Department of Distributed and Dependable Systems
Technical Report no. D3S-TR-2014-02
January 25, 2014

Software Engineering for Software-Intensive Cyber-Physical Systems

Ilias Gerostathopoulos, Jaroslav Keznikl, Tomas Bures, Michal Kit, Frantisek Plasil

Charles University in Prague
Faculty of Mathematics and Physics
Prague, Czech Republic

Abstract: In software-intensive cyber-physical systems (siCPS) the interplay of software control with the physical environment has a prominent role. Nowadays, siCPS are expected to (i) effectively deal with the issues of distribution, scalability, and environment dynamicity, (ii) control their emergent behavior, and, at the same time, (iii) be versatile and tolerant in face of changes and threats. Although approaches that individually meet the above requirements of siCPS already exist, their synergy in a comprehensive software engineering framework is far from trivial. In this paper, we pinpoint the important characteristics of engineering siCPS in an attempt to show that they introduce distinct challenges to traditional software engineering. We argue that this can be addressed by a synergy and adaptation of existing models and abstractions, show our proposal towards such a synergy, and discuss its implications.

Version: January 25th, 2014

This work was partially supported by the EU project ASCENS 257414. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7-PEOPLE-2010-ITN) under grant agreement n°264840.

Software Engineering for Software-Intensive Cyber-Physical Systems

Ilias Gerostathopoulos[1], Jaroslav Keznikl[1,2], Tomas Bures[1,2], Michal Kit[1], Frantisek Plasil[1]

[1]Faculty of Mathematics and Physics
Charles University in Prague
Prague, Czech Republic

[2]Institute of Computer Science
Academy of Sciences of the Czech Republic
Prague, Czech Republic

{iliasg, keznikl, bures, kit, plasil}@d3s.mff.cuni.cz

ABSTRACT

In software-intensive cyber-physical systems (siCPS) the interplay of software control with the physical environment has a prominent role. Nowadays, siCPS are expected to (i) effectively deal with the issues of distribution, scalability, and environment dynamicity, (ii) control their emergent behavior, and, at the same time, (iii) be versatile and tolerant in face of changes and threats. Although approaches that individually meet the above requirements of siCPS already exist, their synergy in a comprehensive software engineering framework is far from trivial. In this paper, we pinpoint the important characteristics of engineering siCPS in an attempt to show that they introduce distinct challenges to traditional software engineering. We argue that this can be addressed by a synergy and adaptation of existing models and abstractions, show our proposal towards such a synergy, and discuss its implications.

1. INTRODUCTION

Cyber-physical systems (CPS) are systems of collaborating elements which closely interact with their environment by sensing and actuating. Typically, CPS are characterized by being decentralized, distributed, and heterogeneous.

With the proliferation of smart embedded and mobile devices (smart phones, intelligent cars, etc.) and wireless networks, there is a further trend of CPS becoming large-scale pervasive systems, which combine data from various sources to control real-world ecosystems (e.g., intelligent traffic control, which gathers data about traffic from cars and other sensors in a city and uses them to navigate cars, control the traffic lights, and manage parking allocation). An important feature of these systems is that they are adaptive, in order to adjust to situations in the physical environment, and exhibit emergent behavior (i.e., behavior that comes about as the joint product of behaviors and interactions of many elements of the system). Another important feature of these CPS, is that they are highly dependent on software – they are software-intensive systems [2, 9]. This means that software is by far the most important and the most complex constituent of modern CPS.

Continuous dependable operation of CPS is particularly important as the close connection to the physical environment frequently renders functionality of CPS safe-critical (e.g., operation of the traffic lights in the intelligent traffic control). In addition to being dependable, the software of CPS has to work reliably in the distributed environment and has to be able to adapt to changing situations in the physical environment. Ideally, it should possess some self-awareness and self-healing properties to cope with not fully anticipated situations.

Along the lines above, in this paper we consider a class of CPS that are software-intensive and, at the same time, distributed on a large scale, inherently dynamic, self-adaptive, self-aware, exhibiting emergent behavior, and safety-critical. It is also important to note that these CPS are targeted by the on-going research agendas (e.g., EU framework Horizon 2020). For brevity, we will refer to these CPS as software-intensive CPS (siCPS) in this paper.

We argue that siCPS have a number of specifics, which prevent to fully employ traditional software models and software engineering methods. This calls for tailored models and software engineering abstractions that address and potentially take advantage of the specifics of the siCPS [13]. In fact, siCPS reach the threshold when it is disputable whether we are still dealing with tailored traditional software engineering or whether we are encountering a new paradigm in computing.

As the particular contribution of this paper, (i) we overview these specifics (Section 2) and analyze how they can be addressed by a synergy and adaptation of existing software models and software engineering abstractions (Section 3). On this background, (ii) we give a practical example of such a synergy (Section 1), namely DEECo, an ensemble-based component system [3]. Finally, based on the lessons learned with DEECo, (iii) we discuss potential challenges stemming from the interplay of the models and abstractions in such a synergy (Section 5).

2. SOFTWARE ENGINEERING SPECIFICS OF siCPS

The large-scale physical distribution and interconnectedness with the physical environment makes siCPS rather specific in terms of software engineering (SE). In this section, we overview these specifics from the perspective of SE assumptions and opportunities.

2.1 SE Assumptions Violated in siCPS

A number of assumptions that are typically presumed in traditional SE of general-purpose software systems (GPSS) are violated in siCPS. The assumptions build on the fact that a lot of complexity related to the networking and environment can be in GPSS considered low-level and abstracted away by operating system and middleware. It is of course normal, that even in traditional SE some key assumptions may be violated when developing GPSS with special needs (e.g., high-availability, open-endedness). Nevertheless, siCPS are clearly distinct in this context by the large number of such violated assumptions

Therefore, below we identify and discuss a number of assumptions in traditional SE of GPSS that we deem to have a

significant simplifying effect on software development but – according to our experience – cannot be preserved in engineering siCPS:

A1 *Static physical structure* – Even though data and code are subject to mobility in GPSS, the physical nodes where the code is running are typically stationary. In siCPS, the physical substratum is continuously evolving, as nodes move in the physical environment. The fundamental challenge is how to map the ever-changing substratum to the network of computational nodes so that stringent requirements on the desired services are always met.

A2 *Location obliviousness* – The cost and profit of reaching a particular node is typically not significantly influenced by its physical location. This independence facilitates creation of open-ended and dynamic distributed GPSS and is generally considered an asset. In siCPS, locality of peer nodes is a fundamental design constraint, since physical proximity directly affects reachability and connectivity on one hand and functional correctness on the other.

A3 *Reachability (clique connectivity)* – GPSS typically rely on the Internet network stack for the underlying communication protocols (Internet-based systems [8]). This means that with high probability any node can successfully establish point-to-point communication links with any other node in the system. In siCPS there is no such guarantee, as nodes often operate over dynamic networks lacking a permanent infrastructure, such as mobile ad-hoc networks (MANETs). This limitation imposes a fundamental constraint in the design of siCPS, since nodes are expected to operate in full autonomy, even detached from their peers.

A4 *Stable connections* – In most GPSS, on top of being able to reach and connect to remote subsystems, connections are typically considered stable. This is manifested in handling of communication errors in such systems: errors are considered as exceptions that have to be handled accordingly. In siCPS errors in communication are the rule, not the exception. Thus, they can no more be handled as exceptions. The property of unstable connectivity has to be acknowledged and ideally reflected in the employed SE abstractions.

A5 *Availability of global state* – Reasoning over the global state of a distributed system is a requirement for many applications. Although techniques exist for traditional distributed GPSS (e.g., distributed consensus), they are not directly applicable to siCPS because of the loose connectivity among the nodes. Also, since the local state in siCPS evolves continuously with the physical environment, attaining global state is generally infeasible.

A6 *Marginality of real-time aspects* – GPSS typically do not impose hard real-time constraints on their operation and communication. When time matters (e.g., internet-based video streaming applications), it is mostly because late responses may impede system performance rather than correctness. In siCPS, the passage of time becomes a central feature of system behavior and design, since stringent notion of time is fundamental for measuring, predicting and controlling properties of the physical environment.

A7 *Crisp consistency* – In traditional distributed GPSS, there is a crisp notion of data consistency – the data is either consistent or not (this includes also eventual consistency etc.). On the other hand, in siCPS, where strict distributed synchronization becomes too expensive, such interpretation of consistency is not desirable. Rather, in siCPS it is important to quantify and/or guarantee a degree of consistency [1].

A8 *Controlled dynamism* – Many GPSS are dynamic in the sense that they dynamically adapt to changes and recover from malign states. The observation is, though, that the dynamism is typically a result of actions initiated by the system itself or its administrator. On the contrary, in siCPS dynamism is inherent, imposed by the physical environment itself. Thus, siCPS need to detect and recover from contingent and often unforeseen situations in their environment in a non-disruptive way and without supervision (they have to be self-aware and autonomic).

A9 *Focus on reactive behavior* – Outputs of a GPSS are typically reactions to explicit stimuli, such as service requests and internal/external events (e.g., computation is initiated as a response to user input). Instead of waiting for an event, siCPS have to operate continuously (e.g., continuously monitoring the environment). This means that the notion of "striving to achieve" is central in system behavior. In that sense, siCPS have to constantly react to and also perform changes based on properties that are either sensed or predicted. Relying on simple (e.g., rule-based) reaction patterns in siCPS is insufficient, since it may lead to oscillations and instability.

A10 *Stateful communication* – GPSS usually assume stateful communication in the communication protocols they employ. This enables effective synchronization among distributed components. Moreover, since stable connections are assumed (A4), errors are treated as exceptional and detected and solved via explicit error recovery. In siCPS, stateful communication does not scale. In fact, extreme network dynamism, typical for siCPS, may incur recurrent error recovery.

2.2 SE Opportunities in siCPS

As pointed out in Section 2.1, none of the discussed assumptions can be generally presumed in siCPS. This makes it a non-trivial challenge to develop siCPS by applying traditional SE methods. However, it would be wrong to perceive all specifics of siCPS as impeding their development, since they may provide opportunities for getting around the violated assumptions. In this perspective, it is desirable to take advantage of such siCPS specifics instead of aiming at adapting traditional SE methods, e.g., building a complex middleware to provide a traditional programming model.

Again, to pinpoint this idea, we have compiled a list of specifics, which we believe can be advantageously exploited in addressing the violated assumptions. Despite of not being complete, we believe that this list still gives an important research direction for siCPS design methods:

O1 *Physical mobility* – Devices used in siCPS span from stationary to portable and mobile ones. Computational nodes deployed on mobile devices can carry information while moving. This contributes to the overall connectedness of the system, as a mobile node covers a much bigger physical area while moving, and can effectively spread the information in the area and connect otherwise disconnected network partitions. For example, a vehicle moving along a street segment can aggregate temperature data measured

from independent sensors positioned in the tarmac along its route (which themselves cannot reach any external network), and publish the data on a remote server, or spread it to other vehicles in the vicinity.

O2 *Physical locality* – The fact that devices in siCPS are physically close provides a natural way to partition the system into subsystems based on geographical location. This is, again, special to siCPS; general-purpose systems are rarely partitioned based on the physical location, because of the otherwise useful assumption on location obliviousness. Having such a natural partitioning can be easily exploited to achieve high levels of scalability.

O3 *Location-dependency of data* – Data in siCPS are often location-dependent, meaning that the value of certain measureable system attributes depend on the physical location of the sensors that provide the raw data. This dependency, in combination with the physical proximity of sensor nodes, allows for data sharing and reuse among nearby nodes and has the potential to contribute to system robustness (in face of sensor failures, etc.).

O4 *Physical laws in data evolution* – Since siCPS operation typically involves sensing physical-environment properties (e.g., position, battery capacity, temperature), one can take advantage of the physical laws that govern the evolution of the values of such properties to estimate/predict their real values. In effect, a value that is slightly stale can still be used, if certain safety bounds on its evolution in time can be established [1]. As an example, consider a wireless-based adaptive cruise control system: a stale value of the front vehicle's position can still be used by the rear vehicle's cruise control, since it is possible to estimate the actual position based on the maximum and minimum vehicle acceleration, typically provided by car manufacturer.

3. APPROACHES THAT PARTIALLY REFLECT THE SPECIFICS OF siCPS

To the best of our knowledge, there are no comprehensive methods nor supporting models that address the specifics of siCPS in their entirety. Nevertheless, our experience shows that some SE approaches target these specifics at least partially. In this section we provide a short overview of such approaches (summarized in Figure 1), with the goal to later show how they can be combined in a comprehensive framework.

Agent-based systems. In order to deal with dynamicity in siCPS, one can be inspired by *autonomous agents*. This abstraction brings conceptual autonomy to the loosely coupled system parts. Each part is designed to operate with a partial view of the whole system, beneficial when the global state is not available (*A5*). For example, in the Belief-Desire-Intention (BDI) architectural model [20], agents maintain a *belief* about the rest of the system to guide their autonomous decisions. In addition, *multi-agent systems* [23] feature the concepts of agent roles and groups, which bring the autonomy to architecture organization and allow building *self-organized systems* that do not rely on the assumptions of controlled dynamism (*A8*) and static physical structure (*A1*). An important problem is that industrial agent implementations do not translate the conceptual autonomy and the other useful agent notions (goals, intentions, roles, groups) into proper software engineering constructs that satisfy real-life requirements on autonomous behavior. In particular, they still rely on the assumption of relatively stable bindings between the agents (*A4*), which is not plausible in most siCPS.

Absence of:	Agent-based systems	Ensemble-based systems	MANET & gossip protocols	Real-time & control systems	Synergy in DEECo
A1 Static physical structure		+	+		+
A2 Location obliviousness		+	+		+
A3 Reachability		+	+		+
A4 Stable connections	-		+	-	+
A5 Availability of global state	+	+			+
A6 Marginality of real time asp.				+	(+)
A7 Crisp consistency		-		+	(+)
A8 Controlled dynamism	+	+	+	-	+
A9 Focus on reactive behavior			+	+	+
A10 Stateful communication				+	+

Figure 1: Overview of the siCPS specifics addressed "+", partially addressed "(+)", and relied upon "-" by the approaches from Section 3 and the synergy in DEECo.

E**nsemble-based systems.** Another important specific of siCPS is the opportunistic fashion of operating in a dynamic environment at a massive scale. To this end, the paradigm of *attribute-based communication* in *ensemble-based systems* has recently gained attention [18]. Here, the target of communication is determined according to the values of its attributes rather than by a direct identifier. This paradigm can be exploited to model a best-effort, dynamic coordination of components, effectively dealing with cases when the assumptions of static physical structure (*A1*), reachability (*A3*), and controlled dynamism (*A8*) are violated. However, the application of this paradigm typically relies on explicit and crisp handling of data consistency (*A7*).

MANET and gossip protocols. At the network layer, extensive research in the areas of *mobile ad-hoc networks* (MANETs) has resulted into a number of routing protocols (see [17] for a comprehensive review), which are able to operate over infrastructure-less dynamic networks. In MANETs, each node acts not only as a host, but also as a router. Node mobility is assumed, resulting in dynamically changing network topology. As such, MANET protocols lift the assumption of static physical structure (*A1*) and work even when the reachability assumption (*A3*) is violated, thus becoming very relevant to siCPS. Moreover, MANET protocols lift the assumption of location obliviousness (*A2*), as they enable position-based packet routing [16] (sometimes called *geocast* routing [15]). A promising synergy for siCPS is to combine geocast protocols at the network layer with *gossip* protocols at the data dissemination layer, effectively enabling proactive, opportunistic communication (*A9*) in MANETs [8]. Integration of gossiping brings a remedy in cases of unstable connections (*A4*) and inherent dynamism (*A8*).

Real-time and control systems. As to strong interaction with physical environment, many techniques already exist in the domain of *embedded real-time systems* [5] and *software control systems* [19]. Such techniques promote proactive behavior (*A9*) and focus on real-time attributes (*A6*). They employ control feedback loops, which continuously maintain the operational normalcy (stability) of a system by adequate scheduling of periodic tasks. These techniques stand as a promising way to

handle data outdatedness in absence of crisp consistency interpretation (*A7*) in siCPS, by effectively setting the bounds that define the range of normal system operation. Communication in embedded real-time systems is also typically stateless (*A10*), consider, e.g., data publishing on CAN bus. Nevertheless, real-time analysis and design typically rely on the assumption of predictable environment, which itself relies on controlled dynamism (A8) and stable connections (A4) assumptions.

4. DEECo: A SYNERGY

In order to evaluate the potential for a synergy of the approaches discussed in Section 3, as a particular example we present DEECo [3, 11] – an *Ensemble Based Component System* that we have proposed specifically for architecting siCPS.

In DEECo, we take the approach of adopting the component-based development (CBD) as the basic substratum on top of which we embed selected SE approaches from Section 3. Our adoption of CBD originates from the necessity of reuse, encapsulation, and separation of concerns so as to tame the complexity of building and maintaining large applications [6]. In CBD, and thus also in DEECo, systems are built around well-defined architectures based on a composition of components, which themselves are seen as encapsulated, reusable, and substitutable entities.

In the remainder of this section, we describe the individual constituents of the DEECo component model with focus on how we approached the synergy. We refer the interested reader to [3] for a detailed technical description of DEECo and for the formal semantics of DEECo. Also, a Java implementation is available[1].

4.1 Component

Adopting the ideas of agent-based and self-adaptive systems, the concept of *component* in DEECo is centered around the features of autonomy, self-adaptation, and belief (A5). Specifically, a component is an autonomous, encapsulated, and composable software entity constituting its own state and behavior.

As is typical for software agents, component state is expressed in terms of *knowledge* (e.g., line 3 in Figure 2). Note that in DEECo, all the data accessible to a component is referred to as knowledge. In alignment with the BDI architectural model, knowledge of a component comprises both the private component state (e.g., calendar) and the component's belief about the rest of the system (e.g., parkingAvailability). As a slight difference from traditional BDI approach, rather than being updated explicitly by the component itself, the belief is updated automatically (by the execution environment, Section 4.3) as a result of component composition (Section 4.2). This decision further stresses the component's autonomy and separation of concerns.

The behavior of a component is represented by a set of *processes* (e.g., lines 4-7 in Figure 2). Following the notions of control systems and self-adaptive systems, a process is essentially a feedback loop, continuously and proactively maintaining operational normalcy of the component (A9). At the same time, each process executes concurrently, independently of the other processes, i.e. it atomically reads its inputs, executes its body, and atomically writes its outputs. Specifically to DEECo, a process operates strictly upon the knowledge of the corresponding component; it may thus interact with other components only through the (externally updated) belief (A4, as there is no "direct" communication among components).

[1] https://github.com/d3scomp/JDEECo

```
1.  component Vehicle
2.    knowledge:
3.      calendar, parkingAvailability, plan, ...
4.    process computePlan(in calendar, in parkingAvailability, out plan):
5.      function:
6.        plan ← JourneyPlanner.computePlan(calendar, parkingAvailability)
7.      scheduling: periodic( 5000ms )
8.    ...
9.
10. component ParkingLot
11.   knowledge:
12.     position, availability, ...
13.   process monitorAvailability(out availability):
14.     ...
15.
16. // updates Vehicle's belief about availability of all ParkingLots along the
    route
17. ensemble UpdateAvailabilityInformation:
18.   coordinator: Vehicle
19.   member: ParkingLot
20.   membership:
21.     ∃ event ∈ coordinator.calendar:
22.       distance(member.position, event.position) ≤ TRESHOLD
23.   knowledge exchange:
24.     coordinator.parkingAvailability ←
        members.reduce(member.availability)
25.   scheduling: periodic( 2000ms )
```

Figure 2: Example of a DEECo component and ensemble definition in a DSL.

4.2 Component Composition

For component composition we adopt the approach of ensemble-based systems and multi-agent systems by employing autonomic self-organization of components into component *ensembles* (in multi-agent systems called groups). This self-organization is based on a declarative representation of a component's membership in an ensemble, based on the component's context (A1 and A3). In order to distinguish in which ensemble the membership is being decided upon, every ensemble has a *coordinator*. Membership in an ensemble with a given coordinator is based on whether a component is able to assume the role of a *member* w.r.t. the coordinator. This is technically expressed via a *membership condition*, which decides whether two given components can form a coordinator-member pair. Following the idea of attribute-based communication, the membership condition is defined upon the attributes (i.e., knowledge exposed for this purpose) of the components in question (e.g., lines 21-22 in Figure 2). Note, that the ensemble definition is generic and determines ensemble instantiation for each group of components meeting the membership condition (w.r.t. particular coordinator). Also, a component can be a member or coordinator of multiple ensembles at the same time.

Within an ensemble, we adopt the idea of stateless, proactive communication employed in control systems and gossip-based systems (A9 and A10). Specifically, the communication takes the form of stateless *knowledge exchange*, objective of which is to recurrently and proactively update the belief of the components within the ensemble based on a given prescription (e.g., line 24 in Figure 2). This form of communication well aligns with the proactive, cyclic execution of component processes. Note, that the statelessness and proactivity make knowledge exchange suitable for cases of faulty connections (A4) and inherent dynamism (A8).

4.3 Execution Environment

The main task of the DEECo execution environment is performing knowledge exchange in a distributed setting. For this purpose, we combine the protocols for geographical routing in MANETs with gossip protocols so as to enable location-aware communication of belief (A2) in mobile ad-hoc environments (A1 and A3) with unstable connections and inherent dynamism (A4 and A8). Specifically, the execution environment proactively advertises the knowledge of a (source) component to all the other potentially-interested (target) components via a geocast protocol. Then, in case the source and target components meet the membership condition of an ensemble, the execution environment updates the belief of the target component according to the knowledge exchange prescription of the ensemble.

Adopting the approach of embedded real-time systems, the execution environment is also responsible for execution of component processes and activities related to knowledge exchange in a (soft) real-time fashion (A6 and partially A7), featuring both periodic and event-based scheduling.

5. DISCUSSION OF IMPLICATIONS

Engineering siCPS with the basic building blocks (autonomous components, ensembles) offered by the proposed synergy in DEECo offers several advantages, but also poses new challenges. As seen in Figure 1, DEECo addresses all of the identified challenges of A1-A10, which we deem a step forward. Certainly, there could be other assumptions, e.g., predictability of underlying platform and global synchronization of beliefs, which still remain to be addressed. Building on our experience in applying the ensemble-based component system paradigm to two real-life case studies, namely the intelligent vehicle navigation [3] and the firefighter tactical coordination [4], this section discusses the implications of merging different methods.

Exploitation of the opportunities. A close synergy of geocast MANET protocols and attribute-based communication, and an integration of membership evaluation and routing in particular, allows exploiting new opportunities based on physical locality (O2) and location-dependency of data (O3) (i.e., membership can effectively exploit physical location). Further, the proactive gossip-based advertisement of belief enables exploiting the physical mobility (O1). The cyclic and real-time nature of component processes also facilitates use of models that estimate/predict the safety bounds of knowledge evolution [1]. This is done by exploiting the physical laws that govern evolution of certain knowledge values (O4).

Components as autonomous agents. Borrowing the ideas of belief and autonomous operation from agent-based systems and coupling them with the encapsulation and deployment facilities of component-based systems results into a dependable platform for robust component-based agent implementations. The robustness is achieved by grafting such "agents" with implicit component binding and communication. Contrary to other agent-based frameworks, the autonomous components thus do not communicate directly, e.g., via sending messages; instead, component knowledge serves as a communication medium. A component's belief, i.e., the part of its knowledge that reflects knowledge of other components, plays a role of "smart" sensors and actuators. For instance, a belief could represent a "smart sensor" providing "positions of up to 10 closest parking lots, which are available". All in all, a component's belief is updated externally – via knowledge exchange handled by execution environment.

Stateless interaction. Adopting the idea of attribute-based communication in component interaction has many advantages when considering that components in siCPS recurrently appear and disappear and form dynamic groups of best-effort coordination. At the same time, having no means of direct component binding and addressing makes it challenging – but certainly not impossible, as we have observed – to realize some forms of protocol-based communication. This is essential in certain interactions, e.g., reserving of a parking place by a specific vehicle at a specific parking lot. Stateless interaction dictates knowledge design in a way that it is always possible to reconstruct the state of the session from the knowledge. For instance in parking reservations, this can be achieved by assigning each reservation request a globally unique identifier (GUID), so that a reservation response could refer to it.

Embedded feedback loops. When designing siCPS, special means have to be provided for feedback loops. By building on the ideas of control-based and real-time systems, the proposed synergy in DEECo embeds the feedback loop operation both at design time and runtime. Systems based on feedback loops typically require a description of operational normalcy, usually in terms of periodic scheduling of tasks. However, adoption of this idea needs a paradigm change in the design process, to explicitly focus on normalcy each process is expected to maintain as opposed to goals to be achieved [10].

Decentralized operation. Coupling best-effort data dissemination of MANET protocols with attribute-based communication and decentralized system operation can result in situations when different parties act based on inconsistent local beliefs – so-called split-brain scenarios. For instance, a component can assume to be a member of an ensemble, while the ensemble's coordinator does not recognize this situation (or vice-versa). This behavior is in a way inevitable, however it has to be accounted for in the design, e.g., by making components only weakly synchronized or by relying on an underlying network or physical environment to provide some guarantees (thus making these split-brain situations temporary with an upper bound for duration).

Ensembles as component connectors. The duality between components and ensembles resembles the classical problem of components and connectors – especially whether connectors are only special types of components and what functionality should be in connectors and what functionality should be in components. In particular, this holds when connectors comprise complex adaptation logic. In DEECo, though, this problem is partially remedied by distinguishing that (i) although stateful, a component has a direct access solely to its local knowledge, (ii) an ensemble embodies only stateless exchange of knowledge among its member components. This is a strong conceptual difference pushing ensembles into the role of simple connectors and components in the role of entities performing the actual computation and data aggregation.

Parallel process execution. The physical world is inherently concurrent. Software engineering abstractions for engineering siCPS have to deal with concurrency by allowing execution of processes in parallel. This leads to challenges related to handling of shared resources, which, if not dealt with, can result into race conditions, deadlocks, etc., effectively jeopardizing the safety of the system. Similar to actor-based design, where the exchanged data are considered immutable, DEECo avoids introducing any dedicated synchronization constructs. Rather, it employs the simple semantics of atomically operating over knowledge while

applying the rule of single-writer for each knowledge field. The downside of the approach is that it sometimes leads to the necessity of having a special "aggregation" process in a component, which merges data coming from different sources (similar situation happens in actor-based approaches as well). However, this seems a reasonable price to pay for having race-conditions prevented by design.

6. RELATED WORK

Since CPS is an emerging class of systems, there are multiple research efforts trying to shed light on the state of the art and the challenges ahead [12, 22]. Unfortunately, not as many solutions are proposed, especially when considering guidance via proper software engineering abstractions specific to CPS. Our work aims at highlighting the problems in CPS software engineering, while, at the same time, proposes solutions to these problems and evaluates their implications. In the same spirit, in [7], the authors focus on the challenges of modeling CPS caused by the intrinsic heterogeneity, concurrency, and sensitivity of such systems. Backed up by a hybrid-system-modeling environment called Ptolemy II, their approach emphasizes determinism and predictability in modeling and simulations of safety-critical CPS. In [13], the author reviews the requirements/specifics of CPS and identifies the absence of timing behavior in core abstractions in computing as the main impediment in developing future CPS. In our work, we focus on the subset of CPS that is software-intensive, where structural models and systematic engineering methods become more relevant.

Our aim at a synergy can be compared to frameworks proposed for self-adaptive/self-organizing systems, e.g., [21], and autonomic agent-based systems, e.g., [14]. In [21], the authors propose a synergy of self-organization, agent-inspired autonomy and rule-based reasoning into a service-oriented architectural framework. Their approach is centered around the concepts of self-describing components, component metadata and interaction policies executed at runtime, resembling the concepts of components, component knowledge and ensembles, respectively. In [14], the authors present a component-based framework for autonomic agents building on agent-based middleware infrastructure. The difference from these and other similar approaches lies in the fact that we deal with the specifics of siCPS, where unreliable communication and extreme dynamism loom large.

7. CONCLUSION

Building software for software-intensive cyber-physical systems (siCPS) is far from trivial. In this paper, we attempted to pinpoint the challenges and pitfalls associated with applying traditional software engineering (SE) methods in siCPS and to show how these challenges can be met by a comprehensive synergy and adaptation of existing SE models, methods and abstractions. This we exemplified on the DEECo component model. The evaluation of the proposed synergy in DEECo outlines a number of interesting research topics in terms of addressed and unaddressed issues, such as design based on maintaining operational normalcy.

8. REFERENCES

[1] Ali, R. Al, Bures, T., Gerostathopoulos, I., Keznikl, J. and Plasil, F. 2014. Architecture Adaptation Based on Belief Inaccuracy Estimation. *To appear in Proc. of WICSA'14* (Apr. 2014).

[2] Beetz, K. and Böhm, W. 2012. Challenges in Engineering for Software-Intensive Embedded Systems. *Model-Based Engineering of Embedded Systems*. Springer. 3–14.

[3] Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M. and Plasil, F. 2013. DEECo – an Ensemble-Based Component System. *Proc. of CBSE'14* (Jun. 2013), 81–90.

[4] Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plasil, F. and Plouzeau, N. 2014. Adaptation in Cyber-Physical Systems: from System Goals to Architecture. Charles University in Prague, TR no. D3S-TR-2014-01.

[5] Buttazo, G., Lipari, G., Abeni, L. and Caccamo, M. 2005. *Soft Real-Time Systems: Predictability vs Efficiency*. Springer.

[6] Crnkovic, I. 2002. *Building Reliable Component-Based Software Systems*. Artech House, Inc., Norwood, MA.

[7] Derler, P., Lee, E. a. and Vincentelli, a. S. 2012. Modeling Cyber–Physical Systems. *Proceedings of the IEEE*. 100, 1 (Jan. 2012), 13–28.

[8] Friedman, R. and Gavidia, D. 2007. Gossiping on MANETs: the Beauty and the Beast. *ACM SIGOPS Operating Systems Review*. 41, 5 (Oct. 2007), 67–74.

[9] Hölz, M., Rauschmayer, A. and Wirsing, M. 2008. Engineering of Software-Intensive Systems. *Software-Intensive Systems and New Comp. Parad.* Springer. 1–44.

[10] Keznikl, J., Bures, T., Plasil, F., Gerostathopoulos, I., Hnetynka, P. and Hoch, N. 2013. Design of Ensemble-Based Component Systems by Invariant Refinement. *Proc. of CBSE'13* (Jun. 2013), 91–100.

[11] Keznikl, J., Bures, T., Plasil, F. and Kit, M. 2012. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. *Proc. of WICSA'12* (Aug. 2012), 249–252.

[12] Kim, B.K. and Kumar, P.R. 2012. Cyber–Physical Systems: A Perspective at the Centennial. *Proceedings of the IEEE*. 100, Special Centennial (2012), 1287–1308.

[13] Lee, E.A. 2008. Cyber Physical Systems: Design Challenges. *Proc. of ISORC'08* (May 2008), 363–369.

[14] Liu, H., Parashar, M. and Hariri, S. 2004. A Component Based Programming Framework for Autonomic Applications. *Proc. of ICAC'04* (May 2004), 10–17.

[15] Maihofer, C. 2004. A Survey of Geocast Routing Protocols. *Communications Surveys & Tutorials, IEEE*. 6, 2 (Apr. 2004), 32–42.

[16] Mauve, M., Widmer, A. and Hartenstein, H. 2001. A Survey on Position-Based Routing in Mobile Ad Hoc Networks. *Network, IEEE*. 15, 6 (Nov. 2001), 30–39.

[17] Natesapillai, K., Palanisamy, V. and Duraiswamy, K. 2012. A Review of Broadcasting Methods for Mobile Ad Hoc Network. *International Journal of Advanced Computer Engineering. Serial Publications, India*. (Sep. 2012).

[18] De Nicola, R., Ferrari, G., Loreti, M. and Pugliese, R. 2013. A Language-based Approach to Autonomic Computing. *Formal Methods for Components and Objects*. Springer Berlin Heidelberg. 25–48.

[19] Patikirikorala, T., Colman, A., Han, J. and Wang, L. 2012. A systematic survey on the design of self-adaptive software

systems using control engineering approaches. *Proc. of SEAMS'12* (Jun. 2012), 33–42.

[20] Rao, A. and Georgeff, M.P. 1995. BDI agents: From theory to practice. *Proc. of the First International Conference on Multi-Agent Systems* (1995), 312–319.

[21] Serugendo, G.D.M., Fitzgerald, J. and Romanovsky, A. 2010. MetaSelf – An Architecture and a Development Method for Dependable Self- * Systems. *Proc. of SAC'10* (Mar. 2010), 457–461.

[22] Sha, L., Gopalakrishnan, S., Liu, X. and Wang, Q. 2008. Cyber-Physical Systems: A New Frontier. *Proc. of SUTC'08* (Jun. 2008), 1–9.

[23] Shoham, Y. and Leyton-Brown, K. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press.