of separation of concerns, there are three subtypes of external interfaces: *Time-Synchronization Interface (TSI)*, Relied Upon Interface (RUI), and *utility interfaces*. The TSI enables external time-synchronization to establish a global timebase for realizing time-aware CPSoS. Most important for the integration of a CS in a CPSoS is its RUI which is the interface the emergent and operational CPSoS service relies upon. The optional utility interface is an interface of a CS that does not need to be considered for the operational service of CPSoSs.

The purposes of the utility interfaces are to *(1)* configure and update the CS, *(2)* diagnose the CS, and *(3)* let the CS interact with its remaining local environment which is unrelated to the operative service of the CPSoS. These three purposes justify the introduction of the following utility interfaces: *Configuration Interface (C-Interface)*, *Diagnostic Interface (D-Interface)*, and *Local I/O Interface (L-Interface)*. Figure 2 shows all external interfaces of a CS.

In time-aware CPSoSs, the CSs have access to a synchronized global time base with bounded precision. Such a global time base can be established by external clock synchronization over the TSI to, for example, a Global Navigation Satellite System (GNSS) like GPS. Time-awareness allows for temporally ordering observed events and temporally correctly executing timely available actions in a distributed setting. Naturally, in case the communication or computation subsystem or both fail to deliver or execute an action at its deadline, the execution cannot be guaranteed to be temporally correct. However, in a time-aware CPSoS the *temporal order* of observed events – no matter which CS observed them – can be always determined.

We briefly discuss the utility interfaces. The C-Interface is an interface of a CS that is used for the integration of the CS into a CPSoS and the reconfiguration of the CS's RUIs while integrated in a CPSoS. In time-aware CPSoSs the C-Interface allows to update the interface specification of RUIs to realize time-controlled evolution (see Sect. 5.3). 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 or maintenance cycle of all impacted CSs). 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. The D-Interface is an interface that exposes the internals of a CS for the purpose of diagnosis. Finally, the L-Interface is an interface that allows a CS

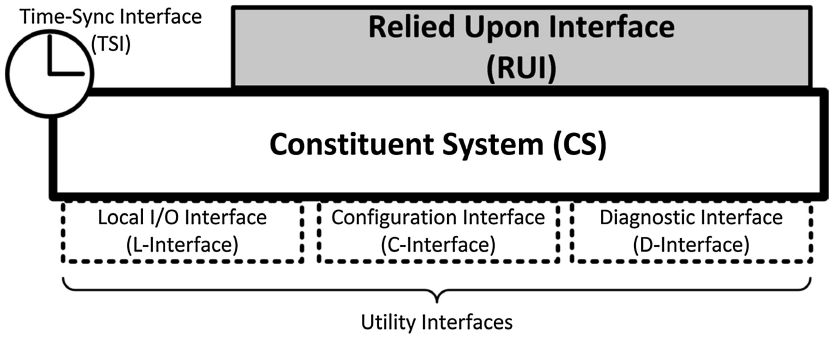

Fig. 2. Interfaces of a Constituent System (CS)

to interact with its surrounding physical reality that is not accessible over any other external interface, for example to realize Human Machine Interfaces (HMIs), or provide other CS-local only services.

A *connected interface* is an interface that is connected to at least one other interface by a channel. Some external interfaces are always connected with respect to the currently active operational mode of a correct system. A disconnected external interface might be the cause of a fault [4] (e.g., a loose cable that was supposed to connect a joystick to a flight controller) which might even lead to a catastrophic failure.

CSs may connect their RUIs according to a *RUI connecting strategy* that searches for and connects to RUIs of other CSs. The RUI connecting strategy is a part of the interface specification of RUIs and searches for desired, with respect to connections available, and compatible RUIs of other CSs and connects them until they either become undesirable, unavailable, or incompatible. For instance, in the global Automated Teller Machine (ATM) network, a cardholder together with a smartcard based payment card form a CS that is most of the time disconnected from any other CSs. The RUI connecting strategy of the payment card CS is influenced by the cardholder's need for cash (desire), nearby located and operational ATM terminals (availability) and whether the ATM terminal accepts the payment card (compatibility).

4 Relied Upon Interfaces

This section discusses the Relied Upon Interface (RUI) model at the previously introduced interface layers, also showing how interface layers are connected. Then the section proposes appropriate execution semantics of the RUI model at the informational layer and closes with a brief discussion of how CPSoS dynamicity is handled by the RUI specification.

4.1 RUI Model Overview

The Relied Upon Interface (RUI) establishes a system boundary of a CS by separating it from its environment. The part of the CS behavior which the CPSoS service relies upon can be observed at the RUI of the CS. Consequently, the interface specification of a CS's RUI hides the possibly complex internal behavior of a CS from the overall CPSoS. However, even more importantly the complexity of the overall behavior of a possibly enormous CPSoS is also hidden from a CS at its RUI. Hence, the RUI specification can be regarded as a complexity firewall because it regulates all interactions taking place across the specified interface. Innate to RUIs, i.e., the points of interactions of CSs, is the transfer of information occurring over these interfaces. It follows an examination of RUIs for each of the three interface layers that we introduced in Sect. 3.2.

RUI Cyber-Physical Layer
Figure 3 gives an overview of cyber-physical interactions at the RUIs of two CSs that are externally time-synchronized and have access to a global timebase. The RUI consists of two sub-interfaces: the *Relied Upon Message Interface (RUMI)* a cyber interface, and the *Relied Upon Physical Interface (RUPI)*.

The RUPI consists of sensors and actuators that take and time-stamp observations of and/or act at a defined deadline on some physical state (e.g., the temperature of a room) in the physical environment according to their design. Environmental dynamics (e.g., heat dissipation through walls) act additionally to other CSs on the physical state. CSs that interact with each other over a common physical environment establish a stigmergic channel [28], i.e., they communicate indirectly by influencing and measuring the physical state.

The RUMI allows *(1)* for the unidirectional transport of state and event messages [25] by means of conventional direct cyber channels, and *(2)* for the indirect coordination with other CSs by means of indirect cyber channels. A state message contains only state observations, i.e., the observed state (e.g., temperature of a room) at a specific instant.

RUI Informational Layer
The informational layer abstracts over informational context-sensitivity, and focuses on direct and indirect information flows among CSs. An indirect channel (cyber or stigmergic) is modelled by instantiating an additional Environmental CS (ECS).

This interface layer is useful during the design of CPSoSs (e.g., model-based design, design space exploration), as well as in the analysis of CPSoSs. For example, we believe that identifying causally related interactions among CSs is paramount for detecting and predicting emergence (see Chap. 3). Naturally, for finding such causal relationships the RUI specifications, the associated interface models, and the environmental models need to be accurate regarding reality, i.e., there should not be any hidden channels. A hidden channel is a latent information flow among CSs that has not been considered by the modeler. Hidden channels might close feedback loops that are believed to be liable for possibly undesired detrimental emergence [28]. In case some

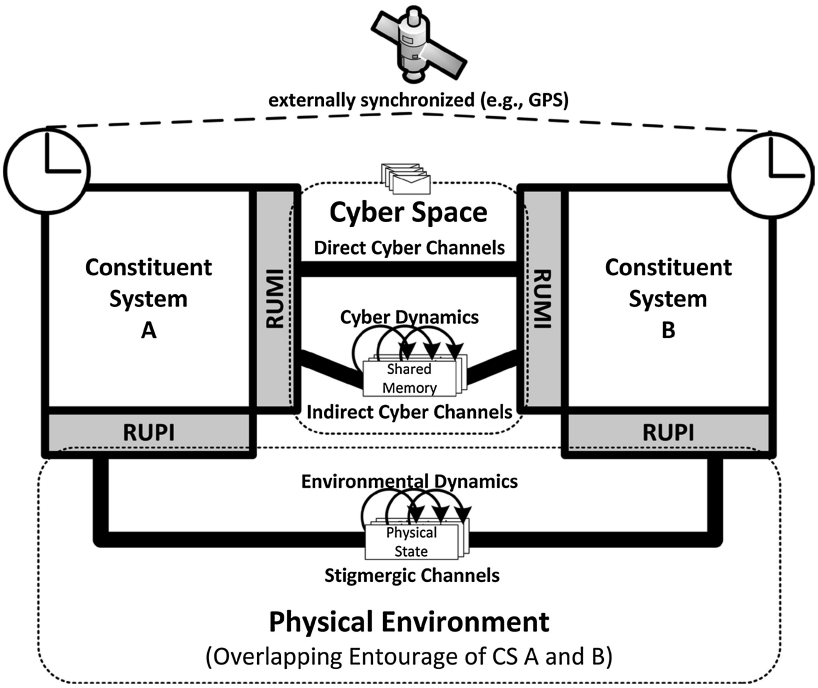

Fig. 3. Relied Upon Interfaces (RUIs) at the cyber-physical layer

behavior is observed at the RUIs of CSs during CPSoS operation, but cannot be reproduced in a simulation at the informational layer, there are hidden channels present that should be identified.

RUI Service Layer

At the service interface layer, we introduce *Relied Upon Services (RUSs)* that are provided at the RUI of a CS. They are described in the Service Specification (S-Spec) of the RUI as a set of RUS-related operations. A service operation is a behavioral abstraction over one or more unidirectional Itom channels. It groups them together and defines their interaction pattern, i.e., the sequence of all operation-related Itoms over all channel endpoints from the perspective of the service provider. Examples of interaction patterns are: request-response, notify, or solicit. Actual Itom channels, or consequently cyber-physical channels are only instantiated (or their provisioning considered) if a RUS is committed to a service requester.

Besides defining the operations of a RUS, the S-Spec also includes a set of quality metrics that allow an independent observer (e.g., a monitoring CS) to determine the quality of a RUS provided at a CS. Based on this quality metrics, a RUS provider can publish its Service Level Agreement (SLA) at the service registry such that service requesters are able to find and request suitable services. RUS providers and consumers are CSs. The service registry – depending on dynamicity and business requirements of

the particular CPSoS – can be either realized as another CS (operated by an SoS authority) to allow for a runtime RUS composition, or it is realized in an off-line manner.

At the service level we model the emergent CPSoS service as a set of dependencies on the required RUSs, such that any CS that wants to use or benefit from the emergent CPSoS service needs to provide these RUSs. However, a CS does not need to directly provide all or even any RUSs a given emergent CPSoS service depends on, as long as the CS is able to request and consume them from other RUS providers.

Example

This section shows in a small example how the interface layers of the RUI are connected. The example CPSoS consists of n interacting CSs. At the cyber-physical interface layer, it contains CSs that interact by using direct and indirect cyber-physical channels. In the informational layer these channels correspond to Itom channels and Itom processing subsystems that implement the behavior of indirect communication. Finally, at the service layer we are able to group channels that are associated with a service and express service dependency relationships.

Cyber-Physical Layer

Figure 4 shows cyber-physical interactions realized by some concrete technology (e.g., data exchanged in cyber space by a TCP/IP network stack, physical location of the CS on a street influenced by actuators). To relate the channels among the CSs and their environment across all interface layers, we draw all channels related to a distinguishable service with the same style. Cyber Channels (CCs) are drawn with a solid line style, while Physical Channels (PCs) are drawn with a dashed line style. Some of the CCs and some of the PCs are labeled for easier identification.

CC 1 and CC 2 are direct cyber channels of the same service (e.g., a database lookup service realized by a request and a response channel). CC 3 and CCs originating from CSs 3 to n-1 are writers of an indirect channel. For example, they publish information whether an alarm occurred. This indirect channel has only one reader (CC 4): CS n which could be an alarm monitor. Further, there is a stigmergic channel realized by PC 1 (actuator which is part of the CS 1 RUPI), physical state variables, and PC 2 (a sensor device of the CS 2 RUPI). Another stigmergic channel (realized in the figure via the overlapping entourages in the lower right of the physical environment) is shown where all CSs are able to influence physical state variables and also observe them, for example the position of CSs on a street.

Informational Layer

Figure 5 shows the example CPSoS at the informational layer. All direct CCs have a corresponding Itom Channel (IC), for example see IC 1 corresponds to CC 1. The indirect CC is realized by an additional Environmental CS (ECS) which implements the behaviour of the indirect CC described by an appropriate environmental model (shared memories with all acting cyber dynamics). Also for each stigmergic channel an additional ECS realizes the environmental model (environmental model with all acting environmental dynamics).

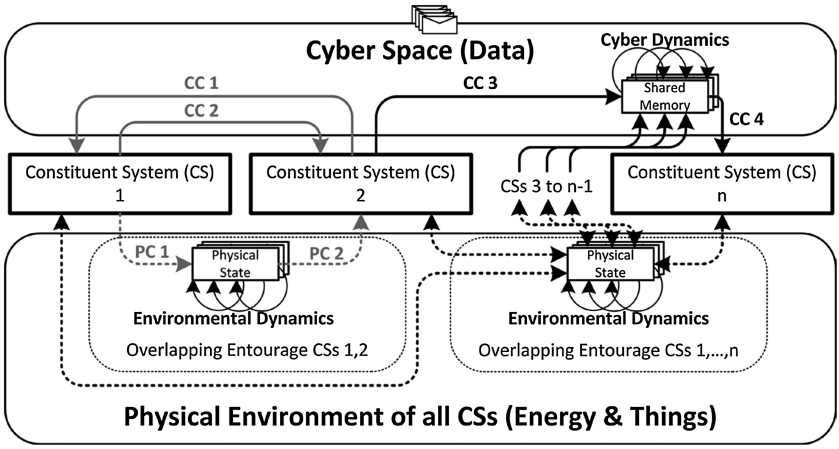

Fig. 4. Constituent Systems (CSs), Cyber Channels (CCs), and Physical Channels (PCs) at the cyber-physical interface layer

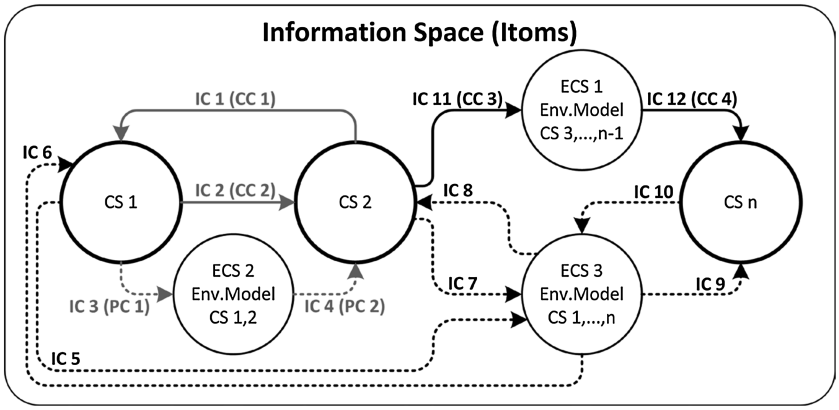

Fig. 5. Constituent Systems (CSs) and Itom Channel (ICs) at the informational interface layer

Service Layer

The service layer of the example CPSoS consists of four services in the dependency relation shown in Fig. 6. For example, service A depends on the environmental services C and D. Service A might be a database lookup service provided by CS 2. The incoming and outgoing arrows on the left of the service vertex symbolize the two Itom channel ports from the perspective of the service provider (one input port and one output port). The three environmental services B, C, and D are provided by ECSs (e.g., environmental service D is provided by ECS 3 in the informational layer). CSs that consume such services need to either act as an influence/writer, or as an observer/reader. The ports on the left side of an environmental service represents the influence/writer service consumer

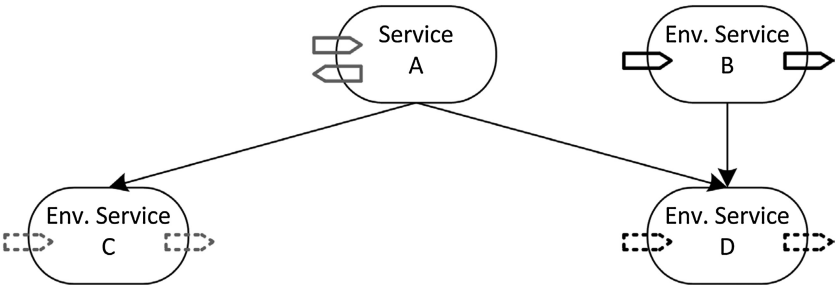

Fig. 6. Service interface layer

(e.g., CS 1 is an influence/writer of ECS 2), and the ports on the right side of an environmental service stand for the observer/reader service consumer (for instance, CS 2 is an observer/reader of ECS 2).

4.2 Execution Semantics of the Informational RUI Model

The interface model describes the part of the system behavior which is observable at that interface. In the previous section we presented an overview of the RUI model. In this section we want to enrich our interface design by suggesting a Frame-based Synchronous Dataflow Model (FSDM) as an execution semantic for the informational layer. Having such execution semantics, allows the detailed study of CPSoSs with respect to behavioral properties at the informational layer in the value domain and in the temporal domain.

Frame-Based Synchronous Dataflow Model
The FSDM is prominent in modeling dependable real-time systems that interact with physical systems or models of them. Further, there are many high-quality tools and languages to design, execute/simulate, and verify such synchronous models, e.g., GIOTTO [20], and Lustre [18].

In the FSDM the dense physical time is discretized into *frames*, i.e., periodic sequences of constant duration. Each of the frames consists of a synchronization phase and a processing phase. During the synchronization phase the input is received (in a sample and hold manner) from the system environment and the output is sent to the environment. In the processing phase the system's function calculates the next state and the output from the input and the current state. Only during the synchronization phase there is interaction between a CS and its otherwise free-running environment. Under the synchronous hypothesis [33] a frame duration is short enough such that the system appropriately reacts to changes in its environment. Hence, the frame duration is determined by the environmental dynamics. For instance, for a keyboard-based Human Machine Interface (HMI) a frame duration of 50 ms is appropriate for most applications, while for a crash detection system in a car a frame duration lower than 1 ms is more appropriate.

It follows a brief overview of the implementation of the CPSoS elements at the informational layer by means of the FSDM:

- **Itom:** Data flows in the FSDM transport Itoms. They are the input and output elements of Constituent Systems (CSs) and Environmental CSs (ECSs) that access, process, or generate them in each frame. Itoms can be described by markup languages, like XML.
- **Direct Itom Channel Model:** A direct channel is connected among one sending and one or more receiving RUI models. It acts in a store-and-forward manner, i.e., the output of this model usually is a delayed copy of the input. Important model attributes are delay, jitter, and optional fault behavior.
- **CS and ECS RUI Models:** RUI models can connect to channel models according to their connecting strategy (see Sects. 3.3 and 4.3). In time-aware CPSoS the internal state of a CS includes the current global time on which input, output and computation actions can be based. ECS RUI models describe the behavior of the common environment of indirectly interacting CSs. They need to integrate the Itoms received from connected CS RUI models in their internal state and apply specified environmental dynamics to this internal state at each frame. The output of ECSs reflects their internal state according to the (often limited) observation capabilities of the receiving connected RUI models. ECSs that model large state spaces and computational intensive environmental dynamics of the physical environment can limit the considered interactions to the overlapping entourages of the involved CSs.

Implementation Considerations
In time-aware CPSoSs the CSs as well as the cyber channels might not be able to guarantee the reaction time constraints imposed by the FSDM. Still, if inputs (and outputs) adhere to the state semantics, there are sophisticated state estimation techniques [22] available to tolerate occasional violations of the synchronous hypothesis (even to the extent of incorporating state observations with significant delay and jitter, e.g., cf. [11], Fig. 2).

In time-aware CPSoSs the CSs can use the available global timebase to drive a periodic control subsystem of a CS as follows: Frame start instants are aligned with a *tick* event of the global time and frame durations are multiples of the *granularity* of the global time. Further, the periodic control subsystem is responsible in the processing phase to activate application tasks, and during the synchronization phase to conduct send and receive actions. There is implementation technology available that readily supports the FSDM. Examples are: TTEthernet [35], TTP/C, or the ACROSS Multi-Processor System-on-Chip [37].

4.3 RUIs Under Dynamicity

Dynamicity describes the reaction or reconfiguration capabilities that have been already considered in the CPSoS design. Therefore, any supported dynamicity needs to be defined in the RUI specifications. We examine three prototypical dynamicity cases:

- dynamicity with respect to connecting two or more CSs at their RUIs,
- dynamicity in making (partial) CS or emergent CPSoS services available to other CSs (RUS composition), and
- dynamicity to adapt to changes in the environment.

RUIs might be connected only for a finite duration within an Interval of Discourse (IoD). A disconnected RUI might be a normal, fault-free interface state and may result at most in CS service degradation, but not in CS failure. This key aspect of RUIs is responsible for the impossibility to establish a static system boundary of a CPSoS. CSs may disconnect and reconnect and they might even be part of multiple CPSoSs (e.g., a modern day NFC enabled smartphone can be part of the global telephone network, the Internet, and the global ATM network which are three large and independently operating CPSoSs).

The RUI connection strategy is part of the interface specification of RUIs that regulates how CSs establish connections. All RUI connecting strategies are local to their respective CS. Still, they influence the dynamic global network topology of a CPSoSs. Consequently, they are co-responsible for the occurrence and regulation of self-organization and emergent phenomena.

At the cyber-physical and informational layers of RUIs it is cognitively complex to describe the dynamic CS interaction that is necessary for accessing a service on the basis of individual channels, because the specific client CSs are unknown before runtime. For example, take a CS that offers a database service. This database service requires a request and a response Itom channel per client CS. For n client CSs one would need to specify 2n dedicated channels at the cyber-physical or informational interface layer. When considering the same situation at the service layer, only the database service together with the two required channels needs to be specified. The mechanisms of service discovery and service composition allow a scheduler to automatically instantiate the request and response channels for each client CS during runtime.

Finally, CSs might need to react to the changing environment and need to reconfigure the set of offered services in order to: *(1)* accomplish overall CPSoS goals (e.g., limit total energy budget, enter a safe state, …), or *(2)* tolerate faults (e.g., suddenly disconnected CSs or failing CSs). At the service interface layer, paradigms known from the Service-oriented Architecture (SoA) are readily available for use. For example, one such paradigm is to replace services with a degraded version, or to replace failed services altogether with services from other (redundant) CSs [21].

5 Interfaces in Evolving Cyber-Physical Systems-of-Systems

Evolution of Cyber-Physical Systems-of-Systems (CPSoSs) concerns design modifications introduced into the interacting Constituent Systems (CSs) that are triggered by changes in the CPSoS environment. Changes of the CPSoS environment might include, for example, advances in technology, or are changes in societal or business needs. Often these needs originate from the desire to change a service towards increased efficiency or the wish to introduce new services altogether. Ultimately, evolutionary changes to the design and consequently operation of the CPSoS should counteract obsolescence in order to keep the CPSoS relevant, increase its business value for involved stake holders, while not deteriorating already provided and still needed services.

When discussing evolution in possibly large and complex systems like CPSoSs we distinguish between *unmanaged* and *managed evolution* [32]. In unmanaged evolution there is no guidance about how a CPSoS evolves. Owners of CSs are free to change services and cooperation with other CSs is motivated by each CS owner's own gain in perceived business value. Facebook, Wikipedia, Google services, and Twitter messaging are examples of CSs whose interfaces are controlled and evolved by the respective owning companies. The composition of such CSs is not driven by a specific central purpose and leads to *virtual SoSs* (e.g., Twitter/Facebook integration of fitness tracking and training applications, or a clever integration of Wikipedia and Facebook to realize file-sharing services). Consequently, unmanaged evolution is most suitable for virtual SoSs where there is no clear central purpose.

In this section we focus on the technical realization of *managed evolution* [32] which is most appropriate for *collaborative SoSs* (but also *directed* and *acknowledged SoSs*, see Chap. 1 for details about SoS classifications). Managed evolution has been originally suggested in the context of large, long-term operational software systems that also show many similarities with CPSoSs (complex functional, semantical, temporal, technical, and operational interdependencies of interacting systems with non-trivially replaceable legacy subsystems). In collaborative SoSs, managed evolution must be planned and supervised by an SoS authority, i.e., an organizational entity (for example established by a CPSoS consortium or enterprise), such that *"the efficiency of developing and operating the system is preserved or even increased"* [32]. The SoS authority has a specific CPSoS purpose in mind, maintains the specifications of Relied Upon Interfaces (RUIs), and has a set of capabilities in order to manage CPSoS evolution. The capabilities are:

- means to introduce changes into a CPSoS by ultimately modifying RUI specifications of CSs,
- monitor the *evolutionary performance* of the evolving CPSoS, and
- give *incentives* to steer the evolutionary process forward, i.e., influence CSs to implement modified RUI specifications.

In the following we discuss scope and challenges of managed evolution in time-aware CPSoSs and in particular investigate evolution at Relied Upon Interfaces (RUIs) of CSs. We attempt to confine risks associated with unexpected detrimental emergence and finally suggest a set of guidelines how to handle evolution in time-aware CPSoSs.

5.1 Scope and Challenges

Managed evolution as described in [32] discusses and addresses challenges related to large, complex, and long-term operational software systems. The comprehensive findings of the authors apply to collaborative SoSs, because they share all characteristics of the systems discussed in [32]. For example, one important challenge the authors address is maintaining *agility* (i.e., the ability to efficiently implement evolutionary changes) while also carrying out necessary changes to increase the system's *business value*.

Further, the authors extensively examine organizational, governance, and even cultural or people aspects of evolving systems, while in this chapter we mostly concentrate on technical aspects. In the following, we identify challenges specifically occurring in the evolution of time-aware CPSoSs that we want to tackle:

- **Continuous Evolution:** Compared to traditional monolithic systems, CPSoSs need to continuously evolve, because replacement of the overall CPSoS as well as a redesign from scratch (green-lawn or greenfield approach) are infeasible with respect to involved costs, risks, or inacceptable operational discontinuities. For example, in the global Automated Teller Machine (ATM) SoS at one point in time a more secure chip-based payment card has been introduced. It is immediately clear that an instantaneous replacement of all payment cards together with the replacement of all Points of Sale (PoS) and ATM terminals worldwide cannot be done because of scaling issues. Consequently, CPSoSs need to evolve continuously, usually during runtime.
- **Multi-version Evolution:** An unavoidable consequence of continuous evolution is that parts of an CPSoS are at different evolutionary states. In large and long-term operational CPSoSs the CSs cannot be replaced or upgraded simultaneously. Hence CSs need to be able to interact with older versions of themselves. Further, CSs cannot be arbitrarily updated and might become at some point legacy CSs that need wrapping to be able to interact with the further evolved CPSoS. In CPSoSs we have CSs in multiple versions and need to take care about appropriately wrapping legacy CSs.
- **Unexpected Detrimental Emergence:** Evolutionary changes might lead to unexpected emergence that is highly undesired (e.g., compromised security or safety properties of an CPSoS). Unfortunately, unexpected emergence cannot be easily or reliably predicated and may only be discovered by accident, because the boundary of a CPSoS is not static and there might by unforeseen environmental effects. For example, consider the British Airways Flight 38 where a Boeing 777 crashed on January 17th, 2008 shortly before the runway at its destination: both engines suddenly failed during landing. The investigation concluded that ice has formed in the fuel system which restricted the fuel flow to both engines [1]. At that time the formation of ice was an unconsidered environmental effect which was only revealed after an accident occurred.
- **Evolving CPSoS Dynamicity:** AMADEOS CPSoSs feature architectural support for adaptive monitoring, analyzing and planning via cognitive and predictive models, and execution of reaction strategies. These architectural means to handle

CPSoS dynamicity need to evolve together with changes in the CPSoS service. For example, in case an evolutionary change allows more efficient use of crossroads and use of street lanes, different traffic situations might arise which require different reaction strategies in order to optimize traffic flow and minimize detrimental effects of faults.

Note that we do not claim that this list of challenges is complete. There might be more challenges related to evolution, especially, if one wants to discuss CPSoSs of specific application domains.

5.2 Local and Global Evolution

We call changes within a CS that do not affect its interactions with other CSs *local evolution*. Local evolution does not modify any of the CS's RUI specifications and consequently remains invisible at the CPSoS level. Still, local evolution is important to optimize the operation of CSs, introduce new or change local-only CS services, or prepare CSs for a pending global *evolutionary step*.

Local evolution harbors the risk of introducing hidden channels (i.e., unconsidered interactions) among CSs which could lead to emergent effects. Hence, local evolution must carefully respect RUI specifications which forbid – in principle – any interaction of a CS with its environment that is not explicitly defined. However, in praxis this is difficult to achieve, especially in relation to stigmergic interactions over a common physical environment. For example, a processor may leak information via its power consumption. In case some local evolutionary change enables an attacker to measure the power consumption, CPSoS security might be compromised.

In contrast to local evolution, we call changes that affect the interactions of CSs and thus the service of the overall CPSoS *global evolution*. As such, global evolution concerns the change of RUI specifications and how these changes are coming into effect. In CPSoSs we have continuous evolution. Consequently, CPSoSs cannot be changed radically, but need to evolve gradually towards changed or new goals in evolutionary steps of limited scope preferably with predictable effects.

Inspired by biological evolution of life (cf. Darwin and natural selection) we regard CPSoS evolution as a tree-like search towards adaptation to environmental conditions. The search space consists of all possible changes that can be realized to address (changed) environmental conditions. Naturally this search space is too large to explore exhaustively; only some of the possible changes are actually realized and can be represented as a tree-like search space exploration. Figure 7 sketches how we view the process of global evolution in CPSoSs. Each of the vertices of the acyclic graph represents a specific *evolutionary state* or version of the CSs making up a CPSoS. The edges in the graph represent evolutionary steps. The vertical dashed line represents the instant *now* which separates the past (versions of CSs that exist and may or may not be in operation) from the future (versions of CSs that are planned and are not operational yet). We assume that some versions of CSs are compatible at the present. In the figure we have indicated two overlapping sets of versions of CSs that are compatible (upper and lower lassos containing a set of vertices each).

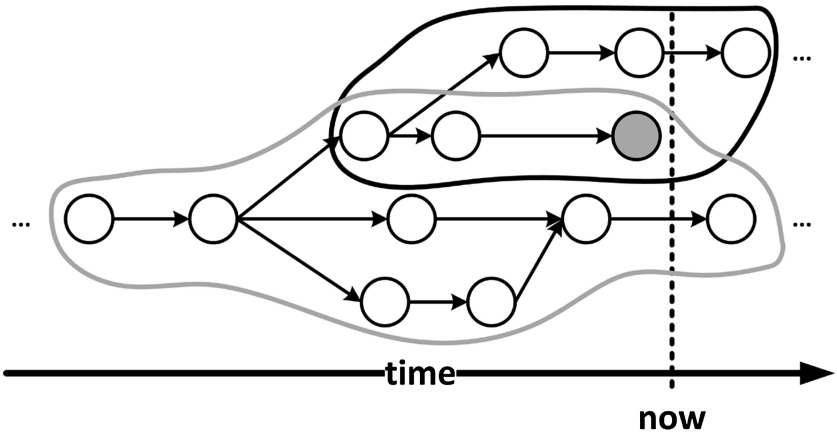

Fig. 7. Tree-like search towards adaptation

Three fundamentally different types of evolutionary steps have been identified:

- **Basic Step:** A basic evolutionary step is a linear, incremental update from one version of a CPSoS to the next one. In Fig. 7 a basic step is represented for example between the two most left vertices.
- **Fork:** In this case two or more different versions evolve from the same ancestor version. For example, a smart grid CPSoS might have evolved in one country up to a certain version which is adopted by a different CPSoS consortium in a different country. Over time these two versions might diverge further up to a point where CSs from one evolved version are not compatible anymore with the CSs from the other evolved version. Figure 7 exemplifies this case after the second vertex from the left where there is a fork into three different versions.
- **Merge:** Finally, two versions of CSs that are part of the same CPSoS might merge in a later version in order to reduce unnecessary functional redundancies, benefit from standardization efforts, or consolidated interfaces. In Fig. 7 this case is depicted in the vertex that has two incoming edges.

Note that in some cases a version can be abandoned and not evolved further as we have illustrated in Fig. 7 by using the shaded version vertex which has no outgoing edge. CSs of such an abandoned version will turn into legacy CSs until they become obsolete.

In terms of managed evolution, forks create mostly business value, while merges increase agility. Basic steps either increase business value or agility. It is in the responsibility of the involved SoS authorities to appropriately control evolutionary steps such that the business value for all stakeholders stays viable while simultaneously agility is not reduced. Both, business value and agility are quality metrics that are composed of CPSoS application-specific quality metrics. For example, the business value of a CPSoS consisting of autonomous cars that should transport humans in a city while minimizing environmental pollution can be assessed by average transportation time per km and average pollution per km.

5.3 Managing Global Evolution at Relied upon Interfaces

Integral to managing evolution in CPSoSs is the establishment of an SoS authority over RUI specifications. The SoS authority needs to carefully plan and execute evolutionary steps that – depending on their magnitude – may require considering the possibility of *unexpected detrimental emergent effects*, and modifying the management of CPSoS dynamicity.

SoS Authority and Management of RUI Specifications
We consider the SoS authority as a mandatory organizational entity in collaborative SoSs that has societal, legal, or business responsibilities in order to keep the CPSoS relevant to its stakeholders. For this purpose, the SoS authority has monitoring and amelioration powers within the CPSoS in order to steer it towards a desired target version. In particular the SoS authority manages RUI specifications and controls how changes of these specifications are rolled out. An SoS authority might be composed of representatives of key CPSoS stakeholders, like CS manufacturers or governments.

SoS authorities select and adopt a suitable set of standards developed by standardization organizations such as the Institute of Electrical and Electronics Engineers (IEEE), the Society of Automotive Engineers (SAE), or the Object Management Group (OMG). The role of standardization organizations in CPSoSs is to provide a stable and broadly accepted conceptual and technological basis for the realization of RUI specifications. For example, the SAE J1708 standard specifies the serial communication (physical layer) of Electronic Control Units (ECUs) in heavy duty vehicles.

In the following we outline a technical realization of RUI specification management on a Service-oriented-Architecture (SoA) approach. *Authorized Relied Upon Service (RUS) specifications (S-Specs)* are administrated at a *service registry* (see Sect. 3.2). Only the SoS authority can authorize and publish S-Specs at the service registry. For example, a CS owner[2] participates in the CPSoS by being a *RUS provider* that offers the RUS in compliance to an authorized S-Spec. In support of multi-version evolution the service registry needs to feature *version management* for authorized S-Specs, i.e., the SoS authority can add new versions of S-Specs to the service registry that coexist with older S-Spec versions. Now owners of CSs that provide RUSs have the possibility to specify in their respective SLAs to which specific version of an authorized S-Spec they refer to. For each supported S-Spec version a different SLA needs to be offered by the CS owner.

Besides managing RUI specifications the SoS authority needs to assess and steer the overall evolutionary process of the CPSoS. The performance of the evolutionary process of a CPSoS is derived from measurable quality metrics describing business value and agility of the CPSoS. Consequently, the measurement of the evolutionary performance can be efficiently integrated in the monitoring and analyzing blocks of the AMADEOS solutions concerning the management of CPSoS dynamicity (see Chap. 7). One example of an important quality metric for the assessment of the

[2] For the sake of brevity we assume here that the owner of a CS is also its manufacturer and user.

evolutionary performance in CPSoSs is the *adoption rate* of existing or newly introduced CSs to new or changed versions of S-Specs.

The adoption rate is controllable by the SoS authority who can give *incentives* in order to move the evolutionary process towards a desired target version. Incentives can be advantages or disadvantages where even monetary penalties apply in case a CS is not upgraded to a more recent version. For example, in the global Automated Teller Machine (ATM) network SoS the payment card industry shifted liability at a specified deadline from money institutes to the Point of Sale (PoS) operators (usually merchants), if they used old and insecure equipment to process payment cards. As the deadline of the liability shift approached and passed, PoS operators risked compensating monetary losses caused by fraud from their own pocket, if an insecure PoS terminal under their responsibility was involved. This strong incentive forced PoS operators to upgrade to more secure PoS terminals where they are not liable in the event of fraud.

Magnitude and Effects of Evolutionary Steps
We define the magnitude of an evolutionary step by considering which of the interface layers (see Sect. 3.2) of RUIs are affected by it:

- **Cyber-Physical Layer:** Technological advances (e.g., different communication protocols, more energy efficient sensors and actuators) may lead to a change of how cyber-physical interactions are carried out. In the case that other interface layers remain unaffected (i.e., there is no change in the information flows) we have a *minor evolutionary step*. A minor evolutionary step does not require any further considerations concerning emergence and managing CPSoS dynamicity. In the context of the AMADEOS Architectural Framework (AF) detailed in Chap. 5, a minor evolutionary step only concerns differences in the implementation level.
- **Informational Layer:** We consider changes at the informational layer as a *major evolutionary step*, because a change in the Itom interactions of CSs needs to be carefully assessed with respect to emergence and the management of CPSoS dynamicity. Regarding the AMADEOS AF a major evolutionary step implies changes at the logical, conceptual or even up to the mission level.
- **Service Layer:** Changes at the service layer are always accompanied by changes in the underlying interface layers. Consequently, a change at the service level also represents a major evolutionary step that has implications on emergence and the management of CPSoS dynamicity.

Methods from Scenario-based Reasoning (SBR) [6] can be employed to pre-evaluate effects of evolutionary steps according to CPSoS-specific quality metrics under quantified uncertainty.

Handling Continuous Evolution
It remains to discuss the transition from one CPSoS version to an evolved version. Continuous evolution in CPSoSs is based on the principle of backward compatibility of its CSs. This backward compatibility needs to be established by upgrading or introducing new CSs that also support the interaction with non-upgraded CSs.

We have already described a multi-version service registry and that Relied Upon Service (RUS) providing CSs can support multiple versions of S-Specs. Now we want to emphasize that also CSs requesting RUS have the possibility to consume different versions of a RUS. In fact, the more versions of RUSs a CS is able to provide or consume, the 'smoother' a CPSoS is able to evolve. Naturally, newly introduced RUSs should not disrupt existing or legacy RUSs.

At some point conflicting or obsolete RUSs need to be retired and old, non-upgradable CSs that depend on them or provide them become incompatible with more recent CSs. In case these old and non-upgradable CSs are still essential for the operation of the CPSoS they become legacy CSs and their service needs to be appropriately wrapped. For example, a wrapping CS can be introduced in an evolutionary step. The wrapping CS is able to offer the service of the legacy CS encapsulated in a version of a RUS that is compatible with all non-legacy CSs.

In time-aware CPSoSs there is the benefit of a global time that allows temporally coordinating the execution of an evolutionary step. The SoS authority can define validity instants in new versions of RUI specifications such that they are switched on at a specific instant. For example, a desired emergent effect may only occur if a critical number of CSs adhere to the new RUS version simultaneously. Also, for some CPSoSs it might be useful to steer its CSs first to a safe, ground, or dormant state, then perform further (physical) upgrades, and finally awake them to interact in the evolved CPSoS (see car-recalls and repair procedures in the automotive domain).

5.4 Avoiding Detrimental Emergence

The occurrence of unexpected detrimental emergence is a problematic case in engineering, operating and evolving CPSoSs. Against our expectations and current predictive capabilities something harmful and possibly catastrophic happened. Why can we not prevent unexpected emergence by design? While the engineering process is indeed responsible for the interaction abilities of the CSs, CPSoSs are also open systems that interact with their environment. This environment of a CPSoS may change over time and/or might be insufficiently understood, i.e., in general we cannot be certain that our models of the CPSoS environment are complete. Also the boundary of a CPSoS is dynamic and influences how the CPSoS environment affects the CPSoS itself. For example, consider a fault-tolerant CPSoS: If the number of its redundant CSs is small, an environment that causes intermittent faults in CSs at a constant rate (e.g., radiation) is much more hostile than compared to the same CPSoS where the number of redundant CSs is large.

An evolving CPSoS attempts to adapt to changes in its environment (real changes or a changed understanding of the environment). Consequently, there are two co-dependent causes that enable the occurrence of unexpected detrimental emergence in CPSoSs:

- An evolutionary step that changes how CSs interact, and
- a change in the CPSoS environment and/or hidden channels.

To the best of our knowledge there is currently no scalable theory to eliminate the first cause with certainty. The number of possible interactions in real-world CPSoSs (physical environment!) diminishes any hope to exhaustively test for all known emergent phenomena, and to evaluate them with respect to their possibly detrimental effects. Even if there was a theory solving this issue, one fundamental problem remains: Identifying unknown emergence, i.e., effects or properties that are conceptually novel at the macro-level (SoS level), but are not present in the non-relational phenomena of the parts at the micro-level (level of CSs).

Unfortunately, the situation is even worse concerning the second presumed cause that enables the occurrence of unexpected emergence. Changes in the CPSoS environment are outside the sphere of control of a CPSoS consortium. Some of these changes may lead to the occurrence of previously rare or unlikely interactions of CSs which may in return result in a (detrimental) unexpected emergent phenomenon. Finally, hidden channels are an unfortunate consequence of our ignorance about the CPSoS environment. They may close causal loops or enable cascading effects which again could trigger (detrimental) unexpected emergence.

In summary, it appears that in principle we cannot prevent all first occurrences of detrimental emergent phenomena with absolute certainty. Further, both an ill-conceived evolutionary step, and (unlucky) changes in the CPSoS environment may lead to undesired emergent phenomena. In the following subsections we discuss a mitigation strategy based on results described in Chap. 3.

Mitigation Strategy

In order to minimize occurrence and subsequent damage due to unexpected detrimental emergence we suggest a mitigation strategy consisting of the following procedures:

- **Augmentation of CPSoS design with expectations about nominal operation:** Relied Upon Service (RUS) specifications should contain assertions that indicate whether the RUS is provided and consumed nominally and according to the designer's expectations. Runtime monitoring implemented in the management of CPSoS dynamicity (see Chap. 7) can check the defined assertions against all interactions of CSs and log as well as timestamp any occurring anomalies.
- **Discovery of the onset of unexpected emergent phenomena:** Quality metrics associated with the onset of emergence (e.g., critical slow-down, density), unexplainable anomalies, and patterns of previously diagnosed and analyzed emergence should lead to the discovery of the onset of (detrimental) unexpected emergence. Again this procedure should be implemented in the management of CPSoS dynamicity.
- **Diagnosis and analysis of unexpected emergence:** After an unexpected emergent phenomenon has been discovered, it must be carefully diagnosed and analyzed. This procedure must reveal the trans-ordinal law and it might expose hidden channels or changes in the CPSoS environment that have not been noticed yet. Based on the result of the analysis inaccurate (environmental) models should be corrected and an appropriate evolutionary step planned to prevent the now expected detrimental emergence.
- **Prevention of detrimental emergence by design:** As soon as new expected detrimental emergence has been found an evolutionary step should be performed

that ameliorates ongoing detrimental emergence and prevents its further occurrence. Amelioration can be implemented in the management of CPSoS dynamicity by deploying suitable reaction strategies (e.g., introduction of randomness to break unintended synchronization [31]). Prevention of detrimental emergence should be implemented by appropriately constraining the RUIs of CSs such that their interactions do not lead to the detrimental emergent effect anymore.

- **Prediction of detrimental emergence:** When planning an evolutionary step, we can predict/search for detrimental emergence by applying analytical methods (e.g., finding causal loops, cascading effects) as well as simulation of models of the CPSoS and its environment.

Detecting Unknown Emergence

Often emergent phenomena are associated with some kind of regularities or shift in densities, but looking for something unknown that does not appear to have any generic and unique characteristics limits our detection abilities. Mogul [31] suggests building a library of signatures of emergence that occur in distributed computer systems and gives interesting examples: trashing caused by 'unlucky' scheduling (not just overprovisioning), unintended synchronization, unintended oscillation or periodicity, detectable by spectral analysis, deadlock and livelock, phase change, chaotic behavior.

While building such a library still requires some decision procedure to classify anomalies as emergent, it would – together with monitoring – allow for efficient detection of already encountered emergence. A decision procedure to classify anomalies could be supported by unsupervised machine learning. Self-organizing Maps (SOMs) appear to be a particularly interesting technique where clusters of structure or density differences in data can be found (see Fig. 8, right).

The SOM [23] is an unsupervised machine learning approach that is based on an artificial neural network of a usually low-dimensional topology (e.g., two-dimensional as depicted in Fig. 8, left). During the training process possibly high-dimensional input data is (after appropriate pre-processing [19]) firstly vector quantized and secondly mapped to the SOM while attempting to preserve the original topology of the input data. Various visualizations for SOMs exist to both make the information contained in the SOM accessible for analysis, but also to allow for assessments concerning the quality of the vector quantization and the quality of the topology preservation. Consequently, visualizations are critical for the interpretation of SOMs. Visualizations often focus on a single or a few aspects of the SOM, i.e., for analyzing SOMs, it is often necessary to study multiple visualizations of them in combination. For example, visualizations can express structural information about input vector density relations, distance relations among mapped input vectors (also called topology), or class information. Further, SOMs can be used to both semantically relate different input samples among one another, and to predict the relation of new input samples to input samples used in training.

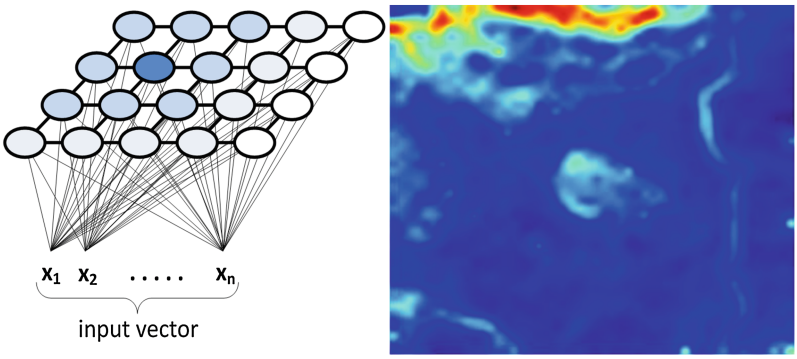

Fig. 8. Self-Organizing Maps (SOMs) reveal emergent regularities in high-dim input data

5.5 Design Guidelines for Evolvable Systems-of-Systems

We conclude this section with a list of design guidelines for evolving CPSoSs that in particular apply to collaborative SoSs:

- Precisely specify temporal properties of RUIs. Local evolution might inadvertently violate RUI specifications, particularly in the temporal domain. Such violations may lead to hidden channels enabling undesired interactions among CSs. The undesired interactions possibly cause unexpected detrimental emergence. Therefore, CSs need to be checked (e.g., by the CPSoS dynamicity management) concerning their conformance to authorized RUI specifications.
- Adopt managed evolution to steer CPSoS evolution in a way such that necessary changes are implemented and the CPSoS remains flexible concerning future changes.
- Implement an SoS authority that has the capabilities to change RUI specifications, to assess the evolutionary state of the CPSoS, and to give incentives to control the onset of the evolutionary process.
- Define evolutionary steps that move the CPSoS towards its changed goal, but limit them in scope such that they remain predictable (with respect to the current knowledge of the CPSoS consortium) in their effects.
- Use the global timebase to temporally coordinate evolutionary steps.
- Use executable CPSoS models (see Sect. 4.2) in simulations and historic data recorded in the evolving CPSoS during runtime to pre-validate planned evolutionary steps.
- Use monitoring and assertion checks at the RUIs (e.g., by management facilities of the CPSoS dynamicity) to validate evolutionary steps. In case of hints about the onset of unexpected emergence update the models and take corrective actions, if possible before a detrimental effect manifests.
- Unexpected emergence appears to be unpredictable in principle even in carefully managed evolution. First occurrence might not be preventable in all cases, therefore use the mitigation strategy described in Sect. 5.4.

- Keep human domain experts in the design loop of evolutionary steps. The CPSoSs that we want to engineer and operate integrate humans, affect humans and should co-evolve with humans. Consequently, only humans are fully capable of judging how to best address a change in the environment of CPSoSs.

6 Conclusion

In this chapter we discussed interfaces in time-aware Cyber-Physical Systems-of-Systems (CPSoSs) for the purpose to investigate behavioral properties of CPSoSs. First, we characterized relevant architectural elements of CPSoSs and introduced three interface layers: the cyber-physical layer, the informational layer, and the service layer. Based on this conceptual groundwork, we identified (among other interfaces) the Relied Upon Interface (RUI) of a Constituent System (CS) as the fundamental interface responsible for the operational behavior of CPSoSs and managing CPSoS evolution.

The RUI is a CS interface on which the global, operational CPSoS service relies upon. We described the RUI model at each of the introduced interface layers, and outlined execution semantics for the informational layer to support the exploration of behavioral effects, like the occurrence of emergence.

In the second part of the chapter we focused on CPSoS evolution and how to manage it at the RUI of CSs. To this end we introduced an SoS authority that is in control of RUI specifications, plans evolutionary steps, and carries them out by changing RUI specifications. We discovered that both – changes in the CPSoS environment and evolutionary changes – harbors the risk of unpredicted detrimental emergence and suggested a mitigation strategy.

Acknowledgments. Warm regards to Sorin Iacob and Andrea Bondavalli for the insightful discussions about stigmergic channels. We thank the following experts for reviewing and helping to improve the chapter: Wilfried Elmenreich (Alpen-Adria-Universität Klagenfurt, AT), Sorin Iacob (Thales, NL), and Wilfried Steiner (TTTech, AT).

References

1. Air Accidents Investigation Branch: Aircraft Accident Report AAR1/2014 – Boeing 777-236ER, G-YMMM, 17 January 2008. Formal report (2008). https://www.gov.uk/aaib-reports/1-2010-boeing-777-236er-g-ymmm-17-january-2008. Accessed 1 Sept 2016
2. Al-Fuqaha, A., et al.: Internet of things: a survey on enabling technologies, protocols, and applications. IEEE Commun. Surv. Tutorials **17**(4), 2347–2376 (2015)
3. Alonso, G., et al.: Web Services. Data-Centric Systems and Applications, pp. 123–149. Springer, Heidelberg (2004)
4. Avižienis, A., Laprie, J.-C., Randell, B.: Dependability and Its Threats: A Taxonomy. Building the Information Society, pp. 91–120. Springer, Boston (2004)
5. Banks, A., Gupta, R.: MQTT Version 3.1.1. OASIS standard (2014)

6. Conrado, C., de Oude, P.: Scenario-based reasoning and probabilistic models for decision support. In: 2014 17th International Conference on Information Fusion (FUSION). IEEE (2014)
7. Caffall, D.S., Michael, J.B.: Architectural framework for a system-of-systems. In: 2005 IEEE International Conference on Systems, Man and Cybernetics, vol. 2. IEEE (2005)
8. Curbera, F., et al.: Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. IEEE Internet Comput. **6**(2), 86 (2002)
9. Da Xu, L., He, W., Li, S.: Internet of things in industries: a survey. IEEE Trans. Industr. Inf. **10**(4), 2233–2243 (2014)
10. Elmenreich, W., Haidinger, W., Kopetz, H.: Interface design for smart transducers. In: Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference, IMTC 2001, vol. 3. IEEE (2001)
11. Engel, J., Sturm, J., Cremers, D.: Camera-based navigation of a low-cost quadrocopter. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE (2012)
12. Erl, T.: Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services. Prentice Hall PTR, Upper Saddle River (2004)
13. Eugster, P.T., et al.: The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) **35** (2), 114–131 (2003)
14. Fan, Z., et al.: Smart grid communications: overview of research challenges, solutions, and standardization activities. IEEE Commun. Surv. Tutorials **15**(1), 21–38 (2013)
15. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Dissertation University of California, Irvine (2000)
16. Fisher, D.: An emergent perspective on interoperation in systems of systems (2006)
17. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding code mobility. IEEE Trans. Soft. Eng. **24**(5), 342–361 (1998)
18. Halbwachs, N., et al.: The synchronous data flow programming language LUSTRE. Proc. IEEE **79**(9), 1305–1320 (1991)
19. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier, Amsterdam (2011)
20. Henzinger, T.A., Horowitz, B., Kirsch, C.M.: Giotto: a time-triggered language for embedded programming. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, pp. 166–184. Springer, Heidelberg (2001). doi:10.1007/3-540-45449-7_12
21. Höftberger, O., Obermaisser, R.: Ontology-based runtime reconfiguration of distributed embedded real-time systems. In: 16th IEEE International Symposium on Object/Component/Service-Oriented Real-time Distributed Computing (ISORC). IEEE (2013)
22. Khaleghi, B., et al.: Multisensor data fusion: a review of the state-of-the-art. Inf. Fusion **14** (1), 28–44 (2013)
23. Kohonen, T.: The Basic SOM. Self-organizing Maps, pp. 105–176. Springer, Heidelberg (2001)
24. Kopetz, H., Suri, N.: Compositional design of RT systems: a conceptual basis for specification of linking interfaces. In: Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 2003. IEEE (2003)
25. Kopetz, H.: Real-time Systems: Design Principles for Distributed Embedded Applications. Springer Science & Business Media, Heidelberg (2011)
26. Kopetz, H.: Why a Global Time is Needed in a Dependable SoS. arXiv preprint arXiv:1404.6772 (2014)
27. Kopetz, H.: A conceptual model for the information transfer in systems-of-systems. In: 2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing. IEEE (2014)

28. Kopetz, H., Frömel, B., Höftberger, O.: Direct versus stigmergic information flow in systems-of-systems. In: 2015 10th IEEE System of Systems Engineering Conference (SoSE) (2015)
29. Mahnke, W., Leitner, S.-H., Damm, M.: OPC Unified Architecture. Springer Science & Business Media, Heidelberg (2009)
30. Maier, M.W.: Architecting principles for systems-of-systems. INCOSE Int. Symp. **6**(1) (1996)
31. Mogul, J.C.: Emergent (mis) behavior vs. complex software systems. ACM SIGOPS Operating Syst. Rev. **40**(4), 293–304 (2006). ACM
32. Murer, S., Bonati, B., Furrer, F.J.: Managed evolution (2011)
33. Potop-Butucaru, D., de Simone, R., Talpin, J.-P.: The synchronous hypothesis and synchronous languages. In: The Embedded Systems Handbook, pp. 1–21 (2005)
34. Quigley, M., et al.: ROS: an open-source Robot Operating System. ICRA Workshop Open Source Softw. **3**(3.2), 5 (2009)
35. Steiner, W., et al.: Ttethernet dataflow concept. In: 2009 Eighth IEEE International Symposium on Network Computing and Applications, NCA 2009. IEEE (2009)
36. Valckenaers, P., Kollingbaum, M., Van Brussel, H.: Multi-agent coordination and control using stigmergy. Comput. Ind. **53**(1), 75–96 (2004)
37. El Salloum, C., et al.: The ACROSS MPSoC–a new generation of multi-core processors designed for safety–critical embedded systems. Microprocess. Microsyst. **37**(8), 1020–1032 (2013)