Schneider and Trapp *Journal of Internet Services and Applications* (2018) 9:15
https://doi.org/10.1186/s13174-018-0084-5

Journal of Internet Services and Applications

RESEARCH Open Access

B-space: dynamic management and assurance of open systems of systems

Daniel Schneider* and Mario Trapp

Abstract

Connected cars, freely configurable operating rooms, or autonomous harvesting fleets: dynamically emerging open systems of systems will shape a new generation of systems opening up a vast potential for new kinds of applications. In light of the hard-to-predict structure and behavior of such systems, assuring their safety will require some disruptive changes of established safety paradigms. Combining current research results from different disciplines with industrial experience, this paper dares to think out of the box and look beyond the limits of traditional safety assurance. It structures upcoming challenges posed by the emergence of open systems of systems, tries to shift existing paradigms to meet those new challenges, and proposes an abstract conceptual framework building on comprehensive interlinked multi-concern runtime models for dynamically assuring the safety as well as other properties of open systems of systems. As there currently is no comprehensive realization of the framework, we discuss what kind of approaches could fit into which parts of the framework and exemplify this for the case of conditional safety certificates.

Keywords: System of systems, Safety, Models at runtime

1 Introduction

Open systems of systems harbor enormous potential for new kinds of applications and thus will have a tremendous impact on our economy and society. Such applications are presently envisioned or even already evolving in numerous embedded systems domains. One prominent example is the automotive domain, where interconnected, autonomously driving cars shall realize the vision of zero-accident, low-energy mobility in spite of the rapidly increasing traffic volume. Another example are tightly interconnected medical devices and healthcare systems intended to ensure the health of an aging society. Such open systems of systems will play an essential role for the economy and for society.

Individual (constituent) systems dynamically connect to each other in order to collectively provide value-adding superordinate functionality. Instead of hierarchical systems, this leads to heterarchical systems of systems. There is no hierarchy, but all systems have equivalent rights and have their own mission and objectives. Nonetheless, they need to collaborate as a collective. The structure and behavior of an open system of systems dynamically emerge at runtime, leading to very flexible and adaptive solutions.

On the other hand, however, adaptivity and flexibility lead to uncertainties since engineers can hardly anticipate the emerging structure and behavior of an open system of system. Moreover, there is no central integrator who assumes responsibility for the final safety assurance of the resulting system of systems. In consequence, safety assurance of open systems of systems could easily become a bottleneck impeding or even preventing the success of this promising new generation of embedded systems.

As a very simple example, assume a farmer connecting a round baler of manufacturer A to a tractor of manufacturer B. The manufacturers developed their products completely independent from each other. However, using a common communication standard, the baler can control the tractor (acceleration, steering, etc.). The manufacturers have assured their product's safety independent from each other, but there has been no hazard analysis and integrated safety assurance of the resulting system of systems. Today this issue is tackled by means of 1-to-1 consideration of concrete pairs of tractors and implements. However, considering the number of possible combinations of tractors and implements, it is impossible (or at least economically infeasible) to consider and certify each conceivable combination manually. Considering further that typical open systems of systems consist of a lot

* Correspondence: name.surname@iese.fraunhofer.de
Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

more than only two constituent systems, assuring the safety of open systems of systems becomes a very challenging task. To remediate this issue, it seems not sufficient to think in the direction of only slight improvements of established methods and techniques. Instead, we argue that it makes sense to explore completely new types of solutions that enable systems to assume certain aspects of safety assurance themselves and during runtime. John Rushby was one of the first researchers publishing about this idea [1]. We subsequently introduced the concept of conditional safety certificates (ConSerts) [2, 3] and now, in the DEIS project, the concept of digital dependability identities (DDI) [4]. ConSerts, as well as DDI, are approaches for the definition of modular runtime safety or dependability models, which enable the dynamic evaluation of corresponding properties across system compositions.

In essence, this means that we shift parts of the safety engineering considerations and activities from development time into runtime. We therefore need to empower systems to assume these new tasks, which, in turn, requires corresponding knowledge as well as mechanisms operating on that knowledge. A general scheme that we deem a promising starting point is the concept of models at runtime [5], which can be utilized to dynamically manage functional properties as well as non-functional properties and assurances [6].

In this article, we introduce B-Spaces as a conceptual framework for comprehensive multi-concern and multi-level models at runtime. Even though the concept is generic and meant to interlink models for all relevant concerns and scopes to enable holistic dynamic management at the system, system-of-systems, and smart ecosystem level, our primary research focus is on assurances, and on safety assurance in particular. Therefore, we start by outlining important background information with respect to safety engineering and correlated challenges in next-generation systems. We then go on to briefly reiterate the concept of models at runtime before introducing the B-Space concept. Subsequently, we elaborate on safety assurance based on B-Space. We then illustrate how our previously published conditional safety certification approach would fit into the overall concept of B-Spaces as a first step towards implementing the concept.

2 Background

2.1 Safety Assurance in a Nutshell

Before we analyze safety assurance challenges for open systems of systems, a basic, common understanding of safety assurance in general is provided in the following.

The precise definition of safety assurance, and particularly of the terms used, depends on the respective application domain. The principal idea, however, is similar across all safety-related application domains. For the sake of simplicity, we therefore use the terms as defined in ISO 26262 [7], which is the relevant safety standard for automotive systems. It is at the same time one of the most recent safety standards.

The overall goal of safety engineering is to ensure 'freedom from unacceptable risk' [7]. The term risk is defined as the 'combination of the probability of occurrence of harm and the severity of that harm' [7]. Usually, however, it is not possible to directly assess the harm that is potentially caused by a system. Instead, safety managers identify the hazards of a system, i.e., 'potential sources of harm' [7]. In many domains, this vague definition is further refined. In the automotive domain, for example, 'hazards shall be defined in the terms of conditions and events that can be observed at the vehicle level' [7]. Usually, harm is only caused when a hazard, a specific environmental situation, and a specific operation mode of the system coincide. This coincidence is called 'hazardous event' [7].

The identification of these hazardous events and the assessment of the associated risks is the first step in any safety assurance lifecycle, namely the' hazard analysis and risk assessment (HRA)' as shown in Fig. 1. This step is performed during the very early phases of the development process, at the latest when the system requirements are available.

As a result of this step, safety goals are defined as top-level safety requirements, which have to be incrementally refined during the safety assurance lifecycle. Usually, any safety requirement consists of a functional part and an associated integrity level. The functional part defines what the system must (not) do, whereas the integrity level defines the rigor demanded for the implementation and verification of this requirement. The integrity level depends on the risk associated with the hazardous event that is addressed by the safety goal. For example, ISO 26262 defines so-called automotive safety integrity levels (ASIL).

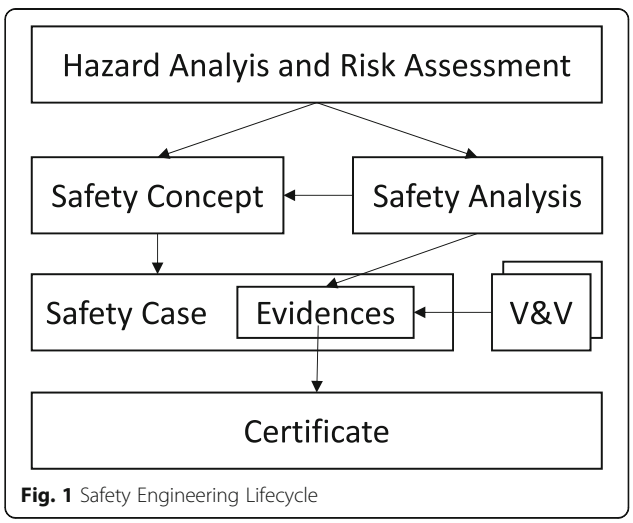

Fig. 1 Safety Engineering Lifecycle

In order to break down the safety requirements, the subsequent steps in the safety assurance process should be performed in parallel to the development activities. To this end, the available development artifacts are used as input to safety analyses in order to identify potential causes of the identified system failures. A wide range of established analysis techniques is available. Failure Modes and Effects Analysis (FMEA) and Fault Tree Analysis (FTA) are certainly the most widely used safety analysis techniques in practice.

Based on these results, a safety manager derives a safety concept. Following the idea of ISO 26262, a safety concept can be defined as a 'specification of the safety requirements, their allocation to architectural elements and their interaction necessary to achieve the safety goals, and information associated with these requirements' [7]. In the same way as the developers incrementally refine the system over the different development phases, the safety manager analyzes the refined development artifacts step by step and refines the safety concept accordingly.

Finally the safety manager has to define a safety case, which forms the basis for certification. A safety case can be defined as an 'argument why an item is safe supported by evidence compiled from work products of all safety activities during the whole lifecycle' [7]. Actually, a safety case can be derived from a safety concept by extending the latter with evidences proving that the requirements have been fulfilled. Evidence might be anything supporting an argument in the safety case. Evidences of particular importance are the results of validation and verification activities as well as safety analysis results. Since a safety case compiles all evidences that are relevant for proving the system's safety, it is an efficient basis for safety certification.

2.2 Safety challenges in open Systems of Systems

Such an established, conventional safety assurance process presumes that the system including all its constituents and all possible configurations and modes of operation are completely known prior to a final certification. Any kind of modification is considered as modification requiring re-certification of the system. Obviously this leads to various challenges for the safety assurance of open systems of systems.

The structure and, in consequence, the resulting collective behavior of open systems of systems is not known at design time. Systems dynamically enter and leave the collective and the constituent systems continuously adapt themselves in order to provide optimized performance in that dynamically changing context. Even in the case of reconfiguration as one of the weakest variants of self-adaptation, and even if all possible configurations of all constituent systems were known at design time, the resulting permutation of configurations would lead to a state space explosion impeding any design time analysis. More flexible self-adaptation approaches would – from a safety point of view – be comparable to a software modification requiring complete re-certification of the system. And, in fact, as they are heterarchical systems, there is no central integrator in open systems of systems who could be responsible for the safety assurance of the resulting system collective. Instead, the systems integrate themselves dynamically at runtime and a final safety assessment of the integrated system of systems by a human safety assessor is impossible.

While the safety assurance community still strongly focuses on very static systems, the dependability engineering community already addresses these challenges by evolving from dependability to resilience [8, 9]. Resilience in this context is defined as "the persistence of dependability when facing changes". *Changes* are characterized in three dimensions as shown in Fig. 2: The nature of a change may be functional, environmental, or technological. The prospect of a change may be foreseen, foreseeable, or unforeseen. The timing of a change may be short term, medium term, or long term.

Obviously, the main challenges of open systems of systems result from changes, namely from changing structures and behavior (functional), from changing context (environmental), and partly from changing technologies, such as changing communication technologies. Applying the idea of resilience to assuring the safety of open

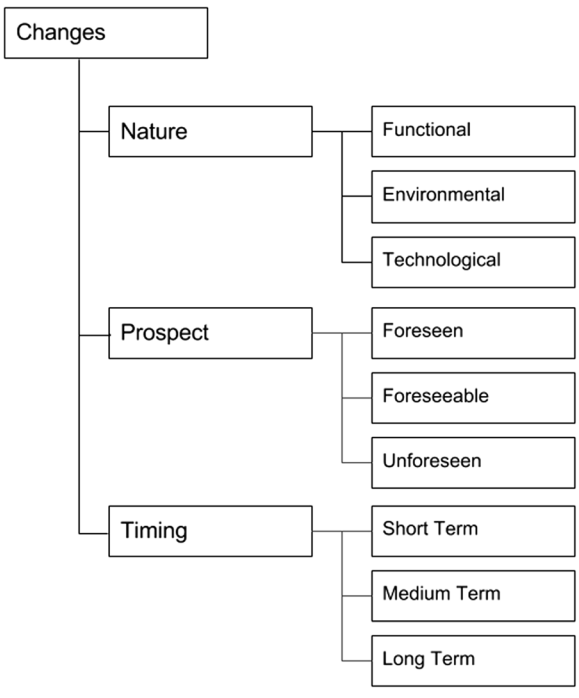

Fig. 2 Classification of Changes addressed by Resilience according to [9]

systems of systems therefore leads to the challenge of guaranteeing the "persistence of safety when facing changes".

2.3 Openness

So far, we have talked about open systems in general. In an open system of systems, individual systems, so-called constituent systems, dynamically connect to each other in order to collectively provide value-adding superordinate functionality. Instead of hierarchical systems, this leads to heterarchical systems of systems. This means that there is no hierarchy, but all systems have equivalent rights and have their own mission and objectives. They are usually developed independently by independent companies, which have their own business goals, strategies, and missions. In contrast to closed systems, this also means that there is no central integrator, no OEM, who is responsible for the final integration and who is liable for the quality of the overall system of systems. Nonetheless, the constituent systems need to collaborate as a collective – just as individual people work together in teams – aiming for joint goals and by following certain rules. In order to realize an open system of systems, the structure and behavior of an open system of systems dynamically emerge at runtime, leading to very flexible and adaptive solutions. In consequence, the single constituent systems need to adapt themselves to such a dynamic context.

To define a safety assurance approach for open systems of systems, we need to take a closer look at the property of openness. Therefore, we suggest a classification that primarily reflects those classification criteria that are of importance for safety assurance. The different possible degrees of openness are illustrated in Fig. 3.

Closed system means there is a final integrator who guarantees the system's safety. This also means that the system will not change its structure beyond the configuration space considered during the safety assessment – i.e., its safety can be analyzed completely during development time since everything relevant is known. Taking the example from the agricultural domain mentioned in the introduction, the tractor as well as the baler would be closed systems.

If we regard open systems of systems, it is important to distinguish between *closed ecosystems* on the one hand and *open ecosystems* on the other hand. Even though the systems integrate themselves dynamically and flexibly at runtime, a closed ecosystem means that all the systems must follow the same rules. And all possible ways of collaboration are known – at least at an abstract level. For example, agriculture defines such a closed ecosystem. The tractor and the baler follow a common communication standard, and it is possible to define common safety rules and guidelines within the industry. Moreover, the ways in which a tractor and an implement may interact are known and can be analyzed for an ecosystem – even though the concrete combination of tractor and implement is unknown at design time. From a safety perspective, we can thus state that we are dealing with *foreseeable changes*, which can be addressed through adequate runtime support, as we will further discuss in this article. In an open ecosystem, however, such rules are not available, as, for example, systems of various different industries need to collaborate in an ad-hoc manner without prior design time consideration of the unfolding collaborations. This means that it is hardly possible to define common safety rules (in the sense of a generic safety concept suitable for any conceivable collaboration during runtime) because we cannot predict which (types of) systems will eventually interact in which way. Referring to the definition of resilience, this means, in particular, that assuring safety in open ecosystems means handling unforeseeable changes. Although open ecosystems pose very interesting research challenges, we will focus on closed ecosystems in this article as an intermediate step between today's closed systems and those open ecosystems foreseen for the distant future.

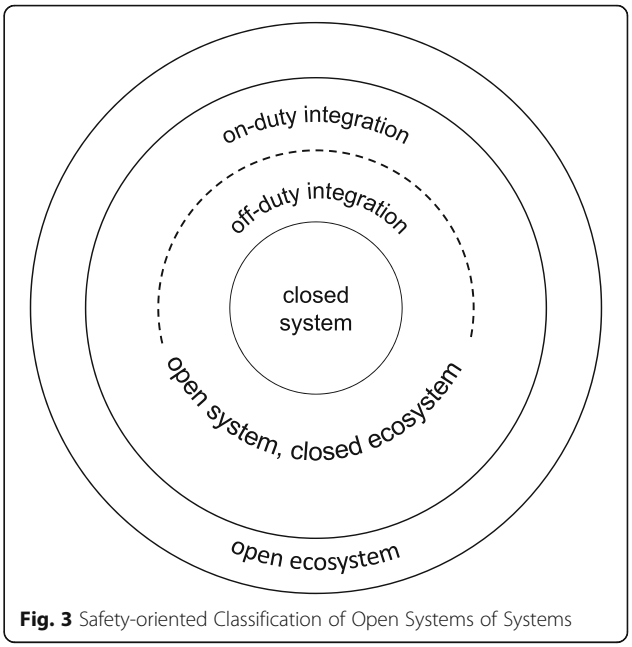

Fig. 3 Safety-oriented Classification of Open Systems of Systems

Considering open systems in closed ecosystems, we further distinguish between on-duty integration (concurrent integration and operation) on the one hand and off-duty integration (preemptive integration) on the other hand. Off-duty or preemptive integration means that the operation of the system of systems is suspended when its structure changes. For example, the tractor and the baler can be set to a special integration mode and it is possible to check the safety of the resulting system of

systems before it starts operation. This means, in particular, that there is sufficient time to perform a safety analysis at the integration level. In addition, even though a farmer is not a safety expert, he might be involved in the checking process, e.g., by performing and confirming manual checks. These advantages are not true in the case of on-duty or concurrent integration. In this case, the structure of the system of systems might change dynamically while it is operating. If we take a crossroads assistant as a Car2X (cars communicating amongst each other or with other entities to realize cooperative services) scenario as an example, the cars approaching a crossroads must dynamically form a system of systems to manage the right of way collectively. Obviously, it is not an option to stop the cars to check the safety of the resulting system of systems before it continues operation. Consequently, on-duty integration poses much higher demands on the required runtime safety assurance approaches since there are hard real-time constraints and user intervention is not possible.

2.4 Safety models at <<run.Time>>

In order to handle changes in general, self-adaptation has evolved as a promising approach for enabling systems to flexibly and nonetheless systematically adapt themselves to changes. Models@Run.Time [5] provides a particularly interesting basis for self-adaptation as it makes adaptations tangible and explicit, which is of utmost importance for safety assurance. In previous work, we therefore introduced the idea of safety models at runtime (SM@RT), which can be used to assure the safety of adaptive systems [2, 4, 10].

In general, "a model@run.time is a causally connected self-reflection of the associated system that emphasizes the structure, behavior, or goals of the system from a problem space perspective" [5]. Applied to safety this means that we shift modified safety models (such as certification models, safety case models, or safety concept models) to runtime. Using self-reflection and monitoring mechanisms, the models are continuously updated to reflect the associated systems' safety state. Based on these safety models at runtime, the system is enabled to systematically reason about and take actions for preserving the system's safety at runtime. To this end, it is possible to apply model-based runtime analyses, which have their origin in established and accepted model-based safety analyses used in conventional safety assurance approaches. This means that safety models at runtime can be seen as an evolution of model-based safety assurance, which increases the likelihood of acceptance by certification bodies. Our approach is based on safety certificates at runtime, which we will describe in more detail in the section *Certificates at Runtime*, and has proven its applicability and acceptance in different industrial applications involving independent certification bodies.

Thus, we essentially need to shift "safety intelligence" from development time to the systems' runtime models in order to enable dynamic safety assurance and management. The question, however, is now how much safety intelligence can and should be shifted to runtime. Considering the need to always provide a sound safety argument, which typically is a creative task requiring a lot of experience, it makes sense to try to keep as much of it as possible at design time. This means, in turn, that we only shift aspects to runtime that cannot be resolved at design time already due to non-resolvable uncertainties. Still, higher levels of safety intelligence at runtime can provide more flexibility and a more optimized "trade-off" between safety and performance. For further reference, we briefly discuss such levels in [11, 12].

As a related issue, we would like to emphasize that the need to shift safety models and management to runtime is not only a burden. It also opens up a huge potential in terms of optimizing performance while ensuring safety at any point in time. The reason for this is that in traditional approaches, any volatile property of the environment had to be considered in the sense of a worst-case assumption, thus leading to designs and parameterizations with sub-optimal performances. Now, when safety-related properties and dependencies are made explicit and analyzable at runtime, the performance levels of a system could be adjusted dynamically based on, for instance, the current safety and general quality properties of a sensor. This, of course, goes into both directions; i.e., if the quality of information deteriorates or if there is a sensor failure (for example), graceful degradation (of the application performance) can be conducted – which is done quite frequently today. Another promising opportunity is to build some flexibility into the top-level safety goals (or in the way safety goals are selected, interpreted, and refined based on changing situations); in other words, to shift aspects of the hazard and risk analysis to runtime. Here it would be conceivable to factor in information regarding the usage context. An example would be an agricultural machine operating on a perfectly flat and dry field versus the same machine operating on difficult topography and wet soil. The first case could have less strict safety requirements in some regards, thus enabling a higher level of performance.

Moreover, if the safety guarantees are not sufficient, graceful degradation could be required. This might also happen during operation if, for instance, a sensor fails and the system needs to be dynamically reconfigured in order to replicate the sensor value based on fusion of alternative sensors. If this leads to a deterioration of qualities (and safety guarantees in particular), the overall application might require new parameterization, such as

lower top-speed of an automated vehicle. Consequently, in addition to the safety models at runtime, reconfiguration/adaptation models would be required as well. Thinking a bit further into this direction, there would not only be such models operating tactically on short timescales, but ideally also models operating more strategically and on larger timescales.

In essence, we argue that future systems would greatly benefit from comprehensive multi-concern models at runtime operating on different levels, which would enable a dynamically managed informed trade-off between different concerns as well as ecosystem-wide orchestration to facilitate higher-level strategic goals. With that, new application features can be enabled, which are impossible to realize based on the scope of the constituent systems alone.

As a first step towards a corresponding concept, it is important to understand the different types and foci that runtime models might assume. In the following, we will introduce the B-Space concept, which provides an overall vision as well as structure and classifications regarding the interplay of multi-concern models at runtime.

3 The B-space concept

We motivated that in order to handle adaptive open systems of systems, it is necessary to shift parts of the development activities from design time to runtime. Models at runtime provide the basic core concepts for doing so. And indeed, many approaches recently emerging for handling such systems hidden behind buzzwords such as Internet of Things, Cyber-Physical Systems, or Industry 4.0 actually rely on ideas similar to those of models at runtime. For example, the idea of *Administration Shells* and *Digital Twins* or *Digital Shadows* has emerged as one core concept for managing open systems of systems – particularly in the context of Industry 4.0. In principle, digital twins are model-based representations of entities of the real world such as machines, cars, or persons. So, in fact, digital twins are model-based representations reflecting the associated systems / things from a problem space perspective. Therefore, digital twins can be seen as a first step towards models at runtime. The basic idea of digital twins, however, is not sufficient to realize open adaptive systems.

Many properties, including safety, are not completely modular, so the sum of all digital twins would not adequately represent the overall system of systems. Therefore, it is necessary to have models at different levels of the dynamic composition hierarchy.

Moreover, digital twins or digital shadows are often used in a very static way for service brokerage or for simulations. In fact, it is necessary to use them in the sense of models at runtime, however. This means that it is necessary to use models at runtime to enable constituent systems, as well as the open adaptive system of systems, to reflect their current status and the status of its context. In other words, it is necessary to create a kind of self-awareness, which is in turn the basis for reasoning about and assuring relevant system properties and qualities at runtime. Again, this monitoring and assurance process must be enabled at different levels. At the constituent system's level, for example, the models monitor the status of a single car and ensure its quality within the given runtime context. At the system of systems' level, for example, models reflect all cars participating in an autonomous platoon, including the platooning process, in order to monitor and assure the quality of the platoon as a whole. Potentially, it is also reasonable to have models at a global level, for example, to reflect information gathered from as many autonomous cars around the world as possible in order to improve functionality and quality in the long run based on field data systematically fed back into the models.

As a further aspect, it is usually necessary to have different models for different purposes. Even if we consider safety assurance only, we need different models such as runtime safety certificates, behavior models, timing models, or dynamic risk models, to name just a few examples.

The set of all of these models and their interrelationships spans a virtual space reflecting and predicting the dynamic structure and the behavior of entities of the real world as well as their desired and undesired interactions. We call this virtual space a ***B-Space***. It reflects the real world, which we refer to as the A-Space. Therefore, a B-Space is kind of a collective digital awareness of the real world.

Following the definition of models@run.time [5], a B-Space could therefore be defined as "a set of models@run.time that are a causally connected self-reflection of the associated systems of an ecosystem or system of systems and the interactions and interdependencies between them. The models emphasize the structure, behavior, or quality of the systems, the systems of systems, and the ecosystem from a problem space perspective. Moreover, the B-Space includes all analyses required for monitoring and optimizing the state and performance of the systems, the system of systems, and ecosystem".

In order to structure the different kinds of models@run.time that can be used in a B-Space, we use the classification shown in Fig. 4. Even though there are no restrictions on the kinds of models and analyses used in a B-Space, the classification provides a generic scheme covering the most typical cases. To this end, we identified three dimensions that mainly reflect the purpose of the model.

The first dimension refers to the hierarchy / composition *level* of the ecosystem that is reflected by the model. At the lowest level, the models reflect a property

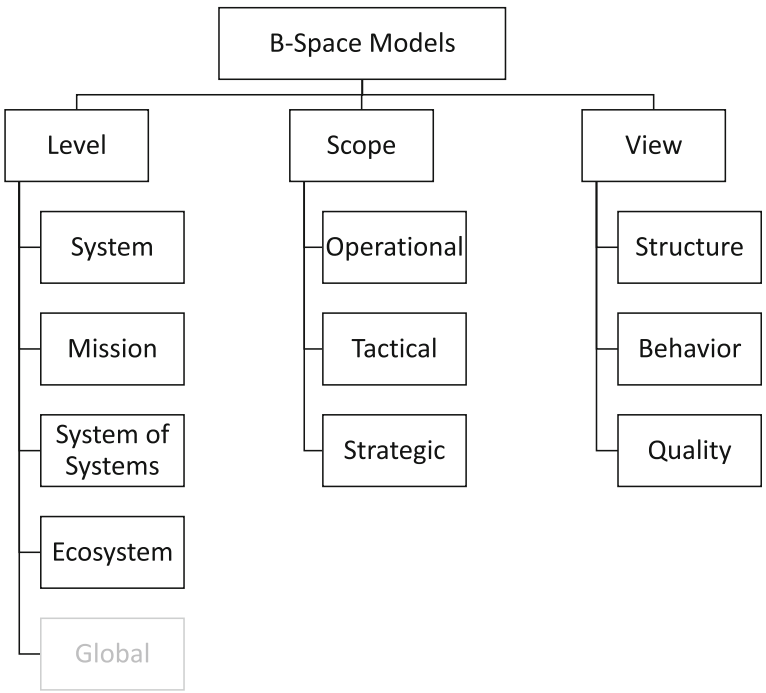

Fig. 4 Classification of Types of Runtime Models in a B-Space

of a single *system* (e.g., a car). On top of the system level, there is an intermediate level for models reflecting a certain *mission* involving several systems. Instead of reflecting a system of systems as a whole and holistically, mission models are limited to a specific mission, i.e., a certain task several systems have to accomplish as a collective, such as a platooning or lane merging mission in Car2Car settings or a harvesting mission involving several combines and tractors. Mission models reflect the current status of the mission and, if necessary, provide the basis for adapting the single systems in order to optimize the mission. Particularly for safety assurance, mission models are very important since it is much easier to define, monitor, and dynamically assure safety bounds of a collective of systems for a concrete scenario than to do so generally for any possible collaboration scenario that might occur in a dynamic system of systems. In consequence, this means that models at the system of systems level reflect the *system of systems* as a whole and are not limited to specific missions. Then again, these models are usually also more abstract than mission models. Models at the *ecosystem* level refer to general aspects of the ecosystem. These models may include service repositories, domain ontologies defining signal semantics, standards, general orchestration rules and mechanisms, overall performance models, or business models, to name but a few examples. The purpose of these models is to monitor and improve the ecosystem as a whole. In principle, and in addition, *global* models could reflect aspects that are of relevance across ecosystems. However, it is more likely that a B-Space reflects a single ecosystem such as Connected Smart Mobility or Smart Farming.

The second dimension refers to the *scope* of the model from the strategic point of view following a typical strategy hierarchy from *operational* to *tactical* to *strategic* scope. Models with an *operational* scope are used for immediate monitoring and control. They are typically quite simple and fast and most likely used at the system or mission level, enabling very short, low-level control and adaptation loops in the range of milliseconds. They focus on the here and now. For example, operational safety models at runtime at the system level monitor and protect a system in real time with very low latencies by detecting problems and adapting the system in order to prevent hazards.

Models with a *tactical* scope are used for adaptations optimizing the system or system of systems considering the next minutes or hours. Therefore, they are usually more complex, but do not need be very fast, and latencies of several seconds or even minutes are acceptable. Tactical models are typically found at the mission or system-of-systems level to optimize the collaboration of collectives in mid-level control and adaptation loops. For example, tactical safety models at runtime are used to monitor, predict, plan, and control missions such as a lane merging procedure on a highway. They consider all involved vehicles and try to optimize the mission as a

whole by dynamically monitoring and adapting important mission characteristics such as the required gap size or the vehicles' speed or the merging order for the next 30–60 s depending on the current traffic volume, types of vehicles, weather conditions, etc. And while operational models are usually used to detect and react to critical situations, tactical models are typically used to prevent critical situations through tactical, predictive control and adaptation of the system of systems to optimize the collective's safety probabilities.

A *strategic* scope usually covers the complete ecosystem or system of systems from a strategic long-term perspective. Using field data, they optimize ecosystems as a whole in high-level adaptation loops. They optimize the ecosystem's goal achievement, which may take several minutes, hours, or even days. The adaptation loops might even involve managers and developers. Their purpose is to continuously and systematically learn from field experience and to optimize the ecosystem accordingly. This might involve Big Data analytics as well as complex optimization algorithms. For example, strategic safety models at runtime for a smart mobility ecosystem could be used to monitor any kind of incident, any previously unknown but important driving situation, any experience gained from adaptations at the mission level, etc. If any accident or incident happened or could be avoided by the driver, this information can be used to improve the (collective) autonomous behavior of the systems. For future applications, the models reflecting the field data could be used to derive thousands of additional parallel simulation scenarios for testing and improving the systems' behavior. The vehicles would not only be tested prior to their delivery, but would be continuously tested and improved in the B-Space using simulations based on strategic models@run.time, which in turn would be continuously improved by reflecting on and learning from real-world scenarios.

The third dimension of the classification scheme refers to typical *views* also used in conventional model-based development approaches. This includes, but is not limited to, *structural*, *behavioral*, and *quality* views. To assure the safety of open adaptive systems, these views are important since structure as well as behavior have a direct impact on safety. In combination with dedicated safety models in the quality view, such as dynamic safety analysis models, dynamic safety cases, or dynamic safety certificates, this enables dynamic and yet systematic safety assurance at runtime.

In general, it is worth mentioning that this classification is independent from the point of computation of these models. Even though it is very likely that operational system-level models at runtime, for example, are executed on the system itself, they could also run in the cloud. In the same way, mission models in a Vehicle2X setting could be executed on an LTE base station, or in a distributed manner on the individual cars, or in the cloud.

B-Spaces provide a conceptual framework for holistically handling open adaptive systems of systems. For safety assurance, it is very important to build a bridge from traditional safety engineering to dynamic runtime safety assurance in order to be accepted by certification authorities. Therefore – and in spite of the potential offered by the complete B-Space concept – for dynamic safety assurance, it is reasonable to start with operational system-level safety models at runtime and then incrementally widen the scope to tactical mission models and models at the system-of-system level.

Even starting with this reduced scope obviously implies a series of further technical and non-technical questions and challenges. Maybe the most important challenge is that stakeholders of a domain or ecosystem actually need to work together and agree on a concept such as the B-Spaces. Then it is necessary to commonly define standards and guidelines manifesting the conceptual backbone and specifying key aspects, such as metamodels ensuring interoperability between the range of different models that are contributed to the B-Space from different companies.

4 Safety assurance using B-spaces

For safety assurance in conventional engineering, the safety view as a refined quality view is of particular interest. Even though functional models and aspects such as tests are also important building blocks for safety assurance, all safety-relevant information is eventually compiled in dedicated safety models and the complete safety assurance process is controlled based on these safety models. Iteratively, the residual risk is evaluated, cause-effect chains are identified, appropriate countermeasures are selected, their appropriateness is evaluated, and the process starts again with a re-evaluation of the residual risk – until the residual risk falls below an acceptable threshold.

In principle, this process is transferred – at least partially – into runtime for dynamic safety assurance. B-Spaces shall now possess the required knowledge (in form of models at runtime) as well as the means (in form of management mechanisms that, for instance, realize dynamic self-adaptation) for transferring certain aspects of the assurance process from the human engineer at development time to the systems at runtime. How much of the process *should* actually be transferred depends on the degrees of openness and adaptivity in the respective systems and the corresponding B-Space "level" and "scope" in correspondence with Fig. 4. In [12], we identified a set of distinct "safety-intelligence" levels (safety certificates at runtime, safety cases at runtime and hazard

and risk analysis at runtime), each implying a certain amount of shifted safety intelligence and each appropriate for different degrees of openness and adaptivity / levels and scopes of the B-Space. The higher categories of "level" and "scope" require (or at least greatly benefit) higher levels of shifted safety intelligence. While safety certificates at runtime are well suited for system/mission level and operational scope, if I would like to operate on the ecosystem level (i.e. having an open ecosystem with formerly unknown constituent systems entering dynamically), then I require some means to dynamically analyze hazards and risks for those completely new collaboration schemes that form within my ecosystems. In the following we re-iterate these distinct levels of runtime safety-intelligence before we go on to exemplify in more detail (based on ConSerts) how a runtime safety certificate approach can be set up.

Certificates at runtime (such as ConSerts) are post-certification artifacts. This means that only few unknowns exist regarding the system's environment and collaborations and that, based on this general knowledge, a certain variability is built into the safety certificates, which is bound to formal conditions. These conditions are associated with properties of the environment (such as safety requirements of an external service) that can be resolved at runtime. ConSerts, as an instance of certificates at runtime, enable a compositional safety evaluation in a heterarchical system-of-systems setting. Due to their limitations, however, ConSerts are mostly suited for closed ecosystems and pre-engineered adaptive behavior. Overall, ConSerts are well suited to constitute a B-Space for dynamic safety assurance on the mission level and with an operational scope. The B-Space would then consist of the ConSerts for all cooperating systems (as well as auxiliary models such as type systems for services and safety properties). Based thereon, safety guarantees can be calculated dynamically and dynamic management can be triggered to maintain necessary guarantees and to optimize performance. Even though ConSerts only cover a slice of the capabilities and dimensions outlined in Fig. 4 (i.e. mission level, operational scope, focus on safety as a quality), they show what we understand under a runtime model and what kind of implications this has regarding established engineering practice. ConSerts will therefore be discussed in more detail in the subsequent chapter. Of course, apart from ConSerts there are other approaches from the state of the art that fit into the B-Space concept as well, either addressing other dimensions (cf. Fig. 4) or addressing orthogonal concerns (such as monitoring properties by runtime verification, which can be relevant for arbitrary B-Space dimensions). Examples are brought up in the following and in the conclusion.

The next level are safety/assurance cases at runtime, which, as SM@RTs, would be the backbone for the dynamic acquisition of evidence by means of runtime V&V (i.e. validation and verification) strategies. And, based thereon, dynamic adjustment of V&V models would enable an additional dimension of flexibility. One example are Digital Dependability Identities (DDI). A DDI can be understood as a dependability-related aspect of a B-Space. A DDI contains all the information that uniquely describes the dependability characteristics of a system or component [4]. This includes attributes that describe the system's or component's dependability behavior, such as fault propagations, requirements on how the component interacts with other entities in a dependable way, and the level of trust and assurance, respectively. The latter can be described using concepts from the theory of safety contracts. A DDI is a living modular dependability assurance case. It contains an expression of dependability requirements for the respective component or system, arguments of how these requirements are met, and evidence in the form of safety analysis artifacts that substantiate these arguments. A DDI is produced during design, issued when the component is released, and is then continually maintained over the complete lifetime of a component or system. DDIs are used for the integration of components into systems during development as well as for the dynamic integration of systems into systems of systems in the field.

The penultimate level of runtime safety intelligence would be to also shift parts of Hazard and Risk analyses to runtime, hence enabling dynamic risk assessment or runtime risk analysis. This, in turn, would be a starting point for enabling dynamic adjustments in the safety argumentation, which, in turn, might affect the dynamic certificates.

Ultimately, safety assurance would be shifted completely to runtime. Comprehensive models within the B-Space covering all relevant concerns would constitute the basis for a fully automated dynamic safety lifecycle yielding an optimal trade-off between safety and performance within an ecosystem. Approaches from the field of artificial intelligence might provide the reasoning capabilities required on top of the information provided by the B-Space. Clearly, this idea of emergent safety is provocative and seems far-fetched; it should be understood as an ultimate vision for tomorrow rather than a concrete research goal for today.

4.1 Certificates at runtime – Conditional safety certificates

In this chapter, we re-iterate the concept of ConSerts as introduced in [2, 3] and outlined in [4] and illustrate how ConSerts are an aspect of and an initial step towards the concept of B-Spaces. In particular, it is shown how B-Space models could look like and what the implications would be with respect to established engineering practice.

ConSerts operate on the level of safety requirements. They are issued at development time and certify specific safety **guarantees** (i.e., guaranteed safety requirements) that depend on the fulfillment of specific **demands** (i.e., safety requirements demanded from the environment) regarding the environment. In the same way as "static" certificates, ConSerts shall be issued by safety experts, independent organizations, or authorized bodies after a stringent manual check of the safety argument. To this end, it is mandatory to prove all claims regarding the fulfillment of provided safety guarantees by means of suitable evidence and to provide adequate documentation of the overall argument – including the external demands and their implications.

There are, however, some significant differences between ConSerts and static certificates that are owed to the nature of open systems: A ConSert is not **static** but **variable and conditional**; i.e., it comprises a number of variants that are conditional with respect to the (dynamic) fulfillment of demands. Moreover, a ConSert must be available in an executable (and composable) form at runtime (i.e., as a safety model at runtime) and systems must be equipped with corresponding mechanisms to operate on the ConSerts. Conditions within a ConSert manifest in relations between potentially guaranteed safety requirements, which can simply be denoted as guarantees, and the corresponding demanded safety requirements, i.e., demands. Demands always represent safety requirements relating to the environment of a component, which cannot be verified at development time because the required information is not available yet. These demands might directly relate to required functionalities from other components. On the other hand, evidence can be required beyond that since safety is not a purely modular property and it cannot be assumed that a composition of safe components is automatically safe. To this end, ConSerts support the concept of so-called Runtime Evidences (RtE) as an additional operand of the conditions. RtEs are a very flexible concept. In principle, any runtime analysis providing a Boolean result can be used. RtEs might relate to properties of the composition or to any context information, e.g., a physical phenomenon such as the temperature of the environment that is safety-relevant and could be measured with a sensor. Other RtEs require dynamic negotiation between components, such as for determining independence between different services used. To this end, a dedicated protocol could be used that builds traces through the composition hierarchy starting from the services in question to identify common elements in the traces.

Accordingly, a ConSert can be specified as a set of Boolean functions, which, in turn, can generally be represented by a corresponding set of (potentially overlapping) rooted directed acyclic graphs in a graphical specification. The root of each of these directed acyclic graphs is constituted by a potential safety guarantee, which becomes true if, at runtime, related (according to the Boolean logic) demands and runtime evidences are satisfied. Each graph consists of:

- A set of Boolean input variables; i.e., demands and runtime evidences
- A set of Boolean gates; i.e., {*and, or*}
- A set of directed edges connecting the elements
- A Boolean output variable; i.e., the guarantee

The guarantees and demands can be specified based on the grammar introduced in [3]. An example follows later in this article.

Note that ConSerts are designed to harmonize with the pre-engineered dynamic reconfiguration. Each constituent system (/component) can have a series of configurations, which can be switched at runtime. Each of these configurations is characterized by a specific profile of required and provided services and equipped with its own specific ConSert, which provides the mapping function between guarantees, demands and runtime evidences as described above (cf. Figure 5). Within the B-Space, the reconfiguration models are an important additional ingredient for enabling not only dynamic assessment of safety guarantees, but also their management and enforcement.

To be utilized as a runtime model of the B-Space, ConSerts (as well as other relevant models) need to be transferred into a suitable representation. Suitability clearly depends on the characteristics of systems and domains and on the corresponding constraints. When we are talking about tiny microcontrollers where resources are very scarce, it may make sense to bring ConSerts into a BDD representation (i.e. a set of BDDs, one for each potential guarantee) [2, 3], which can be optimized at development time to enable very easy evaluation and a minimal memory footprint at runtime. If we are talking about more powerful IoT devices, an XML-based representation might be more appropriate, and if an Internet connection is always available, even a cloud-based ConSert evaluation is conceivable. Looking at the B-Space as a whole, different runtime models for different concerns might reside in different places. ConSerts and reconfiguration models might be distributed modularly over all constituent systems, while other models, such as those working on a larger time scale and with a more strategic scope, may reside and be managed in the cloud.

At runtime, dynamic hierarchies are formed when a top-level application is instantiated in an open adaptive system of systems. Thus, the top-level application requires basic services from other systems, which might in turn require basic services from yet another system, and so on. In this process of forming a dynamic hierarchy, involved constituent systems might be reconfigured/adapted to be

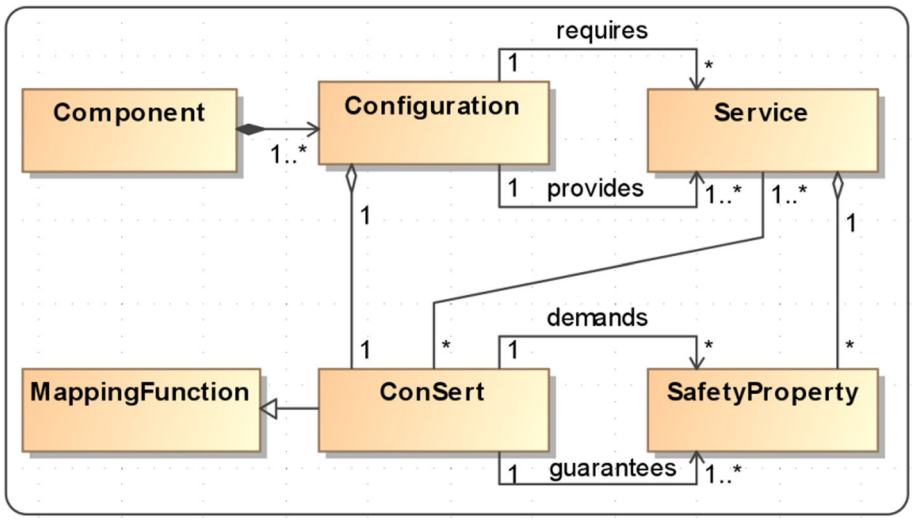

Fig. 5 The Concepts of Dynamic Reconfiguration and ConSerts [2]

fit for their tasks. This also implies that constituent systems deliver services for consumers of different composition hierarchies. Therefore, a heterarchical system-of-systems structure is formed overall.

Throughout these hierarchies, ConSerts are composed and evaluated, yielding the current safety guarantees for the respective hierarchy at their root node. The evaluation is performed from leaves to root, starting from leaf ConSerts that have only runtime evidences and no demands. Generally, based on the fulfillment of demands and runtime evidences, it is determined for each ConSert through Boolean logic which of its potential guarantees are currently valid. If several guarantees of a ConSert are valid, one (i.e. the "best" one) shall be selected. This can be done based on a simple absolute order from best to worst over the guarantees of a ConSert or based on more sophisticated utility functions, which might include context information in the equation to make a more informed decision on what is actually to be considered "best" under given circumstances.

Based on the continuously monitored safety guarantees, managing actions might be performed by the system(s) in the sense of the MAPE-M@RT (monitoring, analysis, planning and execution based on models at runtime) [13] control loop. Thus, the context is monitored and the current safety guarantees are determined; it is analyzed whether the system has the appropriate configuration and parameterization; potential adaptations are planned and executed – all based on the M@RT of the B-Space, in particular the ConSerts and correlated adaptation/reconfiguration models.

4.2 Example

We further illustrate ConSerts with an example from the agricultural domain, which has previously been presented in [4]. The agricultural domain today pioneers innovative applications involving systems of systems and dynamic integration. One of these applications is the so-called *Tractor Implement Management (TIM)*. The TIM functionality enables agricultural implements to control typical tractor functions such as velocity, steering angle, power take off, or auxiliary valves. It is possible to fully automate implement-specific work procedures and to optimize them with respect to parameters such as performance, efficiency, or wear and tear. TIM utilizes a standardized bus system for communication between the participating devices and machines. During TIM operation, control is typically assumed by the implement. It uses the TIM functions of the tractor (e.g., controlling velocity, steering angle, power take-off, hydraulic valves) and auxiliary devices, such as sensors for the respective automation purpose, displays data to the operator, and executes operator inputs. Between different tractors, implements, and auxiliary devices such as virtual terminals (providing the operator UI) or GPS systems of different manufacturers, a huge space of configurations arises, which makes it unfeasible to analyze each potential configuration a priori at development time. For this reason, those TIM applications already available on the market today only work for predetermined concrete pairs of tractors and implements whose integration has been thoroughly analyzed at development time by the involved manufacturers.

The benefit of ConSerts (or SM@RT in general) in this setting is pretty clear. Assume there is a farmer who owns a TIM-capable tractor and a TIM-capable round baler. The TIM baling application is running on the implement, and the user interface is displayed on a virtual terminal in the cabin. In addition to a standard configuration, the baling application also supports an extended

configuration, which additionally incorporates a swath scanner device. This device is mounted on the front of the tractor and measures the volumetric flow and the location of the swath to further optimize the baling operation in terms of tractor speed and steering angle. The baling application can then be enabled when tractor, implement, virtual terminal, and swath scanner are connected and the automated ConSert-based interoperability and safety checks have been successful. Corresponding information is provided to the operator via the terminal. Parameterization and constraints are set appropriately. The actual round baling process can then be activated by the operator, who thus relinquishes control to the round baler. The round baler commands the tractor to drive over the swath with optimal acceleration rates and speed. When the bale reaches a preset size, the tractor decelerates to a standstill and the bale is ejected. The process can then be re-started by the operator.

4.3 Engineering of ConSerts

For the engineering of ConSerts, the role and viewpoint of the implement manufacturer are assumed in this example. The goal of the manufacturer is to develop a round baler with TIM support. From a functional point of view, it is known by the manufacturer (due to existing standards) what the interfaces between the potential participants look like and how they are to be used. However, the implement manufacturer does not know anything about the safety properties of these functions.

From a safety point of view, the engineering of the baling application starts top-down with an application-level hazard and risk analysis. Assume that the agricultural manufacturers agreed by convention that during the operation of a TIM application, the application (and thus the application manufacturer) has the responsibility for the overall automated system. Therefore, the safety engineering goal is to ensure adequate safety not only for the TIM baling application or for the implement, but for the whole collaboration of systems that will be rendering the application service at runtime. Thanks to the ConSert-based modularization, it is thus sufficient to only consider the direct dependencies of the system under development on its environment. Potential "external" safety requirements will be associated either with demands regarding required services or with RtEs. At runtime, it will be determined whether the demands can be satisfied based on guarantees given by external systems (which, in turn, might have demands depending on yet (an)other external system(s)). This negotiation can thus range across several layers and incorporate a series of dependent systems and guarantee-demand relationships.

Relating the ConSerts from the example to the classification of B-Space models, we are talking about quality models (i.e., safety). ConSerts will mostly be utilized with an operational scope, but they might as well be input to considerations with a tactical or even strategic scope (e.g., strategic deployment of TIM machines with respect to weather conditions so that their performance over a season can be optimized). As regards the B-Space level where ConSerts reside, this is at least twofold. There are ConSerts at the system level, but due to their compositional and hierarchical nature, there is always at least one ConSert in a composition hierarchy that is at the mission level – the ConSert of the root node of the hierarchy. For this ConSert, the overall collaboration of the cooperative application (i.e., mission) has been considered in the corresponding safety engineering as exemplified above for the TIM baling application. Beyond that, it would also be conceivable to utilize ConSerts at the levels above, system of systems and ecosystems. However, to date we have not engineered any in-depth case study in that respect.

Going back to the engineering of the baler, relevant hazards of the TIM baling application might be self-acceleration or self-steering during operation or self-acceleration or power take-off during standstill. These hazards would be assessed with respect to their associated risks based on the risk assessment tables provided by ISO 25119 (i.e., the safety standard of the agricultural domain). In a subsequent step, corresponding top-level safety requirements would be derived. In addition, reasonable guarantee levels need to be identified. In the given example, it is conceivable that different safety guarantee levels are required for different locations (e.g., in the midst of nothing vs. a field close to a children's playground) or that guarantee levels are defined in interplay with application-specific parameters (e.g., acceleration or velocity levels, different degrees of automation, etc.) or with relevant conditions of the operating environment (weather, topography).

The next step is to develop a safety concept that ensures the satisfaction of the safety requirements and of the associated ConSert guarantees. This is done in a standard way: Safety analyses are applied to identify cause-effect relationships and to specify the failure logic; corresponding safety measures are identified; and, eventually, a conclusive safety argument is built up that factors in suitable evidence. The main difference to the engineering of closed systems is that besides possible internal causes, there might be external causes that may either be associated with safety properties of the required services or with RtEs. Moreover, there is also some degree of variability to be considered due to different ConSert guarantees and corresponding differences in the correlated demands.

With regard to the causes related to required services, there are two possibilities. First, it is possible to define internal measures, such as error detection mechanisms,

so that failures of the required services can be tolerated. Alternatively or in addition, it is possible to demand that the external service provider has to guarantee certain safety properties for the service. These safety properties need to be formalized and standardized for a domain in order to constitute the basis for the definition of ConSerts guarantees and demands. As for the RtEs, two categories can be distinguished: intra-device and inter-device RtEs. The former can be designed and implemented rather freely because they do not require any information from other external systems. The latter do require such information and thus, they need to be standardized or at least described in guidelines for a given domain. In reference to the example, assume that there is a top-level safety requirement that *self-acceleration must not occur during standstill*. Based on the hazard and risk analyses, it has been determined that this requirement needs to be assigned the integrity level *AgPL d*[1]. However, this is due to a relatively high degree of exposure assumed for bystanders, as would be the case for operation in the vicinity of a residential area. In other areas, *AgPL c* would be sufficient[2]. With ConSerts, it is now conceivable to optimize the trade-off between availability and safety by factoring in dynamic context knowledge. Specifically, this means that it would be possible to use, for instance, a GPS position and (in this case) annotated map data to distinguish between different usage contexts of the TIM system that imply different levels of safety requirements. Or, based on the vision of B-Spaces, we might have a wealth of up-to-date context information from and about systems and persons in the vicinity at our disposal. Of course, such a context-sensitivity mechanism needs to be safe in its own right, but for now let us just assume this can be done.

Thus, following these considerations, three different levels of ConSert guarantees are defined in the example: a) a high integrity one, enabling full automation features of the TIM application; b) a medium integrity one, enabling full features only in specific areas (or, alternatively, enabling operation with some constraints); and c) a default guarantee that can always be granted, enabling only a very constrained operation, e.g., without acceleration from standstill or automated steering. The high integrity guarantee would include *AgPL d* for self-acceleration in standstill as well as a series of other relevant guarantees omitted here for the sake of simplicity. The specification of the guarantee given next is based on a grammar [14] and on service types, safety property types, and rules of refinement specified in a domain-specific standard or guidelines:

```
TIMBalingSwSc(1): AgPL=b, SelfAcc{,-Standstill}.AgPL=d, LateAcc{30s,Stand-still}.AgPl=d, (...)
```

The first element of the specification denominates the associated service (by type) and gives an (absolute) order number for the guarantee (from 1 (best) to n (worst)). More sophisticated orderings could be useful but have not been considered yet for ConSerts. The next element describes an integrity level for the whole service. This is basically a shortcut and implies that all safety properties of the service (as specified by the standard or guideline) are guaranteed with the named integrity level. Then follows a series of concrete safety properties, whose types and parameters (for refinement) are also defined by the standard.

The next step from the ConSert perspective is to determine the demands (i.e., service-related demands as well as RtEs) that relate to the identified guarantees. This relation is modeled by means of a Boolean function, where the demands are input variables and the guarantee is the output variable. There is also a corresponding graphical specification technique based on directed acyclic graphs, where each function is represented by a tuple (D, R, BG, E, g): a set of Boolean input variables D representing service-related demands and RtEs R, a set of Boolean gates BG, a set of directed edges E connecting the elements, and a Boolean output variable g.

Overall, a ConSert is a set of such mapping functions, one for each guarantee level (of each offered service/function of a unit of composition). A unit of composition is typically a self-contained piece of hardware and software, i.e., a system. But it could also be just a piece of software, i.e., an application. If deployed concurrently with other applications on a shared hardware and particularly if dynamic download and update shall be possible, it is also important to include vertical dependencies between applications and (shared) resources in the eq. A corresponding extension of ConSerts has been developed in the EMC[2] project [15–17].

The definition of the guarantees, the demands, and the mapping functions is generally done conjointly with the development of the safety concept and safety argument. In fact, the resulting ConSert becomes an integral part of the safety argument because it needs to be shown in a convincing manner that the ConSert guarantees are actually valid given the fulfillment of their related demands.

4.4 ConSerts at runtime

ConSerts need to be transferred into a machine-readable form to enable dynamic evaluation, and there need to be corresponding mechanisms built into the systems that operate on this information (i.e., ConSerts as SM@RT) to conduct the evaluation. Of course, the evaluation protocols need to be standardized to ensure that every participating system is interoperable from a ConSert point of view. Assume that the operator has installed the swath scanner on the tractor and that tractor and round baler are coupled. The operator initiates TIM via the virtual terminal and explicitly selects the application variant that

provides flexible control of speed and steering based on the input from the swath scanner. The first step is now to start the application, i.e., to dynamically integrate the participating systems. After this has succeeded, the evaluation of the safety guarantees of the application is started. Note that the application forms the root of a dynamically formed composition hierarchy (in contrast to basic services or functions, which are rendered by lower-level components/systems and which are consumed by superordinate components/systems) and the correlated ConSert has the scope of this whole system-of-systems application. The evaluation of ConSerts starts from the leaf systems that have no external service-related dependencies. These systems determine their RtEs and propagate them up in the composition hierarchy. Eventually, all service-related demands of the root (i.e., the TIM baling application) can be checked and, together with the evaluation results of the RtEs, the top-level safety guarantees are determined.

5 Conclusion and future work

A gradual shift of safety intelligence to runtime will be indispensable to ensure the safety of future systems of systems. To which extent this shift will happen and how exactly it will be realized is yet an open (and always domain- and application-specific) question, which will be subject to complex developments with many stakeholders involved. At any rate, we perceive safety models at runtime as an important enabling technology for future systems of systems – because without safety, market introduction is impossible. And once we have actually established a certain degree of safety intelligence within systems, an additional benefit will be that systems can be enabled to dynamically adjust their safety and performance properties based on the current relevant system context in order to always guarantee the required safety while optimizing performance. This is in contrast to traditional safety engineering, where worst-case assumptions are made at development time whenever we face uncertainty with respect to a system's dynamic environments.

But safety is clearly not the only reason why runtime models will play a central role in future systems. In the context of highly connected heterogeneous systems of systems with lots of dynamicity, there needs to be intelligent management on different levels to ensure proper operation despite continuous changes. At the same time, such dynamic management shall not only react and mitigate, but also plan and optimize. Ideally, through comprehensive communication and sharing of information (provided by runtime models), there will be different optimization scopes (i.e., operational, tactical, and strategical) and system levels (i.e., constituent system, mission, system of systems, etc.) for intelligent dynamic management. This will go far beyond what we know today in terms of intelligent collaboration of systems, thus unlocking a huge potential for new applications and features.

However, as of today it is not clear how this vision can and will be achieved. Smart ecosystems are not designed at the drawing board and created on a green field, but slowly evolve and converge from different starting points, system types, and domains. Therefore, there are many stakeholders, regulations, constraints, etc. involved that impact this development, making it hard to identify a clear path forward.

With this article, we contribute to this process by introducing the B-Space, a conceptual framework providing structure and categorization with respect to runtime models of smart ecosystems. We further defined or recapitulated important notions from the worlds of safety and open and adaptive systems that drive the challenges and requirements of the B-Space. We believe that a construct like the B-Space will play a central role in any conceivable solution of comprehensively and intelligently managed smart ecosystems, be it explicitly or implicitly. Thus, we hope that in making the structure and categories explicit, it will be easier to advance related discussions and research and, in particular, to interrelate and integrate existing work from the related research communities.

Speaking about related work which might serve as a puzzle piece (or at least inspiration) regarding the B-Space, in the adaptive systems and models at runtime research community, there are already many significant contributions regarding runtime models for managing requirements at runtime [9], adaptation at runtime [18, 19], dealing with uncertainties [20], etc., which might be used in different dimensions of the B-Space (and with different purpose).

In former years, a lot of work, such as the QoS-A architecture [21], has been done in the field of dynamic quality of service management. A very active group doing adaptive QoS research over the last decades have been Nahrstedt and colleagues. In the Agilos project, the main focus was on runtime adaptation for QoS management and on how to make informed decisions on how, when, and what to adapt [22]. Even though being focused on communication systems and not explicitly working with runtime models, the employed concepts are certainly usable also beyond this scope and thus still a valuable input for the B-Space concept. One core concept of these approaches is to implement a control loop to realize the dynamic adaptation. This concept has equally been used in different approaches for adaptive ubiquitous computing and ambient intelligence systems, one prominent example being the MADAM/MUSIC projects [23]. Here, we already see the transition towards explicit models at runtime (i.e using architecture models for runtime adaptability), which is one of the baseline concepts for the B-Space.

Relatedly, runtime verification approaches (also based on runtime models), such as [24], can be utilized within a B-Space to dynamically obtain adaptation triggers. In case a certain observed property is violated, a dynamic self-adaptation of the system can be conducted to mitigate the issue. Such approaches often built on formal techniques, which in turn tend to lead to significant overhead. This overhead might be reduced by tuning the sampling, which in turn might affect the performance of the verification, thus requiring additional means to deal with that issue [25].

From the field of safety-related research, starting points for dynamic safety assurance through the B-Space are runtime certificates (as detailed in this article) as well as work on safety cases at runtime. An example for the latter is the recent ENgineering of TRUstworthy Self-adaptive sofTware (ENTRUST) approach [26]. ENTRUST uses a combination of (1) design-time and runtime modelling and verification, and (2) industry-adopted assurance processes to develop trustworthy self-adaptive software and assurance cases arguing the suitability of the software for its intended application. ENTRUST is focused on single self-adaptive systems (i.e. the system level in the B-Space) and not on systems of systems, thus an extension is certainly needed to utilize it for more open systems. One corresponding option might be an integration with the concept of runtime certificates, which are designed to provide adequate safety modularization on the level of constituent systems. Thus, the runtime certificates might be adapted dynamically after an adaptation of the system by means of the dynamic assurance case.

Apart from existing related work in the field outlined above, we presently also see a clear uptake in the safety research community on the topic of runtime assurances with significant collaboration projects leading the way. Aspects of runtime assurance played a role in huge European platform research projects such as EMC[2] and occupy front and center in recently started research and innovation actions such as SafeCOP and DEIS. At the same time, there is significant awareness on the part of industry, where this topic is being established on the roadmaps of world-leading embedded systems companies.

In summary, there are already many potential building blocks for comprehensive intelligent management of smart ecosystems. However, actually bringing everything together requires a bigger picture. We hope to contribute to this bigger picture by introducing the B-Space and its concepts and classifications. Within this frame, we can locate and interlink different building blocks from the state of the art and combine them into something bigger than the sum of the pieces. Even though we focus on safety as one important properties, other quality and performance properties shall ultimately be taken into equation as well.

6 Endnotes

[1]E.g., controllability 3 (non-controllable), severity 3 (life threatening), and exposure 3 (often; 1–10% of operating time). As per ISO 25119.

[2]E.g., controllability 3, severity 3, and exposure 2 (sometimes; 0.1–1% of the operating time). As per ISO 25119.

Acknowledgments

The work presented in this article is partially supported by the DEIS project – Dependability Engineering Innovation for Automotive CPS. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732242.

Authors' contributions

DS, MT; The authors respective contributions wrt. to the different sections are as follows. The introduction has been written by both DS and MT in roughly equal parts. The Section as well as the corresponding Sub-Sections have been written by MT, whereas the Sub-Section on Safety Models at Run.-Time was written jointly. The B-Space concept was mostly developed and written by MT with some support by DS. The Section on Safety Assurance using B-Spaces and the corresponding Sub-Sections were mostly written by DS with some support by MT. The conclusion was mainly written by DS. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Received: 13 November 2017 Accepted: 24 April 2018
Published online: 01 August 2018

References

1. J Rushby. "Just-in-Time Certification," 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS). Auckland: IEEE; 2007. p. 15–24.
2. Schneider D, Trapp M. Conditional safety certification of open adaptive systems. ACM Trans Auton Adapt Syst (TAAS). 2013;8.2:8. https://dl.acm.org/citation.cfm?id=2491467.
3. Schneider, D. Conditional Safety Certification for Open Adaptive Systems. PhD Theses in Experimental Software Engineering, Volume 48, Fraunhofer Verlag, ISBN 978–3–8396-0690-2, 2014/3/26.
4. Schneider D, Trapp M, Papadopoulos Y, Armengaud E, Zeller M, Höfig K. WAP: digital dependability identities. In: 26th international symposium on software reliability engineering; 2015. p. 324–9. IEEE.
5. Blair G, Bencomo N, & France RB. Models@ run. time. Computer. 2009; 42(10):22–27.
6. B. Cheng et al. Using Models at Runtime to Address Assurance for Self-Adaptive Systems. Models@run.time, 101–136. Switzerland: Springer International Publishing; 2014.
7. ISO, 26262 Road Vehicles - Functional Safety, 2011.
8. A. Avižienis, JC Laprie, B. Randell, C. Landwehr. "Basic concepts and taxonomy of dependable and secure computing." IEEE Transactions on Dependable and Secure Computing, vol. 1, pp. 11–33, 2004.
9. Cheng, B., Sawyer, P., Bencomo, N., & Whittle, J. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. Model Driven Engineering Languages and Systems. 2009;5795:468–483.
10. M Trapp, D Schneider. Safety Assurance of Open Adaptive Systems–A Survey. Models@run.time, 279–318. Switzerland: Springer International Publishing; 2014.
11. E. Armengaud et al. DEIS: Dependability Engineering Innovation for Industrial CPS. AMAA'2017, 21st Int'l Forum on Advanced Microsystems for Automotive Applications 2017, Springer LNM series book on Advanced Microsystems for Automotive Applications, pp.151–163, DOI: https://doi.org/10.1007/978-3-319-66972-4_13, Springer, 2017.

12. Armengaud, E., Schneider, D., Brenner, E., & Kreiner, C. Towards Dependability Engineering of Cooperative Automotive Cyber-Physical Systems. In: Systems, Software and Services Process Improvement: 24th European Conference, EuroSPI 2017, Ostrava, Czech Republic, September 6–8, Proceedings (Vol. 748, p. 205). Springer, 2017.
13. Cheng BH, et al. Software engineering for self-adaptive systems: a research roadmap. LNCS. 2009;5525:1–26.
14. Laprie JC. From dependability to resilience. In: 38th IEEE/IFIP Int. Conf. On dependable systems and networks; 2008. p. G8–9.
15. EMC2 project website: https://www.artemis-emc2.eu/. Accessed May 2018.
16. Amorim, T., Ruiz, A., Dropmann, C., & Schneider, D. Multidirectional modular conditional safety certificates. In: International Conference on Computer Safety, Reliability, and Security, p. 357–368. Switzerland: Springer, 2014.
17. Amorim, T., Ratasich, D., Macher, G., Ruiz, A., Schneider, D., Driussi, M., & Grosu, R. Runtime Safety Assurance for Adaptive Cyber-Physical Systems: ConSerts M and Ontology-Based Runtime Reconfiguration Applied to an Automotive Case Study. In: Solutions for Cyber-Physical Systems Ubiquity. Switzerland: IGI Global; p. 137–168.
18. Bennaceur A, et al. Mechanisms for leveraging models at runtime in self-adaptive software. In: Bencomo N, France R, Cheng BHC, Aßmann U, editors. Models@run.time. Lecture notes in computer science, vol. 8378. Cham: Springer; 2014.
19. de Lemos R, et al. Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos R, Giese H, Müller HA, Shaw M, editors. Software engineering for self-adaptive systems II. Lecture notes in computer science, vol. 7475. Berlin: Springer; 2013.
20. Giese H, et al. Living with uncertainty in the age of runtime models. In: Bencomo N, France R, Cheng BHC, Aßmann U, editors. Models@run.time. Lecture notes in computer science, vol. 8378. Cham: Springer; 2014.
21. Joshi R, Chen-Khong Tham. "Integrated quality of service and network management," Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium, 2000, pp. 497. https://doi.org/10.1109/ICON.2000.875848.
22. Li B, Nahrstedt K. A control-based middleware framework for quality of service adaptations. IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms. 1999;17(9):1632–50.
23. Floch J, et al. Using architecture models for runtime adaptability. IEEE Softw. 2006;23:62–70.
24. Tamura G, et al. Towards practical runtime verification and validation of SelfAdaptive software systems. In: de Lemos R, Giese H, Muller HA, Shaw M, editors. Software engineering for self-adaptive systems. LNCS, vol. 7475. Heidelberg: Springer; 2013. p. 108–32.
25. Stoller SD, et al. Runtime verification with state estimation. In: International conference on runtime verification. Berlin, Heidelberg: Springer; 2011.
26. Calinescu, Radu, et al. ENTRUST: Engineering Trustworthy Self-Adaptive Software with Dynamic Assurance Cases. International Conference on Software Engineering. IEEE, 2018.