A Role-Based Architecture for Self-Adaptive Cyber-Physical Systems

Tim Kluge
Tim.Kluge1@tu-dresden.de
Technische Universität Dresden, Software Technology Group
Dresden, Germany

ABSTRACT

Today's computing world features a growing number of cyber-physical systems that require the cooperation of many physical devices. Examples include autonomous cars and co-working robots, which are expected to appropriately adapt to any possible context they find themselves in (e.g. the presence of a nearby human).

However, the controlling software continues to be developed using established object-oriented modelling techniques like UML, which do not natively possess a notion of context and thus may introduce accidental complexity. With increasing complexity, the probability of the introduction of software errors rises, which can have fatal consequences in cyber-physical systems. To address this, we envision a model-driven architecture for self-adaptive cyber-physical systems that explicitly models structured context. Entities are modelled as message-passing parallel processes and can play roles in specific contexts, which dynamically alter their behaviour and relationships with other parts of the system. Since the planning of complex adaptations can be cumbersome in real-world scenarios, we envision an intuitive formulation of adaptations as graph rewriting rules on the context model.

This paper discusses the current state of research and identifies open research challenges. Based on this, the envisioned architecture as well as an evaluation strategy are presented.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; *Distributed architectures*; • **Software and its engineering** → *System modeling languages.*

KEYWORDS

self-adaptive systems, context-sensitivity, roles, distributed systems, decentralized systems, graph rewriting

ACM Reference Format:
Tim Kluge. 2020. A Role-Based Architecture for Self-Adaptive Cyber-Physical Systems. In *IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3387939.3391601

SEAMS '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7962-5/20/05…$15.00
https://doi.org/10.1145/3387939.3391601

1 INTRODUCTION

In recent years, pervasive computing has continued to emerge in all parts of human society. While in 2018 more than 8 billion cellular connections were made by smart IoT devices, this amount is expected to reach 22 billion by 2024 [5]. They form a growing number of systems that are expected to work autonomously without requiring human interaction. As self-adaptive systems, they dynamically change their behaviour to adapt to a changing environment and meet functional or non-functional requirements.

Emergent types of connected devices include cyber-physical systems (CPS) like collaborative robots and autonomous cars. Such systems are subject to a variety of contexts, e. g., the weather conditions that the autonomous car drives in. As humans rely on these systems for their safety, failures to adapt can be fatal. Yet, system complexity and the number of considered contexts continue to rise. It has been shown that the number of introduced bugs scales linearly with the number of code lines (LOC) [7]. The probability of a fatal error in the CPS therefore increases with its complexity. As such, the reduction of system complexity, as well as the need for more configurability and context-sensitivity, have been identified as research challenges in the domain of self adaptive systems [22].

The complexity of a software system can be regarded as twofold. On the one hand, the essential complexity of the system can be measured by the amount of concepts, entities and relations that are captured by the system's models. On the other hand, *accidental complexity* is introduced by the failure of models to capture aspects of the concepts that they describe [6].

Obviously, any form of accidental complexity is undesired. Nonetheless, it can be introduced by accident when working with commonly used object-oriented modelling languages. For example, the structural part of the industry-standard object-oriented modelling language UML only models objects and the relationships they have with each other. It has been argued by Steimann and others that in addition to these two concepts, the notion of a *role* is a fundamental concept which object-oriented modelling languages should be able to capture [19]. As the behaviour of an object depends on its context, the behaviour of an object in a specific context is considered its role in that context. Steimann differentiates between *natural types* that do not depend on context and *roles*, which do.

To further explain the feasibility of the usage of roles, we introduce a motivating example in the following section.

1.1 Introductory Example

Consider the following scenario for a self-adaptive cyber-physical system. A number of autonomous cars drive on a road. Cars that are next to each other communicate and dynamically form a car convoy to utilize the reduced air drag and to save on fuel. A car convoy is

led by a single car. If it starts to rain, all cars adapt by slowing down. While this scenario can be modelled without roles, they allow for a natural depiction of the context-dependant behaviour of the cars:

- The car is modelled as a natural type, which defines behaviour and properties that are not context-dependant. This includes the physical car's position and speed, whose representation in the instance model is continuously updated by a monitoring component.
- When regarding a convoy as a context, the cars that join the convoy play the role of a participant in that convoy. One car plays the role of the convoy leader.
- The rain can be modelled as an additional context, in which Cars play the role of a "slow driver".

As shown, the role notion allows for an intuitive modelling of the scenario. Modelling the scenario in a modelling language that does not feature roles would require modelling the roles with other concepts, which introduce additional accidental complexity.

1.2 Summary

We therefore claim that the introduction of roles to the modelling of cyber-physical systems can decrease their accidental complexity while increasing the expressiveness of the models. Based on the notion of roles, we envision a new way to build self-adaptive cyber-physical systems that supports the declarative definition of decentral architectural adaptations of the role model, which in combination with the context-sensitivity of roles allow for a high configurability and context-sensitivity of distributed self-adaptive systems.

The remainder of the paper is structured as follows. In section 2, relevant concepts of roles and self-adaptive software are introduced. The section includes a short overview on related work. In section 3, a research vision and accompanying questions are deduced. The final section 4 gives an overview over the planned PhD process schedule and strategies to evaluate the proposed solutions.

2 BACKGROUND

In this section, concepts that are related to our work are introduced.

2.1 Contexts and Roles

Context-Oriented programming (COP) without roles has been proposed as a simple approach to the development of context-sensitive software [10]. Here, the notion of a layer is the central concept. A layer consists of method definitions, which can be enabled and disabled at runtime. Methods can be executed after, before, around or instead of the base method. In that regard, COP shares concepts with aspect-oriented programming and allows the dynamic toggling of aspects at runtime. COP has been integrated with Java and other object-oriented languages [11]. While COP allows for context-sensitive programming, it does not support the software modelling process as roles do.

Role-based programming in Java has been enabled by Object Teams for some time [9], also known as OT/J. Another library for role-based programming on the JVM is SCROLL [15], which is implemented as an embedded domain-specific language for Scala[1].

[1] https://scala-lang.org

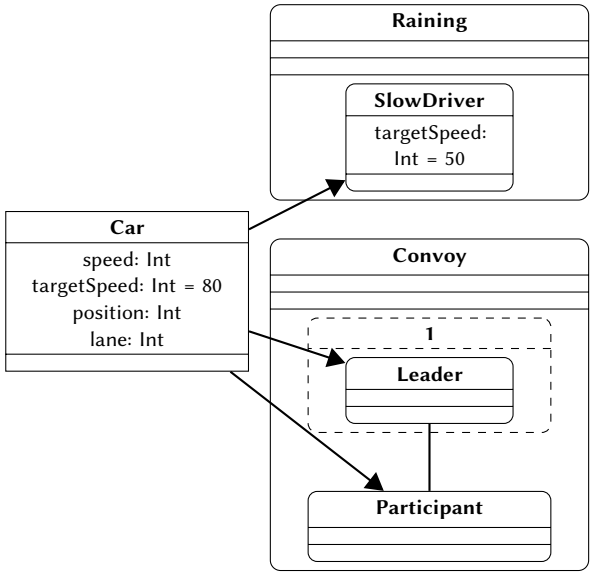

Figure 1: CROM model of the introductory car convoy example

Utilizing Scala language features like implicit functions and compiler rewritings, SCROLL enables a pure embedding of roles with syntactical niceties.

Schütze et al. recently provided an overview of available role-based programming languages, including a quantitative comparison of their performance [18].

While only a brief introduction to roles was given in section 1, Kühn et al. recently conducted a full literature review to build a common understanding of the role concept [14]. The authors introduce the notion of a structured context, named *compartment*. Based on this concept, a role-based modelling language named CROM is designed. In CROM, compartments contain roles, while naturals exist outside any compartment. Relationships only occur between roles. The authors highlight the behavioural, relational and context-dependent aspects of roles and introduce relationship constraints as modelling concepts. Roles can implicate and prohibit other roles. Relationships can be annotated with a cardinality. Kühn et al. also provide a full list of features as well as a graphical notion for the CROM meta model [13].

In fig. 1, we use this notion to depict a visual representation of a CROM model of our introductory car convoy example presented in section 1.1. Here, role plays are visualized as arrows, while the general relationship between the leader and participants of a car convoy is indicated with a simple line. As an example of supported constraints, the leader role is part of a role-group which defines that there needs to be exactly one leader at any time.

It has been shown by Taing et al. that compared to COP, roles support the unanticipated adaptation of the software at runtime [21].

2.2 Self-Adaptivity

Self-Adaptivity is commonly realized using feedback loops. A widespread pattern is a MAPE-K loop, which divides the adaptation

process into the phases monitoring, analysis, planning and execution [12]. All phases share a common knowledge base. In the monitoring phase, data from environmental sensors is collected, which is subsequently analysed for patterns. Based on the findings, the system plans if and how to adapt itself.

As the analysis and planning of adaptations can become complex for real-world systems, different approaches to realize these phases have been proposed. Promising approaches include the adaptation of the system's architecture (i. e. its models) with graph rewriting. In [1], the authors propose graph rewriting rules that consist of a *story*, which defines a graph pattern to be matched and additional conditions. The matched pattern is mapped to a graph pattern that replaces the matched pattern in the architecture. Using the defined rules, possible system states can be formally reasoned about using model checking. Similar approaches that also feature graph-rewriting have been proposed by [23] and by [20] for distributed systems, among others.

Another approach to the planning problem is the definition of system goals, where the system automatically tries to find adaptations that bring it closer to the defined goals [4]. In [2], the authors envision a cyber-physical system for robots, where a team of robots works to minimize a defined cost function, e. g. the distance to a nearby human.

A solution for the execution of planned adaptations for role-based systems has been proposed by Weißbach et al. [25]. Here, adaptations are grouped to transactions, which are decentrally executed on multiple nodes running a role-based system. A single adaptation is a transformation of the role instance model, e. g. the addition of a role-play.

As parts of the system can act independently of each other, a concurrency model is needed to describe possible interactions.

2.3 Concurrency Models

Efficient utilization of concurrency is an emergent problem in software development. In distributed systems, concurrent processes have to synchronize by communication, as no memory is shared. Prominent examples of the modelling of concurrency include process calculi like communicating sequential processes (CSP). CSP defines a formal language for synchronous message exchange between processes via channels. As such, CSP can be reasoned about to formally verify properties, e. g. to identify deadlocks [17]. A prominent language to feature a concurrency model based on CSP is Go[2].

Another approach to model concurrency is the actor model. An actor is a named entity that has a single unbounded message queue, called its inbox. It can asynchronously send messages to other actors, identified by their name. The actor only reacts to incoming messages and does not become active on its own. Over the years, various extensions to the actor concept have been proposed [3]. Actors are the preferred concurrency mechanism in the Scala programming language, among others.

3 RESEARCH VISION AND CHALLENGES

Having discussed the current state-of-the-art in section 2, we line out open research challenges that will be addressed by our work in this section. To illustrate the identified gaps, we will use the introductory car convoy example in section 1.1. The realization of this example serves as a first goal.

As depicted, distributed programming with roles is an open issue. While existing languages and frameworks for role-based programming like Object Teams allow for shared-memory parallelism, a message-passing based solution that maps roles to actors or processes has not been considered yet. As such, there is currently no middleware that allows describing a distributed role-based system with a single CROM model. In our car convoy example, one CROM model has been used to describe the complete structure which spans multiple nodes.

To implement the car convoy example, the actual adaptations (e. g. the formation of the car convoy) have to be implemented as well. The execution of transactional adaptations of a decentral role-based system has already been proposed by Weißbach. However, a concept on how to plan these adaptations has not been considered and the proposed solution only adapts each local role runtime independently.

This leads to the main two research questions:

- **RQ1** How can a role-based context-model be used to build a distributed cyber-physical system?
- **RQ2** How can decentral adaptations of such a system be planned?

Having identified the main research questions, we will line out our strategy to handle them.

4 RESEARCH VISION

In the following section, an overview of conducted preliminary work and the planned timeline for the PhD process and handling of the research questions is provided. The author takes part in the research training group "Role-based Software Infrastructures for continuous-context-sensitive Systems" (RoSI), whose current phase runs from October 2019 to September 2022. Thus, of the available time frame, two full years and approximately nine months remain.

As a first step towards the goal to develop a distributed role-based system, we built an initial prototype together with other members of the research training group. The prototype consisted of a role-based stream-processing application that processed a GPS track across four separated machines. One node read the GPS points stored in a Kafka message queue and two nodes subsequently colourized the points with red and blue respectively. The colourizers were implemented using roles. The final node stored the stream in another Kafka instance. Each node was modelled separately with CROM and implemented using SCROLL. We reimplemented the Weißbach transaction approach with the SCROLL runtime so that the colourizer roles could be swapped out with a transaction. The communication between the nodes was implemented by hand using the Akka[3] framework. The transactions were composed manually of hand-written adaptations of the role model, e. g. "Remove blue colourizer role from operator object A".

As such, the development of this prototype resulted in the identification of the research questions mentioned in section 3.

[2] https://golang.org

[3] https://akka.io

4.1 Research Plan

To gain a more sophisticated understanding of the current state of the art in the research area of self-adaptive systems, it is planned to conduct a systematic literature review and validate the feasibility of the proposed architecture as a first step. Consequently, a survey paper on the reviewed literature will be written by the author in cooperation with other members of the research initiative. In the following sections, the scientific *we* is used to describe the intentions of the author only.

After completing the literature review, we want to advance towards RQ1 by evaluating concurrency models and providing a proof-of-concept implementation of a distributed role runtime where contexts and role-plays are still manually described in transactions as proposed by Weißbach. To achieve this, we consider modelling roles and naturals as actors or CSP processes, where communication channels between processes can be deduced from the relationships in the CROM model. While SCROLL is available as a feature-complete local role runtime which could be extended to a distributed version, a reimplementation that natively supports asynchronous programming and the generation of proxy classes for distributed programming could be feasible. As such, Kotlin[4] might be a suitable target language. We expect to have a prototypical distributed role runtime available by the end of 2020, which could be released to the research community as an artefact in early 2021.

After handling RQ1 by providing a distributed role runtime, we will advance towards RQ2. The goal would be to naturally plan adaptations like the formation of a car convoy in the introductory example given in section 1.1. We expect graph rewriting on the CROM instance model to be a promising option, as it would allow to directly use context-dependent knowledge. Graph rewriting on the CROM model itself might be feasible as well to adapt the role architecture. While researchers have used graph-rewriting with other graph models as depicted in section 2, the literature review should identify more experiences by field researchers and identify alternative solutions. The handling of RQ2 is expected to be completed by the end of 2021. The remaining available time in 2022 is expected to be spent on the thesis writing process and dealing with unexpected research challenges.

Summarizing the schedule, of the remaining two and a half years, one year is planned to handle each research question. The remaining time is reserved for thesis writing and leftover research issues. As part of the handling of RQ2, a qualitative and quantitative evaluation of the architecture is planned, which is described in the following section.

4.2 Evaluation

To show that the proposed architecture supports the development of self-adaptive cyber-physical systems, it is planned to qualitatively evaluate the system by conducting multiple case studies. The following studies are planned:

(1) A virtual simulation of the introductory example as depicted in section 1.1. Several separated virtual machines run the software, with each machine continuously updating one simulated car instance. The convoy formation is then tested. As part of this case study, an introspection tool will be developed, which visualizes the current model for debugging purposes. The case study is intended to function as a minimal example and proof-of-concept study.
(2) An actual cyber-physical system where real industry-grade robots and a human collaborate. Two Franka Emika robotic arms[5] are available for testing at the author's department. A possible demonstrative example is the joint placement of domino stones. The robots enter a safe mode context on a nearby human encounter.
(3) A virtual simulation of a more complex system. This should utilize a scenario found in the evaluation of other self-adaptive systems. Candidates include the DartSIM example, which features the control of a fleet of unmanned aerial vehicles. DartSIM was published as an artefact at SEAMS 2019 [16]. Although DartSIM emphasizes uncertainty and delayed execution in self-adaptive systems, we claim the scenario can be implemented using our approach.

Additionally, a quantitative evaluation will be provided. We plan to use the code complexity as the main measure. As we claimed in section 1, our approach is expected to reduce the accidental complexity and thus reduce the probability of software errors. Possible metrics for code complexity include the number of written statements and the number of independent linear control flow paths through a sequential program (Cyclomatic complexity). For object-oriented programs, the complexity of the underlying object class model is crucial, which amounts to the number of classes and their relationships [24]. This metric could provide a data-set for the quantitative comparison of our DartSIM implementation with other approaches, including a version of the discussed other case studies without roles. Additionally, the resource requirements in terms of computation time and memory usage of our approach will be compared to alternative approaches.

Put together, we will provide qualitative and quantitative measures to support our claim that the proposed architecture supports the development of distributed self-adaptive applications.

4.3 Contributions

The expected contributions of our work are twofold. As a basis, we plan to contribute a distributed role runtime (RQ1) that maps roles to actors or sequential processes as its underlying concurrency model, and a mechanism to adapt the executed role model (RQ2). To plan the adaptations, we consider using graph rewritings rules on the role instance model. Combined, the two contributions are expected to form a new framework for the development of self-adaptive cyber-physical systems that reduces the introduced accidental complexity by modelling the individual components with roles and structured context.

ACKNOWLEDGMENTS

This work is funded by the German Research Foundation (DFG) within the Research Training Group Role-based Software Infrastructures for continuous-context-sensitive Systems (GRK 1907).

[4] https://kotlin-lang

[5] https://www.franka.de/

REFERENCES

[1] Basil Becker and Holger Giese. 2008. Modeling of correct self-adaptive systems: A graph transformation system based approach. *5th International Conference on Soft Computing as Transdisciplinary Science and Technology, CSTST '08 - Proceedings* January 2008 (2008), 508–516. https://doi.org/10.1145/1456223.1456326

[2] Tomas Bures, Filip Krijt, Frantisek Plasil, Petr Hnetynka, and Zbynek Jiracek. 2015. Towards intelligent ensembles. *ACM International Conference Proceeding Series* 07-11-Sept (2015). https://doi.org/10.1145/2797433.2797450

[3] Joeri De Koster, Tom Van Cutsem, and Wolfgang De Meuter. 2016. 43 years of actors: A taxonomy of actor models and their key properties. *AGERE 2016 - Proceedings of the 6th International Workshop on Programming Based on Actors, Agents, and Decentralized Control, co-located with SPLASH 2016* 31 (2016), 31–40. https://doi.org/10.1145/3001886.3001890

[4] Giovanna Di Marzo Serugendo, Noria Foukia, Salima Hassas, Anthony Karageorgos, Soraya Kouadri Mostéfaoui, Orner F. Rana, Mihaela Ulieru, Paul Valckenaers, and Chris Van Aart. 2004. Self-organisation: Paradigms and applications. *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 2977 (2004), 1–19. https://doi.org/10.1007/978-3-540-24701-2_1

[5] Ericsson. 2019. Ericsson Mobility Report (June 2019). *Ericsson White Paper* June (2019), 36. www.ericsson.com/mobility-report

[6] Frank J Furrer. 2019. Future-Proof Software-Systems. In *Future-Proof Software-Systems*. Springer, 45–55.

[7] J. E. Gaffney. 1984. Estimating the Number of Faults in Code. *IEEE Transactions on Software Engineering* SE-10, 4 (July 1984), 459–464. https://doi.org/10.1109/TSE.1984.5010260

[8] Kyle Headley. 2018. A DSL embedded in rust. *ACM International Conference Proceeding Series* (2018), 119–126. https://doi.org/10.1145/3310232.3310241

[9] Stephan Herrmann. 2005. Programming with roles in objectteams/java. *AAAI Fall Symposium - Technical Report* FS-05-08 (2005), 73–80.

[10] Robert Hirschfeld and Pascal Costanza. 2008. Context-oriented Programming. *Journal of Object Technology* 7, 3 (2008), 125–151. https://doi.org/10.7892/boris.37163

[11] Robert Hirschfeld and Michael Haupt. 2009. ContextJ : Context-oriented Programming with Java. 26 (2009).

[12] IBM. 2006. An architectural blueprint for autonomic computing. *IBM White Paper* 36, June (2006), 34. https://doi.org/10.1021/am900608j

[13] Thomas Kühn, Stephan Böhme, Sebastian Götz, and Uwe Abmann. 2015. A combined formal model for relational context-dependent roles. *SLE 2015 - Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (2015), 113–124. https://doi.org/10.1145/2814251.2814255

[14] Thomas Kühn, Max Leuthäuser, Sebastian Götz, Christoph Seidl, and Uwe Aßmann. 2014. A metamodel family for role-based modeling and programming languages. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8706 (2014), 141–160. https://doi.org/10.1007/978-3-319-11245-9_8

[15] Max Leuthäuser. 2015. SCROLL - A Scala-based library for Roles at Runtime. abs/1508.03536 (09 2015), 7.

[16] Gabriel Moreno, Cody Kinneer, Ashutosh Pandey, and David Garlan. 2019. DARTSim: An exemplar for evaluation and comparison of self-adaptation approaches for smart cyber-physical systems. *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* 2019-May (2019), 181–187. https://doi.org/10.1109/SEAMS.2019.00031

[17] A. W. Roscoe. 2005. *Communicating Sequential Processes. The First 25 Years*. Vol. 3525. 15–35 pages. https://doi.org/10.1007/b136154

[18] Lars Schütze and Jeronimo Castrillon. 2017. Analyzing state-of-the-art role-based programming languages. In *ACM International Conference Proceeding Series*, Vol. Part F1296. Association for Computing Machinery. https://doi.org/10.1145/3079368.3079386

[19] Friedrich Steimann. 2000. On the representation of roles in object-oriented and conceptual modelling. *Data and Knowledge Engineering* 35, 1 (2000), 83–106. https://doi.org/10.1016/S0169-023X(00)00023-9

[20] Gabriele Taentzer, Michael Goedicke, and Torsten Meyer. 2000. Dynamic change management by distributed graph transformation: Towards configurable distributed systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 1764 (2000), 179–193. https://doi.org/10.1007/978-3-540-46464-8_13

[21] Nguonly Taing, Markus Wutzler, Thomas Springer, Nicolás Cardozo, and Alexander Schill. 2016. Consistent unanticipated adaptation for context-dependent applications. *Proceedings of the 8th International Workshop on Context-Oriented Programming, COP 2016* (2016), 33–38. https://doi.org/10.1145/2951965.2951966

[22] Gabriel Tamura Morimitsu and Norha Milena Villegas Machado. 2013. *Software Engineering for Self-Adaptive Systems II*. Vol. 7475. https://doi.org/10.1007/978-3-642-35813-5

[23] Matthias Tichy and Benjamin Klöpper. 2012. Planning Self-adaption with Graph Transformations. In *Applications of Graph Transformations with Industrial Relevance*, Andy Schürr, Dániel Varró, and Gergely Varró (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 137–152.

[24] Honglei Tu, Wei Sun, and Yanan Zhang. 2009. The research on software metrics and software complexity metrics. *IFCSTA 2009 Proceedings - 2009 International Forum on Computer Science-Technology and Applications* 1 (2009), 131–136. https://doi.org/10.1109/IFCSTA.2009.39

[25] Martin Weißbach and Thomas Springer. 2017. Coordinated execution of adaptation operations in distributed role-based software systems. *Proceedings of the ACM Symposium on Applied Computing* Part F128005 (2017), 45–50. https://doi.org/10.1145/3019612.3019624