Received: 17 April 2016 | Revised: 7 March 2017 | Accepted: 22 March 2017

DOI: 10.1002/smr.1878

WILEY Journal of Software: Evolution and Process

SPECIAL ISSUE - HASE 2016

Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile

Marco Mori[1] | Andrea Ceccarelli[1] | Paolo Lollini[1] | Bernhard Frömel[2] | Francesco Brancati[3] | Andrea Bondavalli[1]

[1] Department of Mathematics and Informatics, University of Florence, Florence, Italy

[2] Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria

[3] Resiltech S.R.L, Pisa, Italy

Correspondence
Andrea Ceccarelli, Department of Mathematics and Informatics, University of Florence, Viale Morgagni 65, Firenze, Italy.
Email: andrea.ceccarelli@unifi.itb

Funding Information
European Project, Grant/Award Number: FP7-ICT-2013-10-610535

Abstract

In recent years, more and more efforts have been devoted in supporting the design of systems-of-systems (SoS). Designing such systems is a multidisciplinary problem which involves considering emergent phenomena, assuring the achievement of dependability/security requirements, guaranteeing system responsiveness, and supporting dynamicity/evolution and multicriticality of provided services. A first step towards a viable design approach is to provide a conceptual model of SoS which captures SoS concepts, and their interrelationships aiming at enhancing the understandability of SoS to stakeholders and providing the basis for further automated analysis. In this context, the AMADEOS European project is bringing together researchers and practitioners to provide the support to design SoS starting from the definition of a domain specific ontology serving as a vocabulary for SoS. Our contribution consists in the modeling of the key SoS concepts and relationships defined in AMADEOS adopting a systems modeling language visual modeling language. We propose a systems modeling language profile for SoS, and we show its applicability in a Smart Grid scenario. We show how to use the profile in a model-driven engineering process to support different types of analyses, and we discuss how to integrate the profile in a user-friendly model-driven engineering tool for SoS rapid modeling, validation, code-generation, and simulation.

KEYWORDS

conceptual model, MDE process, SysML Profile, system of systems

1 | INTRODUCTION

A system of systems (SoS) results by the integration of independent, autonomously operating, possibly many, and likely heterogeneous constituent systems (CSs), which are brought together in order to realize a global goal under certain rules of engagement.[1] An SoS approach may offer valuable benefits by reducing cognitive complexity of the engineering and operating methodologies of SoS in a wide range of domains where classical system engineering approaches cannot be easily applied anymore. For example, railway, automotive, smart energy grids, the global automated teller machine network, and crisis management may benefit from adopting an SoS engineering (SoSE) approach. Indeed, different techniques can be found in SoSE to evolve an SoS, handle its dynamicity requirements, achieve time-dependent and dependability/security requirements, early identify, and mitigate detrimental emergence phenomena, and fulfill multicriticality requirements. However, a relevant challenge is to integrate all these techniques in a design methodology that produces a high-level SoS architecture ensuring the delivery of the envisioned global goals. This target architecture should be amenable to refinement and design patterns facilities thus supporting automated analysis wherever possible. With this aim, the adoption of an architectural description language (ADL)[2] is useful for abstracting and understanding SoS design-related problems thus fostering information sharing and reuse among SoS stakeholders and describing an SoS using several viewpoints of analysis. Such a language also reduces development risks and flaws by enabling analysis and experimentation processes at early stages of the design cycle.

Based on the outcomes achieved in the context of the AMADEOS[3] project,[1] seven viewpoints have been selected and explored to define a generic SoS architectural framework and design

[1] The objective of the AMADEOS[15] FP7 project is to bring time awareness and evolution into the design of SoS, to establish a sound conceptual model, a generic architectural framework, and a design methodology, supported by prototype tools, for the modeling, development, and evolution of time-sensitive SoS with possible emergent behaviors.

methodology. Under these viewpoints, we studied SoS in different domains (railway, automotive, smart energy grids, global automated teller machine network, and crisis management) and proposed a meta-requirement model[4] to describe a generic SoS and to support its design, development, and evolution. The identified viewpoints are the following: structure, dynamicity, evolution, dependability, security, time, multicriticality, and emergence.[5] Structure represents architectural concerns of an SoS. In particular, it defines the manner in which CSs are composed[6] and how do they exchange semantically well-defined messages[7] through their interfaces.[8] Dynamicity represents variations to the operation of SoS that have been considered at design time to reconfigure the SoS in specific situations, eg, either after a fault or after the variation of an external condition.[9] Evolution represents changes that have been introduced later to accommodate modified or new requirements by means of including, removing, or modifying system functions.[10] Dependability and security[11] consists of nonfunctional critical requirements as availability, reliability, safety, privacy, or confidentiality. Multicriticality aims at integrating together subsystems providing services with different levels of criticality corresponding to different dependability and security requirements.[12] Time is fundamental since SoS are sensitive to the progression of time, and it is necessary to design responsive SoS able to achieve reliably time-dependent requirements.[13] Emergence mainly denotes the appearance of novel phenomena at the SoS level that are not observable at CSs level; managing emergence is essential to avoid undesired, possibly unexpected situations generated from CSs interactions and to realize desired emergent phenomena being usually the higher goal of an SoS.[14]

Following the viewpoint based analysis, in this paper, we propose a semiformalization of the SoS conceptual model that serves as a domain-independent vocabulary for SoS. To this end, we rely on the systems modeling language (SysML),[15] which is adopted as the mainstream ADL for SoSE. In particular, we define an SoS profile that extends the SysML reference metamodel with specific language constructs, eg, stereotypes and their associations. By using our profile a designer could dynamically apply the SoS concepts to a SysML model of an existing system, effectively elevating the abstraction level from a systems-only perspective to an SoS perspective. Our profile consolidates identified SoS characteristics under well-defined SoS concepts; thus, it decreases cognitive complexity concerning the modeled viewpoints and helps stakeholders to implement desired SoS goals. To demonstrate the clarification effects and overall usefulness of our profile, we applied it in a smart grid use case. We show how to use the profile (1) to model the high-level design of the target SoS architecture, (2) to support different types of analyses in a model-driven engineering (MDE) tool chain, and (3) to solve scalability and usability concerns through its integration in a user-friendly MDE tool for SoS rapid modeling, validation, code generation, and simulation.

We remark that a SysML profile can be rarely considered carved in stone, and minor updates can be identified also once it reaches maturity. The 4.0 version that we report here has reached maturity at the end of a 3-year process, during which the profile has been extensively discussed with members of the AMADEOS project,[16] the External Advisory Board, and the scientific community.[17,18] Further, it has been exercised in use cases whose requirements are defined by industries.[16]

This paper is an extended version of Mori et al.[17] With respect to Mori et al,[17] we present an extensive discussion on the relevant concepts of the conceptual model, and we clarify how they are coded into an SysML profile including graphical examples from selected viewpoints. Further, this work includes 2 additional viewpoints, namely, dependability and security, and enhances the emergence viewpoints; these changes are included to align our description to the final definition of the profile. Additionally, we introduce a user-friendly MDE tool that integrates the SysML profile enabling the design, validation, code generation, and simulation of SoS.

The paper is structured as follows. Section 2 introduces a motivating smart grid scenario in the energy domain for showing the applicability of the SoS profile. Section 3 provides a short introduction of the basic SysML elements and diagrams that will be used in the rest of the paper. The SysML profile for SoS is then presented in Section 4 along with the conceptual model it is based on. Section 5 shows the application of the profile to the scenario thus opening it for viewpoints-driven design and analysis. Section 6 discusses how to use the profile in a MDE process to support different types of analyses, and how to integrate the profile in a user-friendly MDE tool for SoS rapid modeling, validation, code generation, and simulation. Section 7 presents a viewpoint-driven gap analysis for SoS design approaches found in the literature, and Section 8 concludes the paper showing possible future work directions.

2 | MOTIVATING SCENARIO

In a smart grid household scenario, different operationally independent subsystems aim at delivering the desired emergent phenomenon of improving the efficiency and the reliability of the production and distribution of electricity through communication facilities. Requests for energy coming from electronic appliances are forwarded towards the subsystems in charge of granting or denying each request while achieving the smart grid goal, ie, keeping balanced the production and consumption rates for connected households.

Figure 1 shows the topology of the main subsystems involved within a single household of the smart grid scenario. Washing machines (WMs) and microwaves (MWs) are examples of electronic appliances. They represent a flexible load which may initiate an energy request. The smart meter (SM) measures energy consumption and production rates; the distributed energy resource (DER) manages the produced energy through energy generating and storage systems, like wind-powered electrical generators or batteries. A command display shows consumption rates and enables inhabitants to interact with their own energy control system. The energy management gateway (EMG) controls the flexible loads and the DER based on measurements received from the SM and in agreement with the coordinator to establish optimal energy distribution. The coordinator is connected to the neighborhood network access point with the aim of keeping the production and the consumption of energy for a set of connected households balanced. A distribution system operator regulates consumption and production rates at the country level. By means of its load management optimizer, a distribution system operator receives information from a meter aggregator and enacts control decisions in cooperation

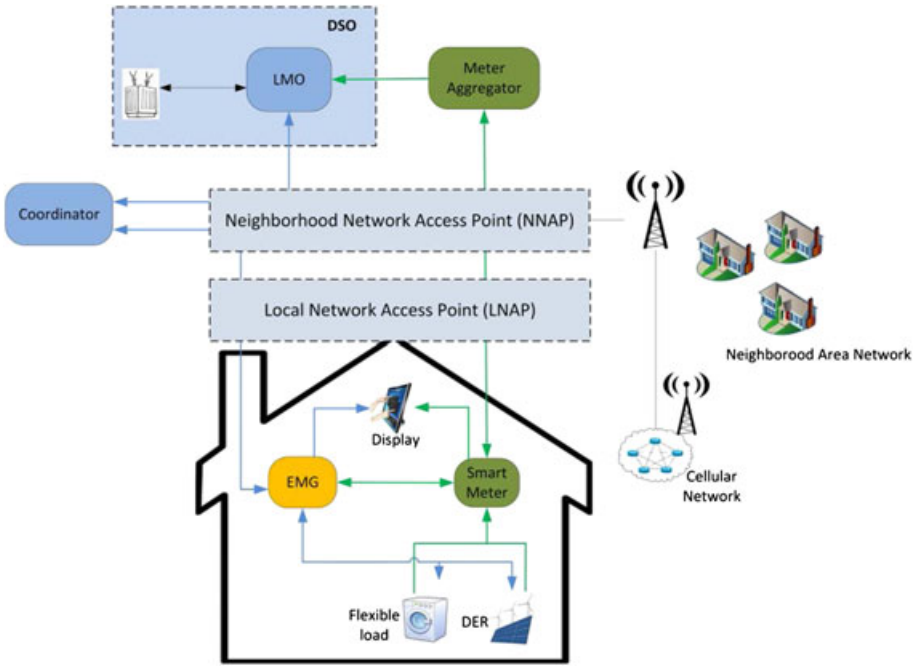

FIGURE 1 Smart grid: energy management scenario. DER, distributed energy resource; DSO, distribution system operator; EMG, energy management gateway; LMO, load management optimizer

with the coordinator. The access to the household is provided by one or more local network access points connected to a neighborhood network access point. All the above mentioned components require proper interfaces to exchange control messages and physical energy entities within and outside the household smart grid.

In a household, electrical appliances have to request when they want to switch on, but they may freely switch off. Consequently, they dynamically connect and disconnect to the grid. However, a decision to turn on electrical appliances is taken at a higher level, as follows: First, the electrical appliance sends a request to the EMG, which periodically receives aggregated consumption and production rates from the SM. Then the EMG, which cannot directly determine how much energy is available in the smart grid, forwards the request to the connected coordinator. The coordinator decides according to the information received by the EMG and on the basis of the current global energy consumption and production rates of the neighborhood, if the request for energy can be satisfied. In a last step, this decision is sent back to the EMG which finally grants or denies the energy request of the initiator electrical appliance. This interaction pattern occurs for many electrical appliances which possibly concurrently request energy in many households contributing to a highly dynamic and—without appropriate modeling techniques—complex smart grid behavior.

Dynamic interactions may also lead to undesired smart grid behaviors that need to be discovered and prevented. In case of an exceptional lighting of a specific public space, a peak of energy request comes from an external EMG (in the neighborhood). Let us now suppose that the corresponding SM fails in transmitting aggregated consumption and production values to its EMG. The latter forwards wrong information to the coordinator which consequently fails in maintaining balanced consumption values, eg, by allowing more request to the households then it is possible. This will then result to a blackout of the household grid. This detrimental emergent phenomenon cannot be captured if we only look at the interactions of the internal household subsystems without considering the external EMG. Appropriate means to describe and recognize interactions that lead to detrimental emergent phenomena are essential to prevent possible negative consequences.

Smart meters of the households may be subject to faults which may hamper their availability and compromise the safety of the whole smart grid. To this end, appropriate counter measures have to be modeled in the design process to early identify possible problems for which solutions are already available according to common standards. Solutions have to be applied, which may span from replication mechanisms to improve availability to error detection techniques aiming to guarantee a certain tolerable hazard rate (THR).

The security of communication within the smart grid has to be carefully considered to avoid the intrusion of third party and the correct functioning of the grid. The protocol to adopt is the Open Smart Grid Protocol adopting RC4 and EN14908 as encryption/decryption algorithm which exploit the OMAK symmetric key defined in the Open Smart Grid Protocol.[19] It is, thus, required to define a security protocol in the design model, its encryption/decryption algorithm and key and link them to the part of smart grid that shall be secured.

Our smart grid scenario shows also that different subsystems may have different levels of criticality. Indeed, the service of public event lighting (PEL) has a higher level of criticality regarding the service provided by MWs and WM in a household. The latter can be interrupted with no detrimental consequences while the interruption of lighting for a public event may cause more severe problems.

Finally, a smart grid is also subject to the introduction of new technologies aiming at maximizing its business value. A new device, a smartphone, may have to be included in the household to replace the command display (ie, a desktop device) to show consumption/production rates and supporting new interactive actions, ie, turning on and off

a device directly from the smartphone. This scenario represents a possible evolution which has to be fully characterized to describe its impact on the Smart Grid.

3 | BASICS ON SysML

The SysML[20] is a general-purpose graphical modeling language, defined by the Object Management Group, based on the well-known unified modeling language (UML[2]). The SysML supports specification, analysis, design, verification, and validation of a broad range of systems, and it includes 9 diagrams instead of the 13 diagrams from UML, making it a smaller language that is easier to learn and apply. It provides structure diagrams to describe system structure and components, and dynamic diagrams to model the behavior of the system. The diagrams that we will use in the rest of the paper are the block definition diagram (BDD—structure diagram), and the sequence diagram (dynamic diagram). The official Object Management Group SysML webpage[3] contains all the details on the available SysML features and diagrams, also in the form of tutorials (eg, Friedenthal[21]).

Blocks in SysML BDD are the basic structural element used to model the structure of systems, and they can be used to represent systems, system components (hardware and software), items, conceptual entities, and logical abstractions. Blocks are shown as UML classes stereotyped "block" and are depicted as a rectangle with compartments that contain block characteristics such as name, properties, operations, and requirements that the block satisfies. A block provides a unifying concept to describe the structure of an element or a system: system, hardware, software, data, procedure, facility, and person. This type of diagram helps a system designer to depict the static structure of an SoS for its CS and possible relationships.

A sequence diagram represents the items involved in a scenario or interaction, and the messages that are exchanged in a chronological order. Items in a sequence diagram are represented by a lifetime. These lifetimes can be generic instances, or instances from blocks defined in the model; instantiating blocks on sequence diagrams establish a link with the static (BDD) system model.

Another important element to be introduced is the concept of profile. While UML provides various generic concepts for software and systems modeling, it cannot cover all possible application scenarios; instead, it can be extended with profiles to add custom model elements to suit the specific needs. Therefore, a profile is a generic extension mechanism for customizing UML models for particular domains and platforms. Profiles are defined using stereotypes, tag definitions, and constraints which are applied to specific model elements, like classes, attributes, operations, and activities. A profile is a collection of such extensions that collectively customize UML for a particular domain or platform. SysML itself is actually defined as an extension of a subset of UML using UML's profile mechanism. The SoS profile we present in this paper further extends the SysML reference metamodel with specific language constructs for modeling the key concepts and relationships in the domain of SoS.

[2]www.uml.org

[3]http://www.omgsysml.org/

4 | SysML PROFILE BASED ON CONCEPTUAL MODEL FOR SoS

A conceptual model consists of a set of stable and unambiguously defined concepts (ie, categories) and semantic relations among them. A conceptual model provides a domain-specific ontology representing a vocabulary for domain discourse. A group of experts may commit to such a domain-specific ontology; hence, they establish a shared view on a domain and are able to collaborate with respect to the domain.

In the AMADEOS project, a conceptual model for SoS[22] has been conceived to find a common language allowing experts to collaborate on modeling, engineering, and analyzing SoS. It is structured in several viewpoints on SoS which we introduced in Section 1. In this section, we give a brief overview of a subset of the concepts contained in the viewpoints and how we have modeled them in a SysML semiformal representation organized in a profile[4] composed by viewpoint-related packages. To this end, we have defined specific constructs and we have exploited already implemented stereotypes available in other related profiles to support specific viewpoints. Our proposed profile is meant to be used by designers in describing the static SoS structure and its dynamic behavior according to the introduced viewpoints. Such an SoS description can be adopted to be kept consistent across viewpoints by tools and for machine-assisted cross-viewpoint analyses (eg, finding detrimental emergent SoS behavior).

In the following, we enlist the solutions we envision in our profile to the needs raised by each of the viewpoints.

4.1 | Structure package

The static structure of an SoS is based on the concept of a CS, which is "An autonomous subsystem of an SoS, consisting of computer systems and possibly of a controlled objects and/or human role players that interact to provide a given service." A CS exchanges information that is either represented by things/energy or data with its environment by means of interfaces. The environment of a CS includes all entities that are able to interact with the CS, including other CSs. In our context, information is any kind of timed proposition about the state of an attribute of an entity which is either an attribute of a physical thing (eg, temperature of a room) or an attribute of an abstract construct (eg, execution time of a program).

The interfaces among which the CSs interact with one another are the relied upon interfaces (RUIs). As such the CS service—which is its intended behavior—is provided at this interface. The RUI is further structured in the relied upon message interface (RUMI) and the relied upon physical interface (RUPI). The RUMI allows for message-based communication of CSs over cyberspace (eg, the Internet), while the RUPI enables the indirect physical exchange of things or energy among CSs over their common environment (see Kopetz and Fromel[23] for details).

The profile supports the description of the static and dynamic structure of an SoS representing: the basic architectural elements and their semantic relationships; the sequence of messages exchanged

[4]https://github.com/AMADEOSConceptualModel/SysMLProfileAndApplication.git—GitHub public link to the AMADEOS SysML profile and the Smart Grid application

among CSs in an SoS; the points of integration, ie, interfaces, allowing the exchange of information/energy among connected entities.

The structural properties of an SoS are described using 3 different subpackages: "*SoS Architecture*", "*SoS Communication*", and "*SoS Interface*" (see the literature[22]). The first defines stereotypes useful to describe the topology of an SoS; the second provides stereotypes to describe the communication aspects between the CSs of an SoS; finally, "*SoS Interface*" semiformalizes internal and external points of interaction of an SoS. For the sake of brevity, in this paper, we focus on the "*SoS architecture*" subpackage that is the foundation for building any type of SoS.

4.1.1 | SoS architecture subpackage

Architectural components are defined within the "*SoS Architecture*" subpackage (see Figure 2). This package extends SysML BDD to model the topology and the relations of an SoS.

The first stereotype is "entity" (something that exists as a distinct and self-contained unit), and it extends the SysML metaclass "Block". We distinguish between 2 different kinds of entities: "thing" (a physical entity that has an identifiable existence in the physical world) or "construct" (a nonphysical entity, a product of the human mind, such as an idea). They extend the properties of entity, so they are also represented as blocks.

A "System" is a type of entity (thereby a Block); it has the same characteristic, but it is also capable of interacting with its environment. As it is expressed by the "sys_type" enumeration, a system can be

- "autonomous"—a system that can provide its services without guidance by another system;
- "monolithic"—if distinguishable services are not clearly separated in the implementation but are interwoven,
- "open" (or "closed")—a system that is interacting (or is not interacting) with its environment during the given time interval of interest,
- "legacy"—an existing operational system within an organization that provides an indispensable service to the organization,
- "homogeneous"—a system where all sub-systems adhere to the same architectural style,
- "reducible"—a system where the sum of the parts makes the whole,

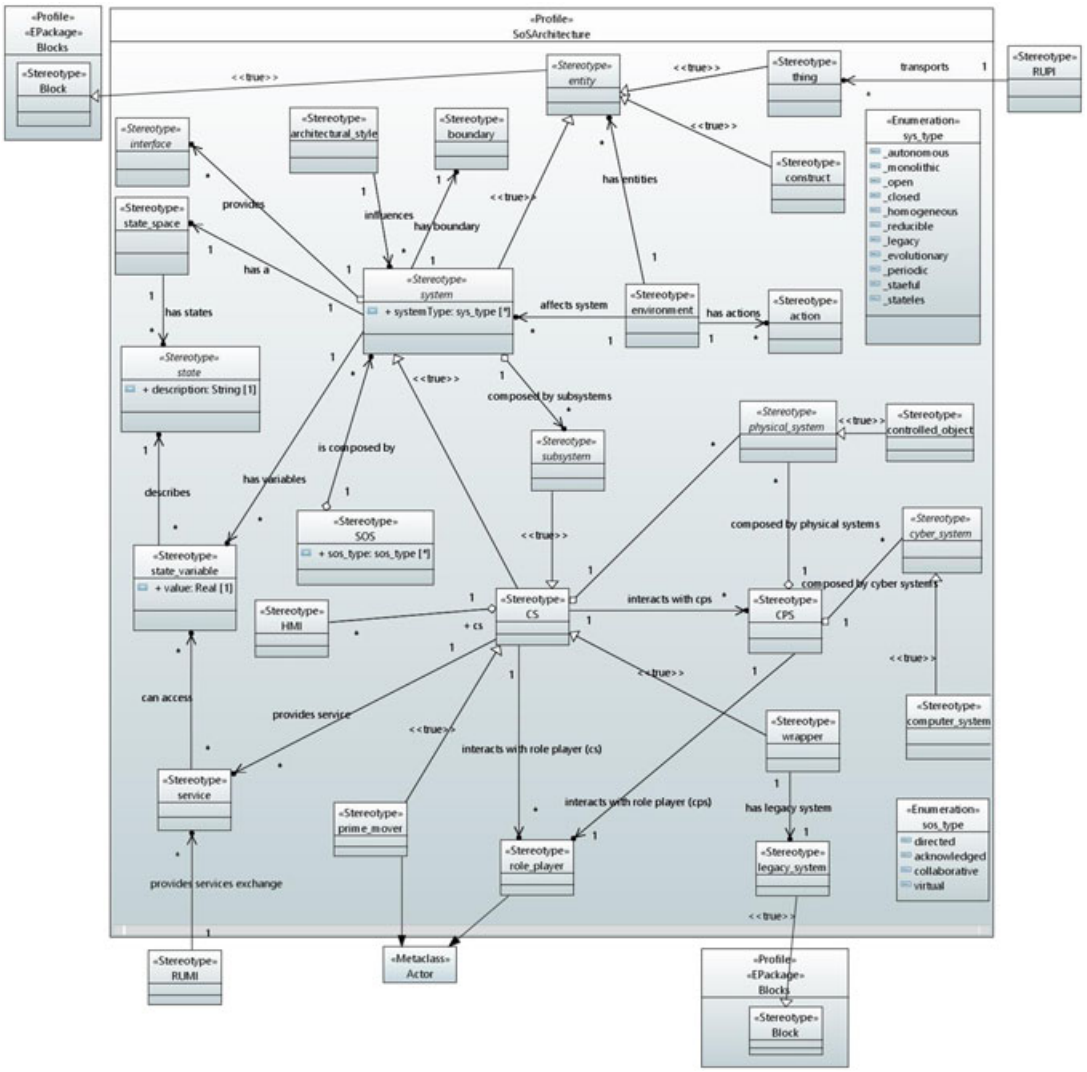

FIGURE 2 Systems-of-systems (SoS) architecture subpackage. CPS, cyber-physical system; CS, constituent system; HMI, human machine interface; RUMI, relied upon message interface; RUPI, relied upon physical interface

- "evolutionary"—a system where the interface is dynamic (ie, the service specification changes during the given time interval of interest),
- "periodic"—a system where the temporal behavior is structured into a sequence of periods, and
- "stateful" (or "stateless")—a system that contains (or does not contain) state at a considered level of abstraction.

Every organization that develops a system follows a set of explicit or implicit rules and conventions, eg, naming conventions, representation of data (eg, endianness of data), protocols, etc when designing the system. This set of explicit or implicit rules and conventions is called the architectural style, which is represented by the stereotype "architectural_style". A system can provide a communication "interface", and it has a "boundary" (a dividing line between 2 systems or between a system and its environment). A "subsystem" is a subordinate system that is part of a system and it is related to "system" by a composite relation.

A CS is an autonomous subsystem of an SoS, consisting of human machine interfaces "HMI" and possibly of physical "controlled_object" and it provides a given "service" by interacting with "role_player" through the "RUMI". The RUMI is a message interface where the services of a CS are offered to the other CSs of an SoS, and "RUPI" stereotype represents a physical interface where things are exchanged among the CSs of an SoS. A wrapper represents a new system with at least 2 interfaces, which is introduced between interfaces of the connected component systems to resolve property mismatches among these systems, which will typically be *legacy_systems*. A prime mover is a human that interacts with the system according to his or her own goal. In the profile, the "wrapper", the "legacy_system", and the "prime_mover" are CS, which is a stereotype that extends the property of "system" that contains multiple "sub_system", which in turn can be CS. A system has a "state_space" composed of states described by the variables that may be accessed by the CS service. In addition, a CS interacts with cyber-physical systems. The "SOS" stereotype represents the integration of systems, ie, CSs that are independent and operable, and which are networked together for a period to achieve a certain goal. As expressed by the "sos_type" enumeration, an SoS can be

- "directed"—an SoS with a central managed purpose and central ownership of all CSs;
- "acknowledged"—independent ownership of the CSs, but cooperative agreements among the owners to an aligned purpose;
- "collaborative"—voluntary interactions of independent CSs to achieve a goal that is beneficial to the individual CS; and
- "virtual"—lack of central purpose and central alignment.

A cyber-physical system ("CPS") is composed by a set of "cyber_system" (ie, computer systems), and "physical_system" (ie, controlled objects).

4.2 | Time package

The progression of time enables changes, ie, dynamicity and evolution, in SoS. In the AMADEOS project, it has been concluded that a global sparse timebase is fundamental for reducing cognitive complexity in understanding aspects related to all nonstatic investigated viewpoints on SoS. For example, a sparse global timebase allows establishing consistently—across all CSs—a temporal order among sparse events, regardless which CSs originally produced these sparse events.

We call time-aware SoS those SoS whose CSs have access to such a sparse global timebase. The global timebase in time-aware SoS is typically established by external clock synchronization, eg, GPS. Further, the sparse global timebase is essential for the temporal precise coordination of distributed interactions with the common environment of involved CSs. For instance, in a time-aware SoS the specification of RUIs can refer to exact points in the sparse time when information should be exchanged (eg, messages sent and received). This tremendously simplifies the agreement of the temporal occurrence of distributed actions. For example, in the cyber domain, it is possible to limit the generation of events (eg, messages) according to ticks of the sparse global timebase and no explicit agreement protocol is required.

The profile supports the definition of responsiveness SoS by enabling the achievements of time-dependent requirements through a global time base. To this end, it allows the instantiation of time concepts extended from Modeling and Analysis of Real-time Embedded Systems (MARTE) profile,[24] which supports the design of real-time and embedded systems (eg, by extending the clock stereotype to support an SoS global time base). Our SysML profile includes the relevant stereotypes from the MARTE profile that are required to describe the timing properties of a System of Systems.

4.3 | Dependability package

Dependability is "The ability to avoid failures that are more frequent and more severe than is acceptable." A failure, ie, a deviation of the system behavior from its intended behavior, is resulted from an error state (eg, flipped bits in a data word) within a system which was caused by a fault (for example, an electromagnetic pulse outside system specifications). A system outage describes the interval where a system does not provide the intended behavior. System restoration is "The transition from system failure to intended system behaviour."[11]

The profile supports the definition of dependability concerns in an SoS by supporting the definition of dependability guarantees which refers to different dependability measures (ie, robustness, safety, integrity, maintainability, availability, and reliability), for which a set of techniques may have to be applied (ie, fault forecast, fault tolerance, fault removal, and fault prevention).[11]

Security and multicriticality are other nonfunctional aspects of systems that relate to unacceptable failures. To emphasize their importance in our viewpoint-based conceptual model, we discuss them separately.

4.4 | Security package

Security is concerned with establishing confidentiality, integrity, and availability for authorized actions only. These nonfunctional attributes are usually accomplished by symmetric or asymmetric cryptographic means (eg, encryption and hashes). Authorization is realized by access control policies and is usually augmented by authentication to ensure the identity of the authorized subject.

To discuss attacks and mitigation strategies on security, the AMADEOS conceptual model introduces a threat as "Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, or other organizations through a system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service." Security attacks exploit system vulnerabilities. Risks quantize the likelihood of specific threats and their impact.

The profile supports the application of security concepts to achieve the encrypted transfer of messages among CSs according either to a public key or asymmetric cryptography mechanism. Access controls policies are also supported to allow users in getting authenticated to the SoS and authorized for a set of granted actions to the SoS. To this end, a monitoring infrastructure following security policies to detect security incidents and vulnerabilities may be put in place.

Security viewpoint is limited to express the concepts identified in the AMADEOS conceptual model, which constitutes a domain-independent vocabulary for SoS. However, this can be potentially expanded with novel concept as well as introducing existing profiles, for example, UMLsec.[25]

4.5 | Evolution package

In contrast to dynamicity, the concept of evolution relates to all changes of an SoS that are not given by requirements and thus part of the design, but arise by changes in the environment (primary evolution), or by new or changed requirements on the SoS service itself (secondary evolution). Note that in the context of SoS, we are only concerned with evolution that affects the behavior of CSs relevant for the SoS. Local evolution within single CSs that is not observable at the RUIs has by definition no effect on the global SoS service and is of no interest to us. In prospect to formalize a methodology which allows evolution to take place in a controlled manner, the concept of managed evolution is most relevant. It is defined as the "evolution that is guided and supported to achieve a certain goal."[9] Evolutionary change is often associated with an increase of the business value of an SoS.

The profile supports the definition of the elements to describe the process of gradual and progressive change of an SoS. Among others, it supports the identification of the evolution type, the objective it aims at achieving and the involved system resources.

To describe this type of processes we have chosen a BDD, because it is designed to show the generic characteristics and structures of a system.

The main SoS concepts related to evolution are modeled within the "*SoS Evolution*" package of our SoS profile. Figure 3 shows the "evolution" stereotype as a block of a BDD, aiming at describing an SoS change. In our conceptual model, we envision 2 different types of evolution:

- "managed_evolution"—Process of modifying the SoS to keep it relevant in face of an ever-changing environment. Examples of environmental changes include new available technology, new business cases/strategies, new business processes, changing user needs, new legal requirements, compliance rules, and safety regulations, changing political issues, new standards, etc.
- "unmanaged_evolution"—Ongoing modification of the SoS that occurs as a result of ongoing changes in (some of) its CSs. Examples of such internal changes include changing circumstances, ongoing optimization, etc.

An SoS evolution has a "goal", it improves the "business value" (overarching concept to denote the performance, impact, usefulness, etc of the functioning of the SoS) by the exploitation of the "system_resource" (renewable or consumable goods used to achieve a certain goal, eg, a CPU, CPU time, and electricity) and can be affected by the "environment" (entities and their actions that are not part of a system but have the capability to interact with the system). Evolution is achieved by modifying CSs and consequently the whole SoS.

4.6 | Dynamicity package

Dynamicity concerns all changes or configurations an SoS can exhibit by design. Most importantly it encompasses all CSs interactions (eg, message or things/energy exchanged over time). Consequently from the viewpoint of dynamicity the service of all CSs is exposed at the RUIs where observable inputs and outputs can be conveniently (1) described in interface specifications for engineering purposes and (2) monitored for diagnosis purposes.

Dynamicity also concerns reconfigurability, which is the ability of a system to change its configuration according to the current demands. For example, in SoS, we often do not have statically connected CSs, but CSs enter and exit autonomously a given SoS over time. The involved CSs must be able to reconfigure themselves accordingly. Conceptually, we modeled this again at the RUI by means of a RUI connection strategy which is "[…] searches for desired, with regards to connections available, and compatible RUIs of other CSs and connects them until they either become undesirable, unavailable, or incompatible."[22]

The profile supports the definition of the dynamic structure and behavior of an SoS. It supports the elicitation of the dynamic elements in an SoS by also allowing the definition of different type of dynamicity. Beyond the static representation of dynamicity, the profile supports also the specification of the dynamic behaviors through the exchange of messages among CSs as also enabled for the structure viewpoint.

4.7 | Multicriticality package

A multicriticality system is a system delivering at least 2 services of different criticality levels, ie, different levels of dependability/security requirements. For example, safety is "The absence of catastrophic consequences on the user(s) and on the environment" and as stated in Bums et al,[26] in many safety standards, "Up to five levels may be identified (see, for example, the IEC 61508, DO-178B, DO-254 and ISO 26262 standards)." Multicriticality can also be applied to services, ie, intended (sub)behavior of a system. A critical service requires a certain level of dependability and security, eg, safety. In certification, it is

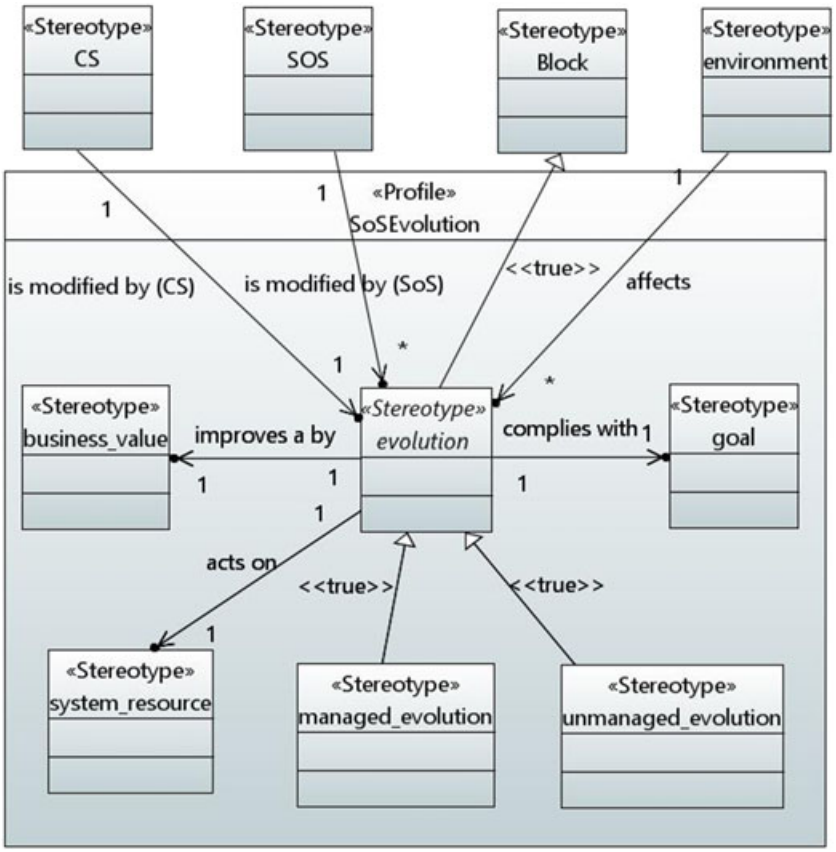

FIGURE 3 Systems-of-systems (SoS) evolution package. CS, constituent system

required practice to declare the criticality level of a system as high as its most critical delivered service unless evidence can be supplied that more critical services can be provided in sufficient isolation (eg, dedicated power supply and dedicate and statically reserved processing time) to less critical ones. Consequently, partitioning and isolation mechanisms are important in systems that support multicriticality.

The profile supports the integration of CSs (through their provided services) having different criticality requirements. To this end, it allows the definition of services being critical according to possibly different levels to which it may correspond different dependability and security requirements as enabled for the dependability and security viewpoints.

4.8 | Emergence package

In the AMADEOS conceptual model, emergence is defined as "A phenomenon of a whole at the macro-level is emergent if and only if it is new with respect to the non-relational phenomena of any of its proper parts at the micro level." Consequently emergent behavior is observable at the macro level (eg, a traffic jam) which cannot be reduced to the behavior of one of the parts at the micro level (eg, a single car analyzed in isolation). If an emergent phenomenon can be described by a transordinal law, ie, a law that explains the emergent phenomenon at the macro level from properties or interactions of parts at the micro level, it is explained emergence. In case such laws have not been found (yet, or maybe they do not exist at all), it is unexplained emergence. An explained emergent phenomenon can be classified as expected (transordinal laws are known), or unexpected (transordinal laws are not known). Orthogonally another classification of the emergent phenomenon with respect to the SoS goals seems reasonable: beneficial or detrimental. Naturally, unexpected detrimental emergence marks a very problematic case which is still under open research. For an in-depth discussion about emergence in SoS, we refer to Kopetz et al.[27]

The profile supports the description of emergent phenomena according to their nature and the CSs that may be affected. Further, it relates the descriptions of the emergent phenomena to the dynamic interactions among the former identified CSs (cf. structure viewpoint).

5 | APPLICATION OF THE VIEWPOINT-BASED SoS PROFILE

In this section, we describe our SoS profile by illustrating how we have applied it to solve the viewpoint-driven needs raised by the energy management scenario. Consequently, we do not present the meta-models of the profile defined through SysML BDDs (where blocks are the basic structural element used to model the structure of systems[14]), but we do present how most of the BDDs can be applied to solve specific problems at hand for a given viewpoint. Interested readers may refer to the literature[22] for further details on SoS profile meta-models and illustrative applications.

5.1 | Structure viewpoint

To support the definition of the SoS structure, the profile contains a BDD to model the topology and the relations of an SoS.

Figure 4 shows how the SoS structure BDD can be applied to our household scenario.

The latter is modeled as an SoS able to produce and reliably distribute electricity by means of its entailed CSs. The EMG is a block and it is stereotyped as a «*cs*»; similarly, SM (ie, the SM) and *CommandDisplay* are each stereotyped as «*cs*». The *Flexible Load* is a «*cs*» and it is composed by a set of appliances: MW, WM, clothes dryer, etc. These appliances are switched on and off dynamically on the basis of the current needs. Each CS is associated with a RUMI and RUPI to transfer control messages and energy, respectively. Nevertheless, by means of this diagram alone, we are not able to describe the actual interactions occurring among CS. We modeled in our profile the message-based communication among CSs over their RUMIs by defining a BDD for a set of stereotypes describing the main characteristics of communication protocols. The application of this diagram consists in adopting a sequence diagram, ie, a UML diagram representing the behavior of a system in terms of a sequence of messages exchanged between parts. The latter are represented as lifelines defining the individual participants in the interaction. The time is showed by the length of the lifeline and it passes from top to bottom while messages are exchanged. Through such sequence diagrams, we aim at representing the exchanged information during the progression of time among the CSs which have been formerly identified. Figure 5 shows an example of the main communication concepts to represent how the SM collects consumption rates from the appliances in order to periodically forward aggregated values to the EMG.

In this scenario, a message flowing from MW to the *SmartMeter* contains the energy consumption rate (*data_field* = 2 kW) and some additional information as displayed in the comment box of the sequence diagram.

5.2 | Evolution viewpoint

To describe the evolution process, our profile defines a distinct BDD with a set of specific stereotypes. Figure 6 represents an application of the profile to our scenario. It shows the evolution of the SoS caused by the replacement of the device controlling consumption and production values.

As shown in the figure, evolution is represented with a *Household_evolution* block which is stereotyped as «*managed_evolution*». It consists of introducing a new technology (*NewAvailableTechnology* stereotyped as «*goal*») thus maximizing the usefulness, ie, the «*business_value*». With this aim, the evolution acts on a new *system_resource* by replacing the *CommandDisplay* with the smartphone. By eliciting the evolution as enabled with our profile, it is possible to help

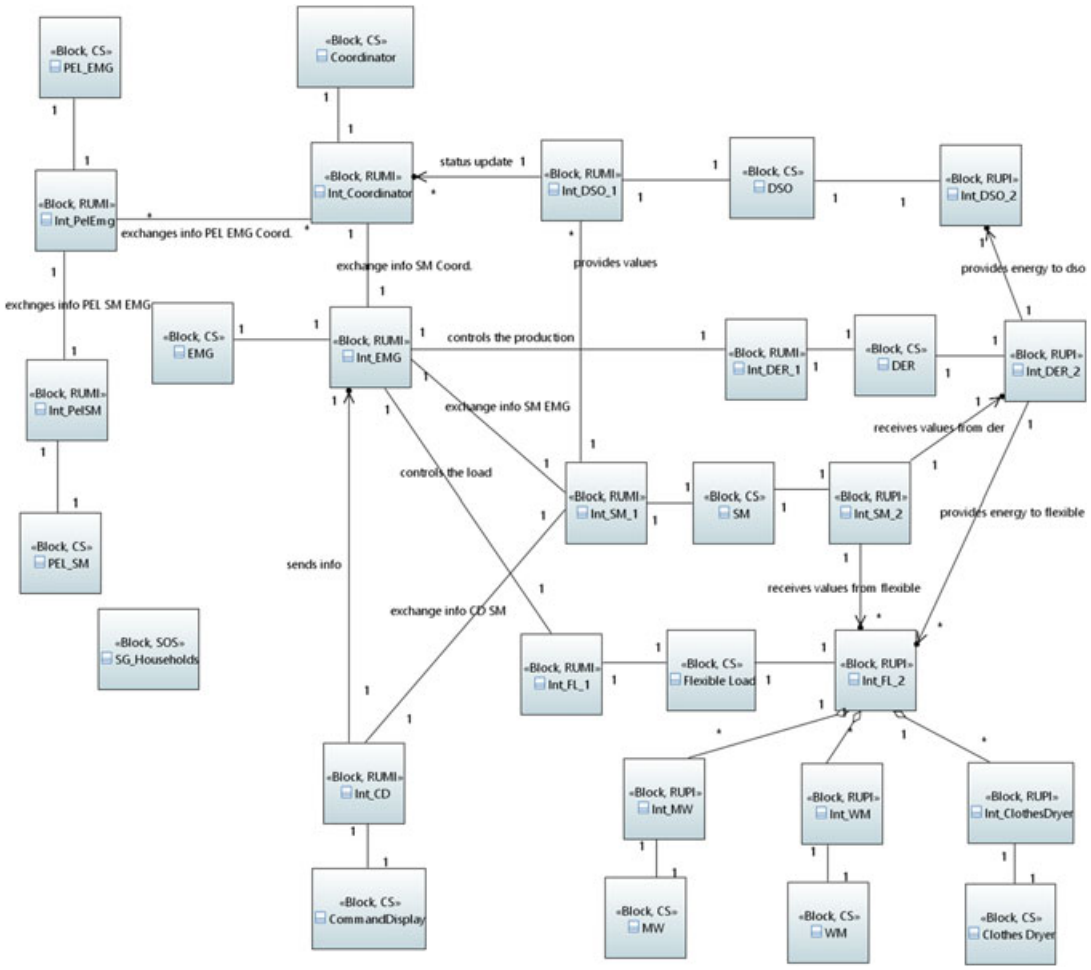

FIGURE 4 Energy management scenario: static structure. CS, constituent system; DER, distributed energy resource; DSO, distribution system operator; EMG, energy management gateway; MW, microwave; RUMI, relied upon message interface; RUPI, relied upon physical interface; SM, smart meter; WM, washing machine

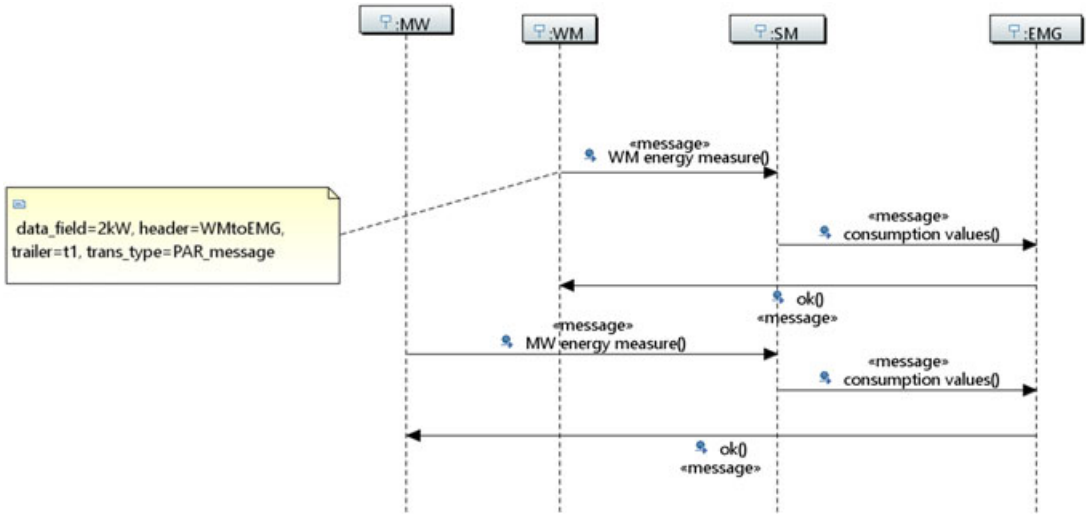

FIGURE 5 Energy management scenario: message exchange. EMG, energy management gateway; MW, microwave; SM, smart meter; WM, washing machine

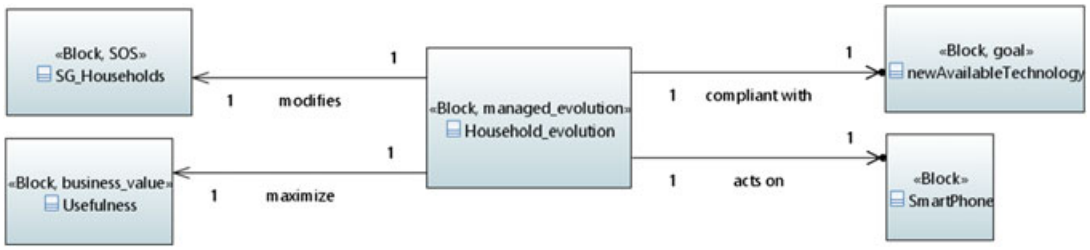

FIGURE 6 Energy management scenario: evolution modeling. SoS, systems of systems

system designers in reflecting on how a progressive change or development could impact the whole SoS.

5.3 | Dynamicity viewpoint

As motivated by the nature of the dynamicity viewpoint (see Section 3), we present (1) how to apply specific profile concepts to elicit dynamicity and (2) how to represent dynamic interactions occurring in an operational SoS.

First, by applying a specific BDD defined in our profile, we show how to identify the dynamic behavior in the energy management scenario which consists in the *connection\disconnection* of the *Flexible Load*. This dynamic behavior is stereotyped as «*reconfigurability*», ie, the variation to the CSs structure (see Figure 7).

Second, since our objective is to fully capture the dynamic behavior of an operational SoS, we propose a 3-step process: the first step consists in selecting the CSs involved in the communication; the second, making use of structure viewpoint (through a sequence diagram), represents the behavior of messages exchanged among CSs; the third step is to analyze the most common interactions. The dynamic behavior supporting the provision of energy (see Section 2) is represented in Figure 8 as a sequence diagram showing continuous reconfiguration of the grid performed by the household electrical appliances.

The latter want to be switched on at a time from t1 to t7, with the support of the coordinator entity, which satisfies energy requests according to the current global consumption and production values. The figure shows that if the current energy load can become unbalanced by allowing the clothes dryer to be switched on (as requested at time t5), the coordinator may decide to prohibit the clothes dryer to be connected at time t6 and t7. In the figure, we have highlighted the only electrical appliance which cannot be switched on according to the message received by the gateway EMG.

Our dynamicity representation supports a system designer in understanding which are the properties of an SoS that are constantly changing and how the SoS may change by rearranging its components.

FIGURE 7 Energy management scenario: dynamicity definition. CS, constituent system; SoS, systems of systems

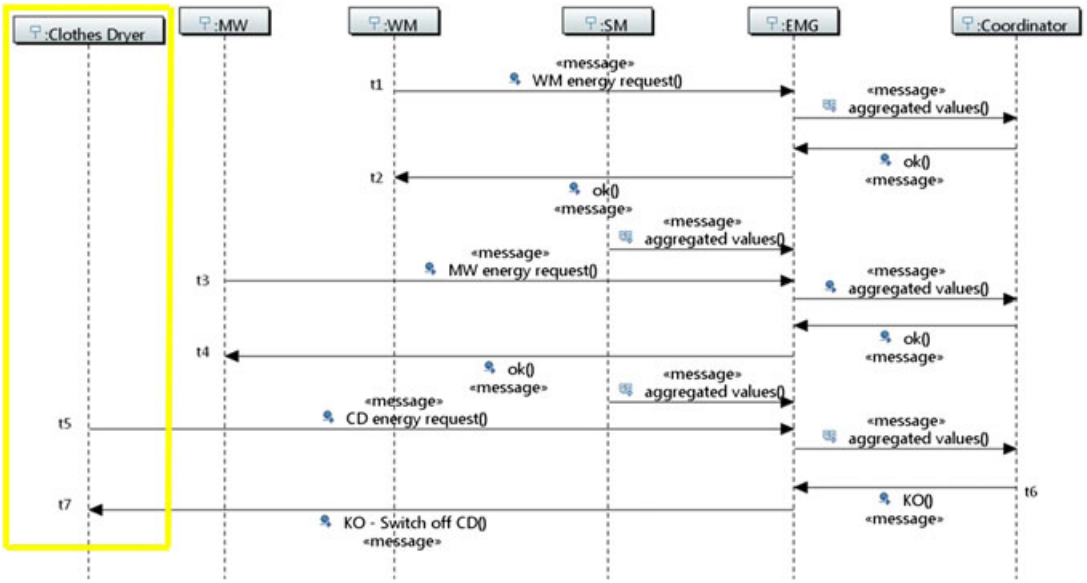

FIGURE 8 Energy management scenario: dynamicity behavior description. EMG, energy management gateway; MW, microwave;WM, washing machine

This dynamic introduction, modification, or removal of CSs can introduce new system behaviors that need to be analyzed.

5.4 | Emergence viewpoint

Figure 9 shows how, through a specific BDD, the emergence phenomenon is described in Section 2.

The latter is labeled as wrong coordination decision and it is represented as an *«explained_emerg_phenomenon»* explained by the coordinator balancing behavior which is stereotyped as a *«trans_Ordinal_law»*, ie, a law explaining what happen globally through CSs interactions in order to balance consumption and production values. This phenomenon results in a blackout for the smart grid which consists in an *«unexpected&detrimental»* behavior putting the grid out of service.

Similar to dynamicity, we cannot model the manifestation of emergent phenomena based on a static description of interacting CSs alone. Consequently, we also introduce a sequence diagram as it has been made available to us in the structure viewpoint. We use this type of diagram to detail the interactions among CSs that lead to a blackout, which we consider as a detrimental emergent phenomenon in the smart grid household scenario. Consider the sequence diagram in Figure 10, where WM is switched on at t1 after the agreement allowed from the coordinator. Next, the coordinator receives (at time t2) and grants (at time t3) the request for switching on the public lighting for the exceptional event. This request is forwarded to the coordinator from the PEL EMG which is external to the household. Before the request is issued by the EMG (at time t2), the SM fails in communicating the actual production/consumption values. This will affect the forthcoming decisions of the coordinator which will act upon wrong information. As next, within the household, the MW and clothes dryer issue (at time t4 and t5) and receive (at time t6 and t7) the grant to be connected to the grid. These decisions are based on the wrong information received by the EMG: the EMG assumes to grant energy requests while still being able to balance consumption and production values. Unfortunately, because the external SM has communicated no information, the grid is accepting more requests for energy than it is able to supply. This puts energy producers under stress and eventually leads to the failure of some energy producers. These failures lead to an even greater imbalance in the grid. The resulting higher stress on remaining energy producers will trigger more failures until finally the grid is not able to deliver enough energy for most consumers, ie, a blackout has occurred. The latter is revealed within the household by the switch-off messages sent from the SM to all the appliances (time t8-t10).

This illustrative example shows that networked individual systems, which work together to realize a higher goal (optimal energy distribution), could lead to a detrimental emergent behavior in the event of an exceptional energy demand in combination with a catastrophic SM failure. This has been modeled (except for the producers interactions) through the exchange of messages leading to

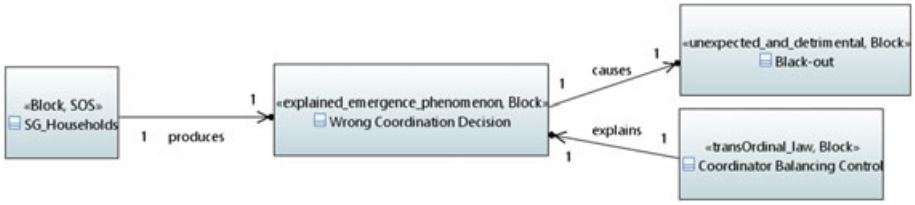

FIGURE 9 Energy management scenario: emergence definition. SoS, systems of systems

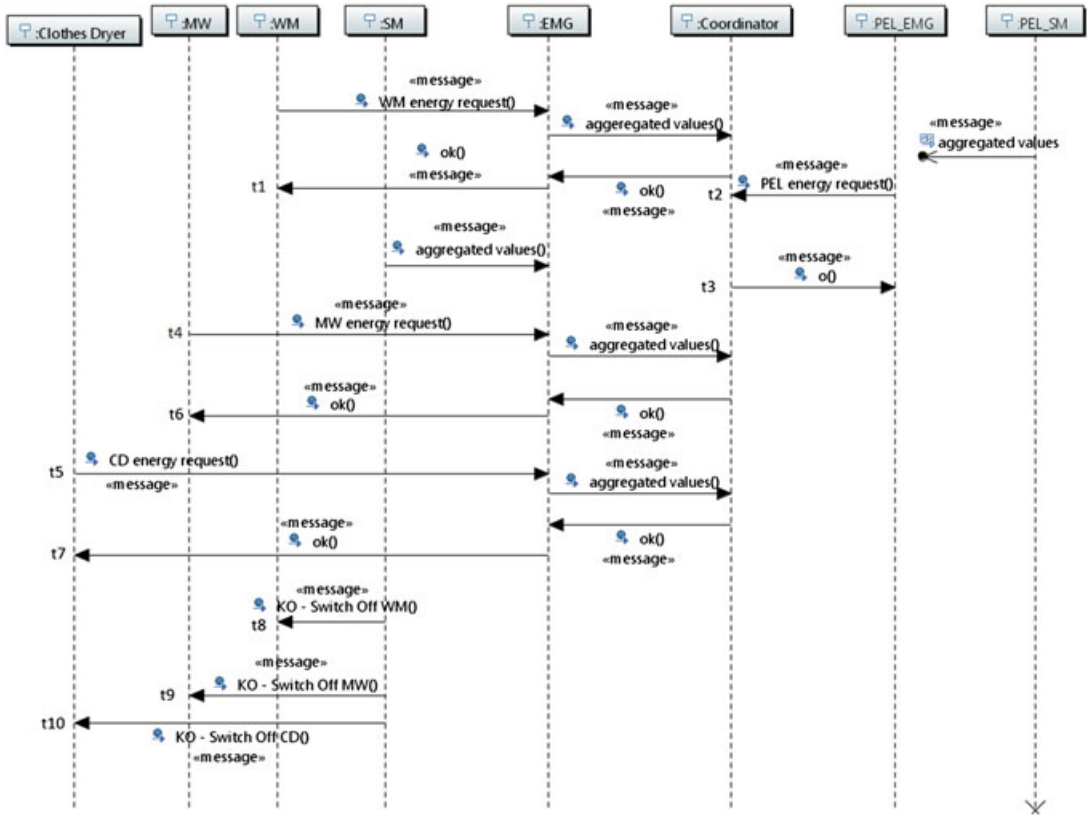

FIGURE 10 Energy management scenario: emergence behavior description. EMG, energy management gateway; MW, microwave; PEL, public event lighting; SM, smart meter; WM, washing machine

unexpected and detrimental emergent behavior caused by a system dynamicity property.

5.5 | Dependability viewpoint

As it emerges from the motivating scenario different dependability requirements have to be accurately taken into account for the household management SoS. In particular, we have discussed the dependability metrics related to safety and availability of the SM. To this end, the profile supported the definition of availability and safety *«dependability guarantee»* (see Figure 11). The first consists in providing an availability > 10^{-3} by means of a replication technique to provide *«fault-tolerance»*. The second consists in providing a THR per hour THR < 10^{-8} through an error correction technique providing *«fault-tolerance»*. The 2 adopted techniques are implemented within the SM CS; thus, in the diagram, it also appears the SM block as imported from the structure viewpoint.

5.6 | Security viewpoint

The security viewpoint supported the definition of secured communication through the *«security»* stereotype as linked to the *SG_Household* (see Figure 12).

The communication to be secured occurs between EMG and SM CSs for which the protocol adopted is the Open Smart Grid Protocol as defined with the *«cryptography»* stereotype. This protocol adopted as encryption and decryption algorithms the RC4 and EN14908 algorithms. The latter are represented by means of 2 different blocks each labeled with *«encryption» and «decryption»* stereotype. This adopted mechanism is based on a secret key infrastructure which exploits the

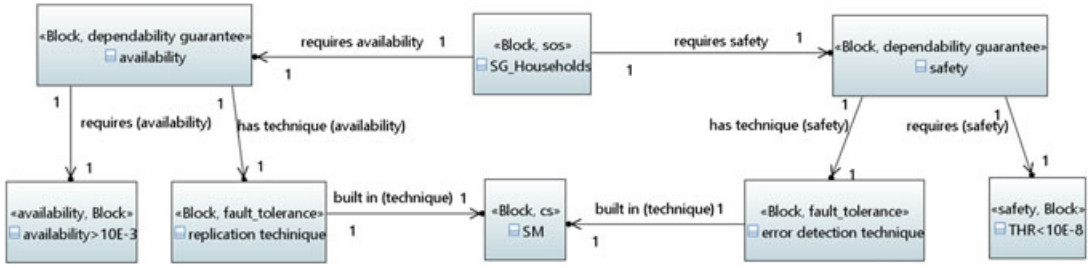

FIGURE 11 Energy management scenario: dependability. CS, constituent system; SM, smart meter; SoS, systems of systems; THR, tolerable hazard rate

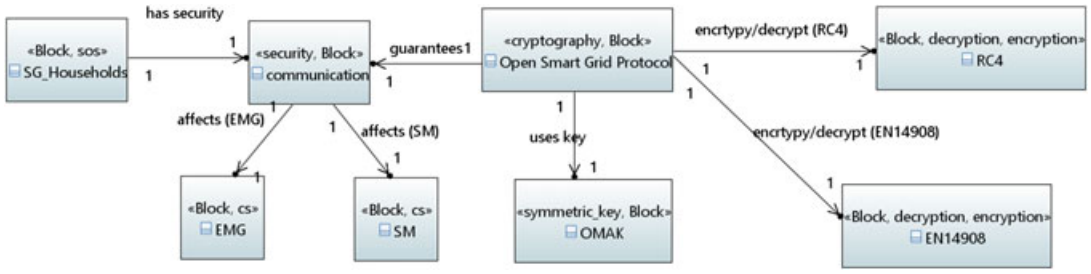

FIGURE 12 Energy management scenario: security. EMG, energy management gateway; SoS, systems of systems; SM, smart meter

so-called *OMAK «symmetric-key»* representing the shared secret between the EMG and the SM.

5.7 | Multicriticality viewpoint

As discussed in the energy management scenario, we have a PEL service with a high level of criticality and a washing service with a lower level of criticality. To represent this situation in Figure 13, we have added the stereotype *critical_service* to both the former services, and we have linked each of them to the correspondent *criticality_level* either low or high (in the specific example). Each of these levels is defined by means of different safety requirements. The high *criticality_level* (associated to the PEL service) consists in a THR per hour smaller than 10^{-8} (corresponding to Safety Integrity[28] Level 4) while the low criticality level (associated to the washing service) consists in no THR set. In this case, we have considered stereotypes identified for structure and dependability viewpoints which are essential to characterize the multicriticality instantiation.

6 | ADOPTING THE SoS PROFILE WITHIN MDE METHODOLOGIES

The profile that we illustrated by applying it to the energy scenario in Section 2 can also be adopted within a MDE approach,[29] which supports system understanding, design, development, maintenance, and evolution by means of models. In this context, by applying our profile it is possible to obtain a platform-independent model (PIM), ie, a viewpoint-driven SoS model describing the architecture and its behavior which neglects platform specific details. A PIM can further be combined with specific platform details to generate a platform-specific model for each viewpoint. Our profile supports the generation of a PIM as the first step for a model-driven methodology. Even though generic and platform independent, a PIM is the starting point for a set of specific tasks. Among others, source code generation may automatically support the translation of the SoS to executable artifacts. System analysis techniques, like hazard analysis (HA), failure mode and effect analysis, and fault tree analysis may be also applied to different purposes (eg, Bonfiglio[30]). Finally, also system testing may also be applied to identify test procedures or to resolve problems of testing coverage. Noteworthy, the mentioned techniques to be valuable completed require additional inputs.

In the rest of this section, we show the applicability and usefulness of the SoS profile in 2 different contexts:

- In Section 6.1, we discuss how the PIM generated with our profile can be used to support the detection/avoidance of emergent behaviors of an SoS by means of an HA.
- In Section 6.2, we illustrate the usage and integration of the profile in a MDE tool to model, validate, query, and simulate SoS.

6.1 | Interface analysis to detect emergent behaviors of SoS

This section describes how the PIM generated with our profile can be exploited to support the detection/avoidance of emergent behaviors of an SoS by means of an HA. The basic idea consists in analyzing interacting events among CSs. With this aim, we have defined a set of steps to be followed. Step 1 defines an SoS architectural model using the elements of the profile; step 2 formalizes the connections among CSs to univocally identify internal SoS interfaces; step 3 identifies a set of events that could lead to emergent behaviors, being it

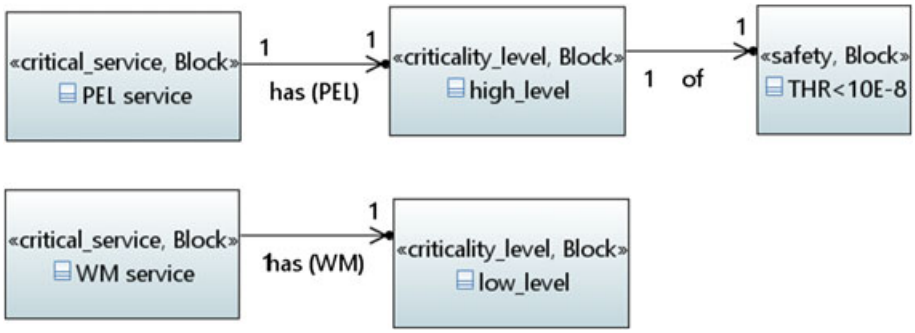

FIGURE 13 Energy management scenario: multicriticality. PEL, public event lighting; THR, tolerable hazard rate; WM, washing machine

TABLE 1 Energy management scenario: internal interfaces

Interface among constituent systems	ID Interface
EMG and coordinator	INT_01
Coordinator and DSO	INT_02
Smart meter and meter aggregator	INT_03
Meter aggregator and DSO	INT_04
Command display and EMG	INT_05
Command display and smart meter	INT_06
Smart meter and flexible load	INT_07
Smart meter and EMG	INT_08
EMG and flexible load	INT_09
PEL SM and PEL EMG	INT_10
PEL EMG and coordinator	INT_11

Abbreviations: DSO, distribution system operator; EMG, energy management gateway; PEL, public event lighting.

either beneficial or detrimental; step 4 associates each event with semantic information, ie, guidewords to explore particular circumstances leading to an emergent behavior.

To illustrate how it is possible to detect emergent behaviors, we exploit the energy management scenario (see Section 2). According to step 1, we already discussed the SoS model for the smart grid (see Figure 4). According to step 2, we univocally identified CS interfaces in the former model as listed in Table 1 (step 2).

Following step 3, we have identified 2 events that could originate emergent behaviors.

(Event 1) A new functionality is added to the command display: the electrical appliances can be switched on and off through the command display HMI. In such a case, the command display sends a message to EMG containing the name/type of an electrical appliance involved. The EMG can distinguish the device that requires/release energy and forward the request/notification to the coordinator.

(Event 2) A new EMG is connected to the Smart Grid to support the provision of energy for PEL. This represents the most difficult step because it is strictly related to the SoS characteristics.

Once we have identified relevant events, it is possible to start with a system interface analysis (step 4). To this end, we exploit the interface HA technique, as defined in Redmond et al,[31] which identifies and mitigate hazards leading to detrimental situations. Our aim is to adopt the HA technique with a different objective, ie, finding emergent conditions (positive and negative) related to the information exchanged through the interfaces. Our hazard-based analysis takes as input the identified events (step 3) and the internal interfaces (step 2), and it produces as output the identification of possible consequences and emergent behaviors.

An extract of an interface analysis is shown in Table 2. Each row represents an event with an associated guideword[5] which is exploited to help designers in detecting hazardous events. The last 2 rows identify 2 types of emergent behaviors. The former represent a beneficial emergent behavior caused by the new functionality of the command display: EMG receives additional information on the electrical appliance willing to be switched on. In this case, EMG can forward additional information to the coordinator. For instance, the type of electrical appliance could be communicated to the coordinator, which may optimize the energy consumption accordingly (ie, knowing that the energy requested by a WM lasts longer than the energy requested by a MW). The last row of the table represents a detrimental emergent behavior caused by failed communication between the PEL SM and the PEL EMG (*INT_10*) which in turn transmits wrong aggregated consumption values to the coordinator (*INT_11*). The latter will then make possibly wrong decisions on granting the provision of energy thus causing a blackout. Mitigation to this case may consist of implementing a distributed failure detector mechanism to guarantee that at *INT_11* information are correctly exchanged and then the coordinator will act on the correct basis.

[5]Guideword (in our example, *not*, *more*, and *early/late*)[32] adds semantic information to the event, ie, *not* indicates the nonoccurrence of an event, *more* indicates the occurring or something additional, *early/late* means an early/late occurrence of an event; in our specific case, guidewords exploited to identify both hazards and emergence behaviors.

6.2 | SoS profile's integration in a MDE design framework

Adopting the presented profile in a model driven engineering process may be difficult for nonexpert designers in SysML modeling; additionally, scalability concerns may arise with the growing complexity of the SoS. In our perspective, we have considered 7 different viewpoints to break the complexity of design models. But still, having a detailed definition of components and their interactions and attributes adds complexity to the modeling phase. Indeed, even with a small case example the readability of the SoS design using SysML can be already difficult, finally leading to the so-called spaghetti diagrams where the model is composed of a huge number of crossing lines connecting the different blocks. To solve the above problems, it is necessary to adopt graphical approaches which have to be easily integrated in the MDE chain while still being usable also with large scale SoS scenarios. In addition, SoS designers should be able to conceive an SoS design without having specific expertise of modeling technologies like SysML. To this end, a new graphical tool shall be adopted to provide efficient design facilities and the compatibility with the MDE chain.

In the rest of this section, we will present a supporting facility tool developed within the AMADEOS project[16] that (1) integrates the SoS profile for enabling the design of SoS, and (2) provides a simple and intuitive design environment extending the modeling capabilities of Google Blockly.[33]

6.3 | AMADEOS supporting facility tool

The objective of the AMADEOS supporting facility tool is to provide a simple and intuitive way to design SoS combining the SoS profile and the Blockly tool. Blockly[33] is a domain specific language adopted to ease the design of SoS by means of simpler and intuitive user interface thus requiring minimal technology expertise and support for the SoS designer. Blockly is a Google open source project under the Apache 2.0 license which consists in a client-side JavaScript library for creating visual blocks programming editors. Its user interface consists of a toolbox entailing all the available blocks and a workspace where the former blocks can be placed. Blockly can be also adopted to generate

TABLE 2 Extract of the smart grid interface analysis

Id Event	Id Interface	Guideword	Hazard	Emergent Behavior	Consequence	Mitigation
Event 1	INT_05	Not	EMG does not receive information from the new command display	No	EMG cannot forward the request/notification to the coordinator	MIT_01: command display has to wait the acknowledgment from EMG
Event 1	INT_05	Not/more	EMG receives wrong information from the new command display	No	EMG makes a mistake in requesting/releasing energy	MIT_02: EMG has to verify the information sent from command display
Event 1	INT_05	Late	MG receives information from the new command display with a delay	No	EMG cannot control flexible load and does not optimize the energy consumption	MIT_01: command display has to wait the acknowledgment from EMG within a fixed interval
Event 1	INT_05	More	EMG receives additional information from the new command display on the electrical appliance switched on	Yes–beneficial	EMG can forward additional information to the coordinator for better balancing the smart grid	Not needed
Event 2	INT_10 INT_11	Not	The PEL EMG do not receive aggregated consumption values from the PEL SM	Yes–detrimental	PEL EMG transmits wrong information to the coordinator which grants more request then it is possible to satisfy (blackout)	MIT_03: distributed failure detector through "I am alive message" sent periodically at interface INT_10 between PEL SM and PEL EMG

Abbreviations: EMG, energy management gateway; PEL, public event lighting; SM, smart meter.

JavaScript, Python, PHP, or Dart code and can be also customized to generate code in any computer language. It has been adopted in different contexts of usage to support the definition of educational games and teaching programming languages concepts.

The AMADEOS supporting facility[6] is a tool to model, validate, query, and simulate SoS. The tool integrates the SysML profile described in this paper thus importing all the terminology and relationships available in the SoS profile. Therefore, all the benefit achievable through the profile can be also achieved through the AMADEOS supporting facility tool, namely, elevating the design from a system to an SoS perspective and decreasing its cognitive complexity.

The editor, based on Blockly, eases the design of SoS by means of intuitive and usable interfaces which do not require any specific SysML technologies expertise, and technological support. Blocks, which represent specific profile stereotypes, can be easily created and deleted by means of drag and drop facilities. As an example, Figure 14 shows the supporting facility tool homepage with the Blockly model—architecture viewpoint—corresponding to the smart grid scenario described in Section 2. At this level of abstraction, we can define the targeted SoS (block "Household") as an autonomous system (as discussed in Section Section 4.1.1) composed by 7 CSs (the blocks "Coordinator", "Household DER", etc).

Scalability problems can be avoided by means of a multilayered view: a viewpoint-based perspective that enables the visualization of portions of artifacts referring to a certain selected viewpoint. By simply selecting the viewpoint of interest (depicted in the left part of Figure 14), the editor shows the relevant portions of artifacts, and it hides the rest of the design models. By simply clicking each block, it is possible to expand it and to navigate through its entailed properties and subelements thus facilitating the switching among SoS views of different granularity.

The advantages in terms of usability and readability of design models have been achieved by defining proper transformations within the MDE chain. The flow of MDE using the tool is described in Figure 15.

First, the SysML meta-model is transformed to Blockly blocks, and these blocks can be used to create an SoS model as depicted in Figure 14. The supporting facility provides rapid modeling, validating, code generation, and simulation facilities to the user. It is an iterative design process, where early design SoS models can be successively refined and extended to account for new information available on the targeted system (eg, new knowledge or insights on some CS) or as a consequence of the analysis of the simulation results (eg, detecting a violation of a safety requirement). Finally, once the model is complete in Blockly, it can be transformed back to SysML in Eclipse for further refinement or formal analysis.

The supporting facility tool can generate 3 outputs: (1) the model in XML, (2) Python code generated for the simulation, (3) a plantUML[7] representation of the equivalent SysML model. The Python code generated by the tool can be further refined and also can be used to

[6]The current version of tool can be accessed at http://blockly4sos.resiltech.com.

[7]PlantUML is an open-source tool allowing users to create UML diagrams from a plain text language. URL: http://plantuml.com/.

connect to other simulators or external systems for interaction while running simulation.

The main features of the AMADEOS supporting facility tool are described as follows:

- Requirements management. The design of an SoS starts with requirements; hence, requirements management is an important aspect of SoS design, where traceability of requirements must be maintained and monitored. Requirements are divided based on the viewpoints. Each block maintains the list of requirements it meets and each requirement block maintains the list of blocks which satisfy its requirement thus offering traceability.
- Design validation and constraints. Supporting facility allows only compatible blocks to be connected with each other thus making the model valid by design. Custom constraints can be specified to make the model precise; the constraints are specified by the designer in JavaScript and the tool uses a specific function to evaluate the constraint statements and change the color of a block to black if the constraints are not satisfied. The constraints are evaluated at each "'onchange' event of the block. The tool also supports constraints during the simulation. These are supported using Python's assert function and evaluated at runtime during simulation.
- Design without lines. To avoid the spaghetti diagram problem, Blockly uses collapsed views to simplify/abstract an SoS model and it does not use lines to show relationship between blocks. Model querying can be used to visualize a model in its traditional view (i.e. visualizing block relationships with lines).
- Model querying. On large models, it is also important to visualize models with a customized viewpoint. For example find all critical services having criticality less than 3 (ie, filter blocks based on a condition). The blocks in the model goes through a filter function in JavaScript and highlights only the blocks that passes through this filter. The filter function is created on the fly from the query entered by the user.
- Behavior, sequence diagrams, and simulation. Behaviors for each block in the model can be added in Python programming language which will be executed during simulation.

Scenarios can be simulated on the basis of the sequence diagrams created by the user. The sequence diagrams can be created using blocks provided in the supporting facility tool. Unlike traditional sequence diagrams, these blocks offer restricted sequence diagrams which adhere to AMADEOS concepts (eg, communication can only be performed through RUIs). These diagrams do not have any ambiguity and can be converted to code. The simulation starts by starting all the CS as threads; then for each system, RUIs are started and wait for communication. The sequence diagrams created by the tool can also be viewed in the traditional format.

6.4 | Lessons learned

The profile is the result of several activities carried out within the AMADEOS project, all aiming to properly define and model the key SoS elements (concepts and their relationships). The profile and the supporting modeling tool have been extensively discussed with AMADEOS members, with major representatives of important organizations in different SoS domains, including energy and smart grid, banking, transport, emergency and cloud, and with the scientific community at large through papers[17] and specific tutorial sessions. The tutorial session[18] organized as part of the INCOSE IS conference in Edinburgh was a formidable opportunity to have a concrete (external) feedback on the profile's definition and on the usefulness of the AMADEOS supporting facilities tool for designing SoS. The tutorial session (6 hours) involved a total of 15 participants both from Industries and Universities, and it had the following objectives: (1) to demonstrate the clarification effects and usefulness of the key elements of the conceptual model, (2) to illustrate the usefulness of adopting the SysML profile for the high-level design of an SoS architecture, and (3) to show the potentialities of the supporting Google Blockly tool for the integrated design of the different SoS viewpoints.

FIGURE 14 The Blockly smart grid model (architecture viewpoint). DER, distributed energy resource; EMG, energy management gateway; UML, unified modeling language

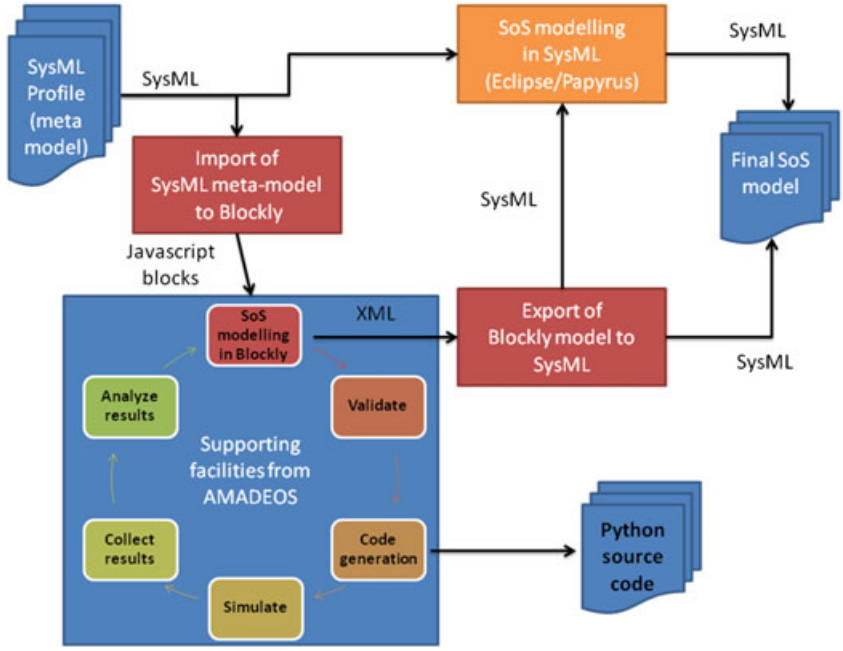

FIGURE 15 Model-driven engineering flow with the supporting facility tool in Blockly4SoS. SoS, systems of systems; SysML, systems modeling language

In general, we got a very positive feedback on the usefulness of the approach for constructing a platform independent description of SoS, which allows to capture the key, cross-domain, SoS concepts considering the different (interconnected) viewpoints. At the same time, it was noted that the profile and the supporting tool should now move towards specific application domains thus extending it with domain-specific artifacts for modeling an SoS in specific application areas. Another important suggestion was to focus on the interoperability and seamless integration of the supporting facilities tool into existing development environments (eg, in sysML ecosystem), paving the ground for its future adoption in real industrial contexts.

A final, comprehensive, assessment of the major AMADEOS project achievements, including the SoS profile and the supporting facility tool, has been presented in a public project deliverable.[34] Results confirm the adequacy of the SysML profile (and therefore of the corresponding conceptual model) for expressing the key SoS elements in the different viewpoints, and underline the very positive Industrial experience of using the supporting facility tool for SoS design, noting that similar design done in SysML without viewpoints was complex and difficult to manage due to too many lines in the diagram. The capability to simulate the behavior of the target SoS was another selling point of the approach, allowing system architects to quickly test hypothesis regarding future systems and determine what attributes will lead to advantageous or poor results.

7 | RELATED WORKS

Modeling of systems, or specific aspects of systems, has been performed for many years in many different disciplines; several models can be developed at different abstraction levels, depending on their purpose and the forms of analysis that are to be performed on them. For example, the MARTE[24] profile focuses on real-time systems, the CHESS[35] profile on industrial systems, the CONCERTO[36] profile on embedded systems, and the UMLsec[25] profile is instead specific for introducing security aspects.

While the foundations of these profiles constitute a solid reference and background, they are generally not able to represent the many facets of SoS. Modeling SoS requires capturing the different facets of SoS; however SoSE and SoS modeling, as reported in Nielsen et al.,[37] is a relatively novel area of research, where established modeling solutions are still missing. While different modeling approaches have been proposed in the last years, several research questions are still open. These are due to the complex and variegate nature of SoS, and the inherent difficulty in identifying a generic and comprehensive way of describing them.

In this section, we present a viewpoint-driven analysis of related ADL design approaches presented in the literature of SoS. This analysis, whose results are reported in Table 3, is not meant to be exhaustive but it is based on the most representative related works on designing

TABLE 3 Viewpoint-based literature analysis of SysML approaches to SoS design

	Structure	Dynamicity	Evolution	Emergence	Time	Multicriticality	Dep & Sec
Huynh et al[38]	✓	✗	✗	✗	✗	✓	✗
Lane et al[39]	✓	✗	✓	✗	✗	✗	✗
Rao et al[40]	✓	✓	✓	✗	✗	✗	✗
COMPASS[41-43]	✓	✗	✗	✓	✓	✗	✓
DANSE[44,45]	✓	✓	✓	✓	✓	✗	✓

Abbreviations: SoS, systems of systems; SysML, systems modeling language.

SoS. Its objective is to determine at what extent viewpoints-based SoS concepts have been already captured in the literature.

In Huynh and Osmundson,[38] the authors propose the use of SysML in representing an SoS by adopting and in some cases extending canonical SysML diagrams to model different viewpoints of an SoS. In particular aspects related to the structure viewpoint have been deeply considered to identify the SoS internal structure, its boundaries with the environment through well-defined interfaces, SoS functionalities, and how interactions occur by exchanging messages. Beyond structure, a specific support to the multicriticality viewpoint is also provided by adopting the specific stereotypes aiming at grouping requirements according to qualitative and quantities metrics to support trade-off analysis. Nevertheless, in Huynh and Osmundson[38] a specific support to the time viewpoint has not been considered to assure the responsiveness of SoS and dependability/security viewpoints have not specifically addressed. The authors did not consider viewpoints like dynamicity, evolution, and emergence.

A partial answer to the above issues is given by the approach presented in Lane and Bohn[39] providing support to structure and evolution viewpoints of an SoS by exploiting several SysML models. The authors propose the adoption of diagrams to determine an evolving SoS and its environment and the interactions occurring between an SoS and the environment and among CSs themselves. Noteworthy, the approach is still missing specific support to dynamicity, emergence, and multicriticality viewpoints and although the executable models presented are a first required step to assure dependability/security requirements and responsiveness (time), it still missing a specific support to those viewpoints.

In the SysML modeling approach presented in Rao et al,[40] the authors allow the definition of the SoS structure and how to support dynamicity and evolution viewpoints by means of understanding the dis-alignment of a simulated SoS with respect to its requirements. The approach makes use of different executable diagrams to simulate Net-centric SoS through the Petri net formalism thus describing the dynamic behavior and assuring that end-user requirements are met. Noteworthy, the approach[40] is still missing a specific support to emergence and multi-criticality. Concerning dependability and security viewpoints as well as responsiveness (time), the approach,[40] similarly to Lane and Bohn,[39] does not provide any specific support.

The approach presented in Bryans et al[41] and Ingram et al,[42] in the context of COMPASS EU project,[43] provides support to model the structure of an SoS and emergence by means of the extension to SysML diagrams. Analyses of the former models are conducted to evidence that requirements are fulfilled. COMPASS exploits tool's well-established extension mechanisms to extend traditional systems modeling as needed to model and analyze SoS with the support of the formal COMPASS modeling language. The latter has been exploited to support fault handling (dependability viewpoint) and responsiveness (time viewpoint) of an SoS. Nevertheless, the approaches in Bryans et al[41] and Ingram et al[42] provide no specific support to dynamicity, evolution, and multicriticality.

The approach in Gezgin et al,[44] within the context of the DANSE EU project,[45] supports the definition of an SoS structure, dynamicity, and evolution (by means of Graph Grammars), emergence, etc, with the only exception of multicriticality. DANSE presented a set of methodologies and tools to model and to analyze SoS based on the unified profile for DoDAF and MoDAF. In particular, DANSE focuses on the 6 models that can be represented as executable forms of SysML as partially reported in Gezgin et al,[44] according to a well-defined formalism to relate basics SoS concepts and their relationships. In the context of DANSE, the goal specification contract language assures the achievement of dependability and security requirements and it guarantees the timely response of an SoS.

The following works are referred here because they focus on different viewpoints in SoS modeling, but they are not in the Table 3 as they do not provide a profile. The approach in Baldwin et al[46] presents a theoretical model to describe autonomy, belonging, connectivity, diversity, and emergence in SoS. The work first models these characteristics and their properties, then it uses a computer simulation to demonstrate the presented model. SoSADL,[47] currently under development, is a formal language derived from π-ADL and targeted to SoS architectures. SoSADL includes static and dynamic architectural specifications, with specific focus on reconfiguration modeling.

As shown in the literature, different attempts exist to apply SysML approaches to specific viewpoints that we deemed essential in supporting the design of SoS. These approaches have shown the utility of adopting SysML formalisms to model architectural aspects of SoS thus supporting different types of analysis and a first step towards executable artifacts which can be automatically derived. Although these approaches provide detailed insights for different viewpoints aspects, it is still missing (1) an homogeneous synthesis at a more abstract level of key design-related SoS concepts and (2) and a viewpoint-based vision. Bringing this perspective in one single consistent reference model, it is possible to provide solutions to specific design problems while still keeping the required interconnections among viewpoints.

8 | CONCLUSION AND FUTURE WORK

This paper presented a viewpoint-driven approach to design SoS by adopting a SysML profile. We pointed out the gaps in the literature of ADLs for SoS with regards to a set of viewpoints that we deemed essential for understanding SoS. We outlined the conceptual model at the basis of the profile, and we presented how to solve specific viewpoint needs in an integrated fashion by exploiting the high-level SoS representation in a small scale scenario. We implemented the profile in the Eclipse-integrated development environment jointly with Papyrus,[48] ie, an Eclipse plug-in supporting advanced facilities to manipulating UML artifacts and SysML profiling. We discussed the integration of the profile in the MDE chain and the support for different type of SoS analyses. As an example, we showed how the HA may support the detection of emergent behavior. Furthermore, we discussed the usage of the profile with a MDE tool to support the SoS designer in managing the complexity of SoS models. To such extent, scalability and usability concerns have been discussed.

As future work, we are applying our profile and the Blockly tool to a use case on industrial automation. More in detail, we are currently supporting the design of an automated industrial system, which includes the coordination of autonomous carts for loading and unloading goods, with the ultimate target of resource optimization

and of course without compromising safety of personnel and equipment. This case study focuses strongly on, among other, real-time requirements, safety requirements, and planning for positive emergence. We plan to apply the solutions presented in this paper to support the design and the simulation of the targeted SoS.

We also envision the application of the profile and the Blockly-based tool to support the different stages of an SoS life cycle. In fact, as a longer-term objective, we plan to study the application of our solution for verification and validation activities. Concerning verification, the approach is suitable for system analysis: the paper presented an approach for HA, but other techniques can be investigated as well, for example, failure mode and effect analysis and fault tree analysis. Concerning validation, the SoS model can be the basic layer to define test procedures or resolve problems of testing coverage. This is particularly relevant for SoS composed of several interacting CSs.

A further action, which is nontechnical but it is very relevant for us, is instead on the dissemination side. The profile and the Blockly-based tool are freely available, and it is our ambition to promote their usage. The Blockly-based tool has been submitted to the attention of Google (Blockly's owner), and as per today, it is reported on the Blockly Wikipedia website.[49] Further actions consist in finding novel use cases and pilots where the solution can be exercised, as well as further disseminate the results to encourage their usage and collect feedbacks from practitioners.

ACKNOWLEDGMENT

This work has been partially supported by the European Project FP7-ICT-2013-10-610535 AMADEOS.

REFERENCES

1. Jamshidi M. *Systems of Systems Engineering—Innovations for the 21st Century*. Wiley and Sons; 2009.
2. SAE Architecture Analysis & Design Language, AS5506, 2009-01-20.
3. "*Cyber-Physical Systems of Systems. Foundations – A Conceptual Model and Some Derivations: The AMADEOS Legacy*", A. Bondavalli et al. (eds.), LNCS 10099 (open access), 257 pages, https://doi.org/10.1007/978-3-319-47590-5
4. Ceccarelli A, Mori M, Lollini P, Bondavalli A. Introducing meta-requirements for describing system of systems. *HASE*. 2015;150-157.
5. Ceccarelli A, Bondavalli A, Froemel B, Hoeftberger O, and Kopetz H. Basic Concepts on Systems of Systems. In: *Cyber-Physical Systems of Systems* (pp. 1–39). Springer International Publishing, https://doi.org/10.1007/978-3-319-47590-5_1.
6. Nakagawa EY, Gonçalves M, Guessi M, Oliveira LB, Oquendo F. The state of the art and future perspectives in systems-of-systems software architectures. *SESoS*. 2013;13-20.
7. H. Kopetz. "Conceptual model for the information transfer in systems of systems", ISORC 2014, pp. 17-24, IEEE Press. 2014.
8. S. A. Selberg, M. A. Austin. "Toward an evolutionary system-of-systems architecture", INCOSE, pp. 1065-1078, 2008.
9. Schmerl B, Aldrich J, Garlan D, Kazman R, Yan H. Discovering architectures from running systems. *IEEE TSE*. 2006;32(7):454-466.
10. Murer S, Bonati B, Furrer FJ. *Managed Evolution: A Strategy for Very Large Information Systems*. Springer; 2010.
11. Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. *IEEE TDSC*. 2003;1(1):11-33.
12. Verissimo P. "Travelling through wormholes: a new look at distributed systems models," SIGACT News 37(1), pp. 66-81, 2006.
13. Kopetz H. *Real-time Systems: Design Principles for Distributed Embedded Applications*. Springer; 2011.
14. Mogul J. "Emergent (mis)behavior vs. complex software systems", EuroSys, pp. 293-304, ACM, 2006.
15. OMG. System modelling language (SysML) - http://www.omgsysml.org
16. FP7-ICT-2013-10-610535 AMADEOS—architecture for multi-criticality agile dependable evolutionary open system-of-systems - http://amadeos-project.eu/
17. M. Mori, A. Ceccarelli, P. Lollini, A. Bondavalli, B. Froemel. (2016). A holistic viewpoint-based SysML profile to design systems-of-systems. In 2016 IEEE 17th International Symposium on High Assurance Systems Engineering (HASE), pp. 276-283.
18. P. Lollini, and A. Babu. "Architecting systems of systems: from basic concepts towards a SoS profile and supporting tools", Tutorial session within INCOSE IS 2016, July 2016, Edinburgh. http://amadeos-project.eu/event/incose-international-symposium/
19. Jovanovic P, Neves S. Practical cryptanalysis of the Open Smart Grid Protocol. In: *Fast Software Encryption*. Berlin Heidelberg: Springer; 2015.
20. OMG, Systems modeling language (SYSML) specification, version 1.3 (June 2012). URL http://www.omg.org/spec/SysML/1.3/PDF
21. S. Friedenthal, A. Moore, R. Steiner. *OMG Systems Modeling Language Tutorial. September*, 2009. Available at: www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf
22. AMADEOS Consortium, "Deliverable 2.3—AMADEOS conceptual model - revised", 155 pages, 30 September 2016. Available at http://amadeos-project.eu/
23. H. Kopetz, B. Fromel. "Direct versus stigmergic information flow in systems-of-systems." System of Systems Engineering Conference (SoSE), 2015 10th. IEEE, 2015.
24. "A UML profile for MARTE: modeling and analysis of real-time embedded systems", OMG Document Number: ptc/2008-06-09.
25. Jan Jürjens. "UMLsec: extending UML for secure systems development." International Conference on The Unified Modeling Language. Springer Berlin Heidelberg, 2002.
26. Alan Burns, Davis Robert, "*Mixed Criticality Systems—A review*." Department of Computer Science, University of York, Tech. Rep (2013).
27. H. Kopetz, O. Höftberger, B. Frömel, F. Brancati, A. Bondavalli. "Towards an understanding of emergence in systems-of-systems" SoSE, pp. 214-219, IEEE, 2015.
28. EN 50129:2003: railway applications—communication, signalling and processing systems—safety related electronic systems for signaling.
29. Schmidt DC. Guest editor's introduction: model-driven engineering. *IEEE Computer*. 2006;39(2):25-31.
30. V. Bonfiglio, L. Montecchi, F. Rossi, P. Lollini, A. Pataricza, and A. Bondavalli. "Executable models to support automated software FMEA". HASE, pp. 189-196, IEEE, 2015.
31. P.J. Redmond, J.B. Michael, P.V. Shebalin,. "Interface hazard analysis for system of systems", SoSE, pp. 1-8, IEEE, 2008.
32. P. Tommaso, R. Esposito, P. Marmo, and A. Orazzo. "Hazard analysis of complex distributed railway systems", Proceedings of International Symposium on Reliable Distributed Systems (SRDS), pp. 283-292, 2003.
33. N. Fraser. Google Blockly—a visual programming editor. URL: https://code.google.com/archive/p/blockly/, accessed Sep. 2014.
34. Amadeos Consortium, D4.5 - Final results and lesson learned from proof of concepts, 2016. Available at http://amadeos-project.eu/
35. A. Cicchetti, F. Ciccozzi, S. Mazzini, S. Puri, M. Panunzio, A. Zovi, and T. Vardanega. (2012, September). CHESS: a model-driven engineering tool environment for aiding the development of complex industrial systems. In Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on (pp. 362-365). IEEE.

36. Silvia Mazzini. "The CONCERTO project: an open source methodology for designing, deploying, and operating reliable and safe CPS systems", ADA USER, 2015, 36.4: 264.
37. C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, J. Peleska. "Systems of systems engineering: basic concepts, model-based techniques, and research directions." ACM Computing Surveys (CSUR) 48.2 (2015): 18.
38. T. V. Huynh, J. S. Osmundson. "An integrated systems engineering methodology for analyzing systems of systems architectures", Asia-Pacific Systems Engineering Conference, Singapore, 2007.
39. Lane JA, Bohn TB. Using SysML modeling to understand and evolve systems of systems. *System Engineering*. 2013;16(1):87-98.
40. Rao M, Ramakrishnan S, Dagli C. Modeling and simulation of net centric system of systems using systems modeling language and colored petri-nets: a demonstration using the global earth observation system of systems. *Systems Engineering*. 2008;11(3):203-220.
41. Bryans J, Fitzgerald JS, Payne R, Kristensen K. *Maintaining Emergence in Systems of Systems Integration: A Contractual Approach using SysML*. INCOSE; 2014.
42. C. Ingram, J. Fitzgerald, J. Holt, N. Plat. "Integrating an upgraded constituent system in a system of systems: A SysML case study." INCOSE International Symposium. Vol. 25. No. 1. 2015.
43. COMPASS, "Guidelines for architectural modelling of SoS. Technical note number: D21.5a version: 1.0", September 2014 -http://www.compass-research.eu
44. Gezgin T, Etzien C, Henkler S, and A. Rettberg, "Towards a rigorous modeling formalism for systems of systems", ISORCW, pp.204-211, IEEE, 2012.
45. DANSE Consortium, DANSE methodology V2 - D_4.3 - https://www.danse-ip.eu
46. W. Clifton Baldwin, Brian Sauser. "Modeling the characteristics of system of systems." System of Systems Engineering, 2009. SoSE 2009. IEEE International Conference on. IEEE, 2009.
47. Franck Petitdemange, Isabelle Borne, and Jeremy Buisson. 2015. Approach based patterns for system-of-systems reconfiguration. In Proceedings of the third international workshop on software engineering for systems-of-systems (SESoS '15). IEEE press, Piscataway, NJ, USA, 19-22.
48. MDT/Papyrus. Eclipse model development tools (MDT). http://wiki.eclipse.org/MDT/Papyrus-Proposal
49. Blockly, https://en.wikipedia.org/wiki/Blockly

How to cite this article: Mori M, Ceccarelli A, Lollini P, Frömel B, Brancati F, Bondavalli A. Systems-of-systems modeling using a comprehensive viewpoint-based SysML profile. *J Softw Evol Proc*. 2017;e1878. http://doi.org/10.1002/smr.1878