Service Virtualization for Self-Adaptation in Mobile Cyber-Physical Systems

Amir Taherkordi[1,2], Peter Herrmann[1], Jan Olaf Blech[3], and Álvaro Férnandez[1]

[1] NTNU Trondheim, Norway
{amirhost,herrmann,alvarof}@item.ntnu.no
[2] University of Oslo, Norway
amirhost@ifi.uio.no
[3] RMIT University, Melbourne, Australia
janolaf.blech@rmit.edu.au

Abstract. Mobile Cyber-Physical Systems (mCPS) consist of cooperating units that often operate in an unpredictably changing environment. Thus, they need to adapt quickly to varying *spatial* and *temporal* conditions during operation, e.g., to avoid collisions. The control software of the mobile units has to reflect this complex dynamics, and traditional device-level adaptation models are usually not efficient enough to engineer them smoothly. We address this challenge by proposing a *Virtual Adaptation Services Framework (VASF)*. It provides a virtualized application-level view to adaptation requirements, enabling adaptation coordination between cooperative mCPS devices. In particular, the VASF allows us to describe the contextual conditions of mCPS by abstract rules that are analyzed at runtime by the tool-set BeSpaceD. Based on this analysis, the control systems of the involved mCPS units are automatically reconfigured using the OSGi framework. The approach is demonstrated with DiddyBorg robots that are operated by Raspberry Pi boards.

Keywords: Mobile CPS, spatiotemporal reasoning, virtualized adaptation services

1 Introduction

A *Cyber-Physical System* (CPS) integrates a physical mechanism with a computer-based control and monitoring system. Usually, it contains feedback loops in which the physical processes affect embedded computer systems and vice versa [10]. Being tightly integrated with physical processes, a CPS is not always predictable and does not necessarily operate in a controlled environment [25]. Therefore, it needs to respond and adapt quickly to changes during operation, such as hardware and software defects, changes in resource use efficiency and surrounding environments, unexpected conditions, and non-continual feature usage.

This concern is particularly important for *mobile Cyber-Physical Systems* (mCPS), in which a unit moves freely in its environment and may cooperate with other units, e.g., in mobile robotics and smart vehicles [37]. The mCPS essentially run in highly dynamic environments (e.g., coordination of autonomous vehicles). In consequence, they need to adapt their behavior in a collaborative manner at runtime. The kind of adaption that

mCPS software systems may perform can be the dynamic allocation of resources [22], content adaptation [26] (e.g., multimedia content), or the adaptation of the structure and functionality of software, to name the most significant ones. Among these, dynamic and autonomous adaptation of mCPS software is considered the most difficult aspect to implement. It involves the deployment of software modules and reconfiguration of networks and software architectures at runtime, as well as the adaptation of mCPS control software parameters (e.g., the speed of a unit) [13].

To coordinate a bunch of mCPS units and to adapt them to a varying environment in a timely manner, we need efficient *context reasoning* solutions that guide the reconfigurations carried out in the various devices. The control systems of the involved mCPS have to consider numerous *spatial* and *temporal* contextual conditions (e.g., humans in the vicinity). This and the potentially large number of cooperating units makes context processing and self-adaptation of mCPS software a non-trivial design problem for which traditional device-centric adaptation approaches are hardly suited [36].

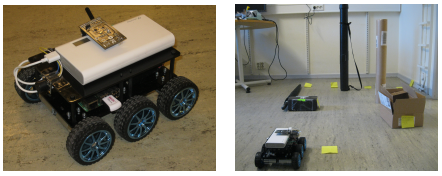
The challenge to create adaptable mCPS software systems has received growing attention by the research community. Existing work addressed this challenge within specific application domains, e.g., robotics and smart automotive systems [14]. Most proposed solutions are either devoted to low-level reconfiguration issues for adaptation (e.g., component-based frameworks [11]) or focus on specifying the CPS adaptable behavior through high-level language abstractions, such as Domain-Specific Languages (DSL) [33, 34]. However, the state-of-the-art has not sufficiently addressed the aforementioned complexity issues in modeling and developing adaptive mCPS.

We propose a self-adaptation service framework for mCPS addressing the complexity in modeling and adapting to highly dynamic location and time aspects of cooperative mCPS, called the *Virtual Adaptation Services Framework* (VASF). In contrast to device-oriented approaches, it allows us to define adaptation requirements for a whole system of cooperating mCPS units. In particular, we can model spatiotemporal system properties using a set of rules that relate contextual aspects to reconfiguration and other change tasks. VASF uses the tool-set BeSpaceD [3] to reason about which rules to apply in a certain context. The result of this analysis is a number of adaptation actions that are automatically executed at the involved mCPS units. The provision of a rule-based reference context model eases the modeling of simultaneous and complicated contextual conditions, while the automatic runtime operation of VASF facilitates software reconfigurations on cooperating mCPS devices. We demonstrate the framework implementation on DiddyBorg robots [30] operated by Raspberry Pi boards.

The rest of this paper is organized as follows: In Section 2, we introduce a motivating scenario followed by the proposed adaptation framework in Section 3. We present related work in Section 4 and conclude in Section 5.

2 Motivating Scenario

Autonomous mobile robots are getting popular in various application domains ranging from healthcare to warehousing and transport (e.g., autonomously operating cars and stellar rovers). Like mCPS in general, the robots operate collaboratively in varying physical environments that may have different contextual properties. For instance,



(a) (b)

Fig. 1: (a) A DiddyBorg robot and (b) its testing environment

transport robots can face changing indoor and outdoor environments ranging from sterile "highways" in which they are protected from conflicting obstacles to areas crowded by humans, stationary barriers, and other robots which may be on conflicting courses. The differing environments aggravate the construction of a correct and timely working control software since, for example, the sensors to be used need to change constantly. In the following, we list some typical contextual environment properties:

- **Surface:** The state of the surface, a mobile robot operates on, may effect relevant control parameters (e.g., braking distances are larger on wet surfaces than on dry ones). Further, the surface may influence the sensors to be used since, for instance, some sensors tend to provide wrong readings when the robot movement is bumpy.
- **Network access:** Larger distances and obstacles may impede the communication between robots operating in the same area which, e.g., may lead to a delayed reaction when two robots are on a conflicting course (see [18]).
- **Self-localization:** For many tasks, it is important for a robot to know where it is (see, e.g., [24]). The way to find the current position, however, depends on various aspects (for example, GPS can only be used outdoors; for triangulation, a number of beacons have to be located at the right spots).
- **Coordination and collision prevention:** For the control, it is relevant whether a robot operates alone in a sterile environment or if it has to coordinate with other robots. A relevant use case is, of course, the protection of humans that may be severely hurt when colliding with a robot.
- **Energy consumption:** Many robots are run by batteries, and for a sustainable operation it is often relevant to keep energy consumption as low as possible. For instance, it may be useful to switch off sensors at the side and the back of a robot if it operates straight ahead for a while.

When a robot moves between locations for which the above mentioned contextual properties differ, it has to adapt its control software. If, moreover, several robots collaborate, the adaptations may embrace the control systems of all of them.

Our testbed consists of DiddyBorg robots [30], see Fig. 1. That are affordable and easily manageable demonstrators, each consisting of six motors that are controlled by a Raspberry Pi. To use DiddyBorg robots in practice, we provided them with sensors in order to enable self-localization and to avoid collisions with fixed and moving obstacles. In our current arrangement, we test the use of infrared and ultrasound sensors combined with an accelerometer and a magnetic sensor determining the direction of the robot.

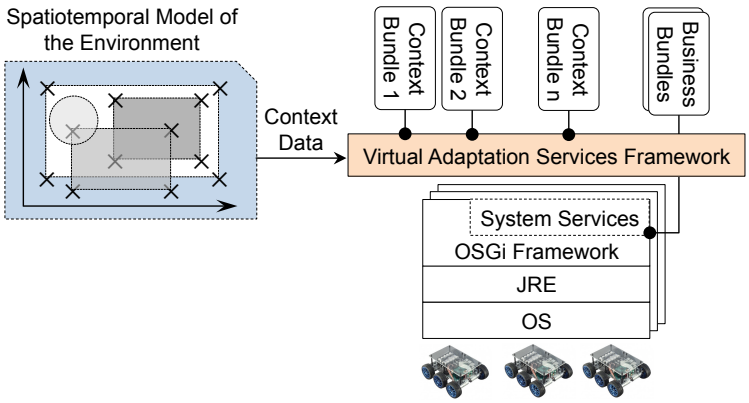


Fig. 2: Overview of our virtualization-based approach for self-adaptation in mCPS

Moreover, the robots are provided with sensors for temperature and air pressure which enables us to adapt their behavior in, e.g., icy conditions. Infrared sensors are suitable for very close distances until 80 cm. They fit nicely to the ultrasound sensors which cannot render precise measures to obstacles closer than 60 cm but provide good distance sensing of more distant objects of up to about 5 m. Using all these sensors, we have to prevent straining the batteries of a DiddyBorg robot. Its motors are operated by 10 AA cells connected in series while we use a rechargeable 5V battery to drive the Raspberry Pi and the sensors. We consider the following dynamic adaptations beneficial:

- To save energy, it seems sensible not to use all sensors at once but to start and stop them according to the environment, the robot operates in, and to adapt the control software accordingly.
- Typical controller parameters like maximum speed and protective braking distances shall be adjusted depending on the physical location and other properties of the environment. For instance, when the robot operates in a long and empty corridor, and the ultrasound sensor detects no obstacle within three meters, the robot can operate with full speed. When enclosing an obstacle, the speed may be reduced depending on the distance to the obstacle.
- We also consider special constructional features which might affect the quality of the sensor readings (e.g., we found out that the heating pipes in one of our labs falsify the readings of the ultrasound sensors which render wildly shaking values when the pipes are approached).
- Temporary conditions in an environment can occur, e.g., due to spills or accidents, that afford extra caution or to avoid an area entirely.

Our VASF-based solution introduced below, shall address these adaptation types.

3 Adaptation Framework

The main goal of the VASF is to provide a unified abstraction for adaptation in mCPS in order to reduce the complexity of modeling spatiotemporal-related contextual changes

of the control software of mCPS. In particular, it allows us to express the contextual aspects of a mCPS that may consist of various cooperating units, with a set of easily understandable rules. Each rule allows the user to relate contextual information to context change tasks. Based on sensor inputs and other existing context data, a *context reasoning system* driven by BeSpaceD [3] analyzes the rules and sorts out potential conflicts between them. The result of the analysis is a set of reconfigurations that VASF automatically forwards to the involved units. They may vary from basic parameter-based reconfiguration of software (e.g., switching from one localization sensor type to another type) to service-level software replacements (e.g., replacement of a energy-draining localization software service with a less powerful but more energy-efficient one).

Figure 2 illustrates the main design elements of VASF. That includes: i) the context modeling space which represents the mCPS operation environment; ii) the VASF which resides on the software system of mCPS devices and offers the context monitoring and processing, and reconfiguration functionality. Reconfiguration is performed through services relying on the Java-based OSGi framework [28]. From the developer's viewpoint, the VASF provides a unified virtualized view to the mCPS-level adaptation needs which will be further detailed for each mCPS device. The different design aspects of the VASF are introduced below.

3.1 Spatiotemporal Context Modeling and Reasoning

For context modeling and reasoning, we propose the use of our BeSpaceD language and tool [3] for spatiotemporal modeling and reasoning. Spatiotemporal models provide a formal view of the context modeling space that robots are operating in. Such models can include *geometric* information like obstacles, regions with specific operating conditions as well as *topological* information such as interconnection information between different regions that do not depend on a specific geometry. Temporal aspects are used to describe the change of geometry and topology over time, and the appearance and disappearance of structures and other robots. Furthermore, BeSpaceD formalizes conditions under which a change may occur.

BeSpaceD is implemented in Scala and its core functionality runs in a Java environment. It has been successfully applied in different contexts such as decision support for factory automation [4] and for verification of spatiotemporal properties of industrial robots [19]. Besides of basic logical operators (e.g., `AND`), BeSpaceD offers special constructs for space, time, and topology. For instance, `OccupyBox` refers to a rectangular two-dimensional space parameterized by its left lower and its right upper corner points while constructs like `TimeInterval` make the modeling of temporal aspects possible. For example, the following formula expresses that the rectangular space with the corner points $(1050, 2056)$ and $(1502, 2603)$ will be temporary closed between the time points 200 and 600:

```
IMPLIES(AND(TimeInterval(200,600),Owner("TemporaryClosure")),
    OccupyBox(1050,2056,1502,2603))
```

BeSpaceD formulas can be efficiently analyzed for spatiotemporal and other properties. For instance, we can specify a point of time and a predicate and derive the spatial

implications from these definitions. Likewise, logical quantifiers can be applied to specify and check the existence of a spatiotemporal condition in a specification, or to prove that a certain property holds for a distinct time and space area. In addition, algorithms and tools such as external SMT solvers (e.g., we have a connection to z3 [8]) can help to resolve geometric constraints such as the overlapping of different areas in time and space. A variety of different operators (for instance, breaking geometric constraints on areas down to geometric constraints on points) exist which facilitates the reasoning about geometric and topological constraints.

We decided to utilize BeSpaceD for reasoning about the context rules since it addresses their spatiotemporal nature well. The BeSpaceD language is used to specify the rules while the efficient automatic analysis of BeSpaceD makes it suitable to find out which rules have to be triggered in a certain context. Moreover, the expressibility of BeSpaceD allows us to define prioritization schemes in order to find out which of two or more contradicting rules to prefer.

3.2 Software Reconfiguration

The control software of an mCPS unit shall support runtime changes in the configuration parameters as well as in the software structure, e.g., through software component replacement. In addition, the mCPS software has to support *plugability of context data provider* components. Such general-purpose components are dynamically loaded and unloaded from the mCPS device, providing contextual information such as sensor data.

The software reconfiguration mechanism is built on the Java-based OSGi (formerly known as Open Service Gateway initiative) platform [28]. We chose this framework as resource-efficiency has been one of the core design goals of OSGi and therefore it does not impose high resource overhead on mCPS devices. Some existing implementations, such as Concierge [32], exhibit a reasonable memory footprint for resource-constraint devices (80 kB). Our testbed, the Raspberry Pi node used to control a DiddyBorg, can host the Eclipse OSGi framework, called Equinox [9]. OSGi offers a class-loading mechanism to dynamically load and unload Java packages, so-called *business bundles* that, in our context, may implement a service like listening to a sensor or controlling an activator. For that, OSGi offers functions to activate, deactivate, and replace a business bundle. In particular, it preserves automatically the dependencies of a system when some of its bundles are installed, uninstalled or reconfigured.

Figure 3 shows the architecture of the VASF. The core idea of the proposed architecture is to provide a virtualization layer supporting the access to the collaborating units. Thus, it allows abstract services like the *Context Monitoring* to interact with one or more actual services on selected devices in order to perform a part of an adaptation process, e.g., loading a bundle on devices.

Service *properties* are the main source of information for determining the filtering, i.e., which data to consider. They enable the association of additional metadata with services, in addition to the service name. We propose two types of properties for services in order to achieve adaptation: i) generic properties that describe the functionally of a service, e.g., `localization.ultrasound`; ii) properties describing the non-functional aspects of services, e.g., `accuracy.distance.high` or `energy.low`.

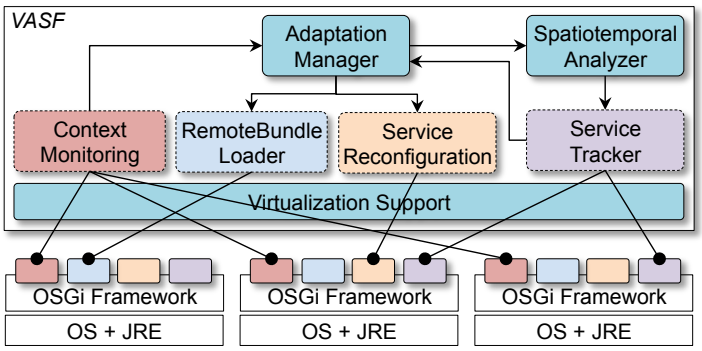


Fig. 3: Architecture of virtualization-based adaptation using the OSGi framework

For example, if the functional property for filtering is decided to be `localization.shortrange`, another level of filtering can be performed by including the non-functional property `energy.low` in order to select a short range localization service that demands less energy. If there is more than one bundle implementing a service, the `service.ranking` property, a standard OSGi property, is defined for such services in order to select a single service with the highest ranking.

The VASF can be either implemented and deployed on a network node (e.g., a gateway) or it can reside on an mCPS unit that is powerful enough to perform the context processing and to communicate with the other devices for performing the adaptation actions. In the latter case, we need a leader selection protocol (see, e.g., [29]) to compensate for the failure of the leading unit. The key component of the VASF, the *Virtualization Support*, handles the communications between the VASF and the mCPS units, in our case, the Raspberry Pi devices. The WiFi capability of devices along with the socket-based data transmission between the VASF device and robots will be used, e.g., to perform service tracking and load remote bundles.

The *Adaptation Manager* receives the current system context retrieved by the *Context Monitoring* service from the corresponding actual services on the units. The *Adaptation Manager* forwards the context space information to the *Spatiotemporal Analyzer*, running the BeSpaced-based analysis. Based on its results, the *Spatiotemporal Analyzer* selects the OSGi mechanism to execute, as well as the new property settings and notifies the *Service Tracker*. This service is used to detect services of a specific type, e.g., sending notifications when a service is started or stopped. Its instance at the VASF level is in charge of communicating with the equivalent services on selected mCPS devices to perform service tracking. Thanks to OSGi, we can build our own *Service Tracker* based on OSGi's service tracking component. Specifically, when the *Spatiotemporal Analyzer* suggests a new service for update, the *Service Tracker* will perform investigations regarding the current status of the old service and ensure that other bundles using this service can continue their execution safely during reconfiguration.

After the new service is located by the *Spatiotemporal Analyzer* and the *Service Tracker*, the *Adaptation Manager* will initiate the service reconfiguration phase using the dynamic service reconfiguration feature of OSGi. If no service with the specified

Table 1: Context change rules for the test bed

No.	*Location*	*Other context aspects*	*Reconfigurations*	*Other changes*
1	Hallways	US sensor detects no obstacle within 300 cm	Switch off IR sensor	
2	Everywhere	US sensor detects obstacle between 100 and 300 cm	Switch on IR sensor	Reduce speed to 50%
3	Everywhere	US or IR sensors detect obstacle within 100 cm		Reduce speed to 25%
4	Everywhere	US or IR sensors detect obstacle within 70 cm	Rely on IR sensor only	
5	Room B216	Magnetic sensor detects course towards the heating pipes	Switch on detector for unstable readings of the US sensor	
6	Room B216	Detector shows unstable readings of the US sensor		Reduce speed to 25%
7	Outdoors	Temperature sensor shows a temperature below 3°C		Reduce speed to 50%
8	Hallways	Time is between 00 and 15 of the hour (lecture break)	Switch on IR sensor	Reduce speed to 50%
9	Everywhere	Two or more robots within 10 meters	Switch on IR sensor at all effected robots	Reduce speed to 50% at all effected robots
10	Everywhere	Two or more robots within 5 meters		Reduce speed to 25% at all effected robots

filtering criteria is found, the *Adaptation Manager* invokes the *RemoteBundle Loader* to download the bundle that implements the requested service and load it to the OSGi runtime system. This service is also used to load *Context Bundles* (see Fig. 2) that are dynamically added to the system in order to collect new context information based on the available sensors of the robot. Like the *Service Tracker*, the *Service Reconfiguration* on the VASF communicates with the actual equivalent services on selected mCPS devices to perform the configuration process.

3.3 Combining Context Reasoning and Software Reconfiguration

The business bundles in our DiddyBorg testbed comprise functionality like path planning, the control of the six engines, reading the various sensors, and special tasks necessary in a certain environment (e.g., a detector for unstable sensor readings close to heating pipes). Depending on the reconfigurations selected in the current context, bundles may change in one or more robots. For example, in the case of an area temporarily closed for an mCPS, a new path planning algorithm may be loaded and replace the previous one. Furthermore, once BeSpaceD identifies the approach of an uncritical area, we may temporarily unload bundles that control some sensing devices if these are not needed for the uncritical area. This can result in energy savings.

To clarify our approach, we list in Table 1 a set of rules[4] guiding context reasoning and software reconfiguration for our DiddyBorg robot testbed. The rules depend on the current robot location and other contextual aspects like sensor readings, input from collaborating units, time properties, or reports about other robots being in the same room. They are described as BeSpaceD formulas and guide the context reasoning. The results of a rule may be software reconfigurations and other changes, e.g., the adjustment of control parameters. For instance, rule 5 defines that if the robot is in room B216 and the magnetic sensor detects that it heads towards the heating pipes which may effect the reading of the ultrasound sensor, an OSGi-bundle containing a detector function is added which constantly checks the readings of the ultrasound sensor. If the detector finds out that the readings get unstable, i.e., start to jump ferociously, this is an indication that the robot approaches the heating pipes. In this case, following rule 6, the control parameter determining the robot speed is changed such that the speed is reduced to 25% preventing the robot crashing against the pipes.

As discussed above, rules may be contradictory. For instance, rule 1 demands to switch off the infrared sensor saving energy when the robot is operated in one of the hallways and the ultrasound sensor does not detect any obstacles within 3 m. In contrast, according to rule 8 the infrared sensor has to be kept operating between 00 and 15 of the hour since then there is a break between lectures and many people are expected in the hallways. Thus, the likelihood of somebody encountering the robot from the side leaving a room is significantly higher and the infrared sensor may provide a better reading of the exact distance. The context reasoning shall, of course, guarantee the safest possible solution which, in our case, means prioritizing rule 8 keeping the infrared sensor on. Using its prioritization scheme which, for instance, may determine that a sensor has to be active if desired by one of the contradicting rules, BeSpaceD decides that rule 8 will be applied and not rule 1.

The Diddyborg demonstrator is currently under development such that we cannot provide reliable measures about the time needed for the various adaptations of the control systems of the robots. First tests of OSGi on a Raspberry Pi, however, give the impression that the typical OSGi reconfiguration functions like starting or changing business bundles are quite resource-efficient such that we expect no major performance penalties of our approach. Of course, that has to be analyzed more in-depth when the full demonstrator is running. Likewise, we have to determine the impact of the BeSpaceD-based analysis on the overall performance. As discussed in [21], BeSpaceD proved to run efficiently for similar approaches such that we do not expect a serious penalty here either.

4 Related Work

Self-adaptation of CPS software systems has received great attention because of their inherent unpredictability and high dynamicity. In the following, we discuss the related work with respect to context modeling techniques for mCPS and the solutions for enabling self-adaptation of mCPS software system.

[4] In Tab. 1, *US* stands for *ultrasound* and *IR* for *infrared*.

Spatiotemporal specifications and means to reason about them, are an important part of our work. A variety of different specification mechanisms such as a process algebra-like formalism and a related type system for concurrency and resource control exists [6]. In the SPEEDS project [2, 17], contracts between components are used to model behavior in the form of transition systems. Means to reason about spatial and geometric constraints are described in, e.g., [1, 20]. Additional logic approaches for hybrid systems (e.g., [12, 31]) provide comprehensive languages and tools for describing CPS including time and space. In contrast to these works, our system combines the Java basis of the OSGi context with spatiotemporal reasoning at runtime that can take place on the robot controller to support runtime adaptivity of CPS.

Other work addresses the large amount of information available for context processing [7, 23]. This also includes the diversity of application variants for software reconfiguration [5, 27]. However, the view to diversity in these types of approaches is different from what we envisage for mCPS applications, i.e., growing and diverse *spatiotemporal* contextual changes. Another aspect of context processing is to address uncertainty in adaptation. In [15, 16], meta-adaptation strategies are proposed that extend the adaptability of a system by constructing new tactics at runtime reflecting the changes in the environment.

With respect to dynamic software adaptation models for CPS, a framework for mapping the large component model Kevoree into micro controller-based architectures is discussed in [11]. The main goal of this work is to push dynamics and elasticity concerns directly into resource-constrained devices, based on the notion of models@runtime. The proposed dynamic component model is benchmarked against certain key criteria such as memory usage and reliability, on an Arduino board with an ATMEL AVR 328P microcontroller. The main focus of this work is on efficient development of dynamic components for resource-constrained CPS, which is different from the goal of this paper, i.e., modeling contextual aspects in a scalable manner and adopting a service-based view to adaptation instead of the device-level view. In [33], an approach is proposed for the development of adaptable software applications for embedded systems based on a Domain-Specific Language (DSL). The authors chose DSL-based adaptation to specify adaptation policies and strategies at a high-level, using rules that produce the necessary runtime reconfigurations independent from the application logic. In particular, they develop the adaptation framework of a Lego NXT Mindstorms robot exploring the environment. Similarly, PLASMA [35] and Sykes [34] utilize ADL and planning-as-model-checking technologies to enable dynamic replanning in the architectural domain in robots. The focus of these works is on specifying the adaptable behavior through DSL, while our work is characterized by a high-level service-oriented abstraction model, which is domain-independent and simplifies adaptation modeling and implementation in typical mCPS.

5 Conclusions and Future Work

Enabling efficient dynamic software adaptation is considered a key requirement of mCPS with respect to their high dynamicity and spatiotemporal contextual changes. In this paper, we propose a service-based software adaptation framework to address

this concern. The main goal of the presented framework is to facilitate the adaptation of mCPS software, referring to the support of complicated contextual changes in the spatiotemporal aspects of mCPS. Further, our approach supports the collaborative nature of mCPS for which we need an application- and high-level view to adapting the whole mCPS, rather than individual devices. We achieve this by introducing a new software architectural model for dynamic mCPS systems, which is based on the idea of virtualizing the adaptation services at the application level through a Virtulization Services Adaptation Framework (VASF) and then interpreting the process to actual low-level adaptation actions performed on selected devices. We use BeSpaceD for modeling and reasoning, while the framework components and the self-configuration model are demonstrated for the DiddyBorg robots with Raspberry Pi boards using OSGi. As our future plan, we will develop a generic version of the VASF and use it in a variety of mCPS environments. As an demonstrator, we will apply the adaptation framework to a bottling plant deployed in the RMIT's advanced manufacturing precinct.

References

1. B. Bennett, A. G. Cohn, F. Wolter, and M. Zakharyaschev. Multi-Dimensional Modal Logic as a Framework for Spatio-Temporal Reasoning. *Applied Intelligence*, 17(3), 2002.
2. A. Benveniste et al. Multiple Viewpoint Contract-based Specification and Design. In *Formal Methods for Components and Objects*, 2008.
3. J. O. Blech et al. BeSpaceD: Towards a Tool Framework and Methodology for the Specification and Verification of Spatial Behavior of Distributed Software Component Systems. *ArXiv e-print*, abs/1404.3537, 2014.
4. J. O. Blech et al. Efficient Incident Handling in Industrial Automation through Collaborative Engineering. In *Emerging Technologies Factory Automation (ETFA), 2015 IEEE 20th Conference on*, 2015.
5. G. Brataas, S. O. Hallsteinsen, R. Rouvoy, and F. Eliassen. Scalability of Decision Models for Dynamic Product Lines. In *SPLC (2)*. Kindai Kagaku Sha, 2007.
6. L. Caires. Spatial-behavioral Types for Concurrency and Resource Control in Distributed Systems. *Theoretical Computer Science*, 2008.
7. D. Conan et al. Scalable Processing of Context Information with COSMOS. In *Distributed Applications and Interoperable Systems*, LNCS 4531. Springer-Verlag, 2007.
8. L. De Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008.
9. Eclipse. Eclipse Equinox Framework. `http://www.eclipse.org/equinox/`, 2016.
10. J. Eidson, E. Lee, S. Matic, S. Seshia, and J. Zou. Distributed Real-Time Software for Cyber-Physical Systems. *Proceedings of the IEEE*, 100(1), 2012.
11. F. Fouquet et al. A Dynamic Component Model for Cyber Physical Systems. In *Proc. of 15th ACM Symposium on Component Based Software Eng. (CBSE '12)*. ACM, 2012.
12. G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable Verification of Hybrid Systems. In *Computer Aided Verification (CAV)*, 2011.
13. S. Fritsch et al. Time-bounded Adaptation for Automotive System Software. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th Conf. on*, 2008.
14. S. Fritsch et al. Time-bounded adaptation for automotive system software. In *Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on*, 2008.

15. I. Gerostathopoulos et al. Meta-adaptation strategies for adaptation in cyber-physical systems. In *Software Architecture: 9th European Conference, ECSA 2015*, 2015.
16. I. Gerostathopoulos et al. Self-adaptation in software-intensive cyberphysical systems: From system goals to architecture configurations. *Journal of Systems and Software*, 2016.
17. S. Graf et al. Contract-based Reasoning for Component Systems with Rich Interactions. In *Embedded Systems Development*, volume 20 of *Embedded Systems*. Springer, 2014.
18. F. Han et al. Model-Based Engineering and Analysis of Space-Aware Systems Communicating via IEEE 802.11. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 2, 2015.
19. P. Herrmann et al. A Model-based Toolchain to Verify Spatial Behavior of Cyber-Physical Systems. *International Journal of Web Services Research (IJWSR)*, 13(1):40–52, 2016.
20. D. Hirschkoff et al. Minimality Results for the Spatial Logics. In *Foundations of Software Technology and Theoretical Computer Science*, LNCS 2914. Springer, 2003.
21. S. Hordvik et al. A Methodology for Model-based Development and Safety Analysis of Transport Systems. In *11th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2016.
22. N. Huber et al. Model-based Self-adaptive Resource Allocation in Virtualized Environments. In *SEAMS '11*, 2011.
23. S. Kang, J. Lee, H. Jang, H. Lee, Y. Lee, S. Park, T. Park, and J. Song. SeeMon: Scalable and Energy-efficient Context Monitoring Framework for Sensor-rich Mobile Environments. In *MobiSys '08*. ACM, 2008.
24. M. Lauer, S. Lange, and M. Riedmiller. Calculating the Perfect Match: An Efficient and Accurate Approach for Robot Self-localization. In *RoboCup 2005: Robot Soccer World Cup IX*, LNCS 4020. Springer, 2006.
25. E. Lee. Cyber Physical Systems: Design Challenges. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, 2008.
26. W. Y. Lum and F. C. M. Lau. A Context-Aware Decision Engine for Content Adaptation. *IEEE Pervasive Computing*, 1(3), 2002.
27. V. Nallur and R. Bahsoon. A Decentralized Self-Adaptation Mechanism for Service-based Applications in the cloud. *Software Engineering, IEEE Transactions on*, 39(5), 2013.
28. OSGi Alliance. OSGi Service Platform. `http://www.osgi.org/`, 2016. accessed: 2016-01-22.
29. S. Patterson and B. Bamieh. Leader Selection for Optimal Network Coherence. In *49th IEEE Conference on Decision and Control*, pages 2692–2697, 2010.
30. PiBorg. DiddyBorg Raspberry Pi Robot. `http://www.piborg.org/diddyborg`, 2016.
31. A. Platzer. Differential Dynamic Logic for Hybrid Systems. *Journal of Automated Reasoning*, 41(2), 2008.
32. J. S. Rellermeyer and G. Alonso. Concierge: A Service Platform for Resource-constrained Devices. *SIGOPS Oper. Syst. Rev.*, 41(3), 2007.
33. A. C. Santos et al. Specifying Adaptations through a DSL with an Application to Mobile Robot Navigation. In *SLATE'13*, 2013.
34. D. Sykes et al. From goals to components: A combined approach to self-management. In *SEAMS '08*, 2008.
35. H. Tajalli et al. Plasma: A plan-based layered architecture for software model-driven adaptation. In *Proc. the IEEE/ACM Conf. on Automated Software Engineering*, ASE '10, 2010.
36. D. Weyns et al. On decentralized self-adaptation: Lessons from the trenches and challenges for the future. In *SEAMS'10*, 2010.
37. J. White et al. R&D Challenges and Solutions for Mobile Cyber-Physical Applications and Supporting Internet Services. *Journal of Internet Services and Applications*, 1(1), 2010.