International Journal on Software Tools for Technology Transfer (2020) 22:389–397
https://doi.org/10.1007/s10009-020-00565-0

FOUNDATION FOR MASTERING CHANGE

Special Section REoCAS

Rigorous engineering of collective adaptive systems: special section

Rocco De Nicola[1] · Stefan Jähnichen[2] · Martin Wirsing[3]

Published online: 13 May 2020

Abstract
An adaptive system is able to adapt at runtime to dynamically changing environments and to new requirements. Adaptive systems can be single adaptive entities or collective ones that consist of several collaborating entities. Rigorous engineering requires appropriate methods and tools that help guaranteeing that an adaptive system lives up to its intended purpose. This paper introduces the special section on "Rigorous Engineering of Collective Adaptive Systems." It presents the seven contributions of the section and gives a short overview of the field of rigorously engineering collective adaptive systems by structuring it according to three topics: systematic development, methods and theories for modelling and analysis, and techniques for programming and operating collective adaptive systems.

Keywords Adaptive systems · Collective adaptive systems · Software engineering · Formal methods · Rigorous methods

1 Introduction

A collective adaptive system (CAS), often also called ensemble, consists of collaborating entities that are able to adapt at runtime to dynamically changing open-ended environments and to new requirements [40,45]. Often the entities of such a system have their own individual properties and objectives; interactions with other entities or with humans may lead to the emergence of unexpected phenomena. Many modern systems are collective and adaptive: smart systems such as smart cities and smart traffic, collective cyber-physical systems such as robot swarms and sensor networks, as well as socio-technical systems such as distributed energy systems.

Rigorous engineering of collective adaptive systems requires appropriate methods and tools that help guaranteeing that—in spite of dynamically changing environments—they live up to their intended purpose. This includes theories for designing and analysing collective adaptive systems, techniques for programming and operating such systems, rigorous methods for devising adaptation mechanisms, validation and verification techniques as well as approaches for ensuring security and trust and optimising performance.

Research projects In the last 10 years research on collective adaptive systems has been fostered by European and national research programs and projects. The notion of collective adaptive system was elaborated in a workshop on "Fundamentals of Collective Adaptive Systems", organised by the European Commission in 2009 [45]; earlier in 2006–2008, the notion of ensemble was developed in a series of workshops of the Interlink project [71], funded by the Future and Emerging Technologies (FET) Unit of the European Commission. As a sequel, the FET unit launched two proactive initiatives. The action ICT-2009.8.5 "Self-Awareness in Autonomic Systems"[1] called for "new concepts, architectures, foundations and technologies" for computing and communication systems "that are able to optimise overall performance and resource usage in response to changing conditions, adapting to both context (such as user behaviour) and internal changes (such as topology)."

✉ Martin Wirsing
wirsing@lmu.de

Rocco De Nicola
rocco.denicola@imtlucca.it

Stefan Jähnichen
stefan.jaehnichen@tu-berlin.de

[1] IMT School for Advanced Studies Lucca, Lucca, Italy

[2] TU Berlin and FZI Forschungszentrum Informatik Berlin, Berlin, Germany

[3] Ludwig-Maximilians-Universität München, Munich, Germany

[1] https://cordis.europa.eu/programme/id/FP7_ICT-2009.8.5, read on 2020/02/03.

The follow-up action ICT-2011.9.10 on "Fundamentals of Collective Adaptive Systems"[2] focused on socio-technical systems "that are constructed as a collective of heterogeneous components and that are tightly entangled with humans and social structures". Altogether 12 projects were funded including ASCENS [73,74] on "Autonomic Service- Component ENSembles", SAPERE [75] on "Self-Aware PERvasive service Ecosystems", ALLOW Ensembles [17] on large scale collective adaptive ensembles, QUANTICOL [14] on a quantitative approach to the design and management of collective and adaptive behaviours, and the coordination action FoCAS.[3]

More recently, a number of National projects or initiatives have been funded in Europe, here we would like to mention just two that are very close to us.

The German Transregional Collaborative Research Centre 248 "Foundations of Perspicuous Software Systems"[4] aims at enabling comprehension in a cyber-physical world with the human in the loop. The starting consideration is that computer programs increasingly participate in actions and decisions that affect humans but that the understanding of how these applications interact and what is the cause of a specific automated decision is lagging far behind. These "smart systems" calculate and propagate outcomes of computations, but unfortunately do not provide explanations.

The Italian collaborative project IT-Matters[5] has the goal of developing and the experimenting with a novel methodology for the specification, implementation and validation of trustworthy smart systems based on formal methods. Three basic steps are envisaged: (i) providing and analysing system models to find design errors, (ii) moving from models to executable code by translation into domain-specific programming languages and, finally, (iii) monitoring run-time execution to detect anomalous behaviours and to support systems in taking context-dependent decisions autonomously.

ISoLA This special section on ''Rigorous Engineering of Collective Adaptive Systems" was inspired by three successful tracks [30,44,72] on rigorous engineering at the ISoLA conferences in 2014 [51], 2016 [52], and 2018 [53]. In 2014, the topic of the track was "Rigorous Engineering of Autonomic Ensembles" and the six papers of the track presented results of the ASCENS project [73]. The tracks of 2016 and 2018 were not dedicated to the outcome of a single project, but addressed more widely the theme of rigorous engineering collective adaptive systems. The 12 scientific contributions of 2016 and the 18 contributions of 2018 covered a broad range of topics such as formally modelling, analysing and programming collective adaptive systems as well as security, machine learning, and software support for collective adaptive systems.

Contents The seven papers of this special section are revised and extended versions of papers presented at the ISoLA 2018 track [30]. In this paper we give a very short account of the field of rigorously engineering collective adaptive systems by structuring the research about collective adaptive systems along three different lines:

- systematic development of CAS,
- methods and theories for modelling and analysing CAS,
- techniques for programming and operating CAS.

The seven contributions of the special section are presented as part of this general overview.

2 Systematic development of CAS

Key features of collective adaptive systems are the so-called self*-properties, such as self-awareness and self-adaptation. Systematic development approaches address these features and support (some or most of) the classical phases of the software development life cycle such as requirements elicitation and specification, design, implementation, validation and verification, and deployment. In the following we distinguish goal-oriented methods with focus on the early development phases, methods based on feedback control loops, and automated synthesis.

Goal-oriented methods Well-known early goal-oriented engineering approaches are KAOS [28] and Tropos [15]. KAOS [28] distinguishes hard and soft goals, is formally based on (linear temporal) logic, and proposes activities for refining the goals and deriving operational requirements which serve as the basis for system design. Tropos [15] is a method for developing multi-agent systems. Its notion of agent is founded on the BDI (*Belief*, *Desire*, and *Intention*) architecture [62]. Key modelling concepts are actors, their dependencies as well as the goals, plans and capabilities. For KAOS and Tropos, the development process covers all phases from requirements to implementation; special focus is put on the identification of stakeholders and the specification of the environment in the early requirements phases. Neither KAOS nor Tropos offer direct support for adaptation, but in [24,58] it is shown how to use KAOS for developing dynamic self-adaptive systems, while a Tropos extension, called Tropos4AS, is presented in [57] as a tool for engineering self-adaptive systems.

[2] https://cordis.europa.eu/programme/id/FP7_ICT-2011.9.10, last accessed 2020-04-16.

[3] http://www.focas.eu, last accessed 2020-04-16.

[4] https://www.perspicuous-computing.science/, last accessed 2020-04-16.

[5] http://itmatters.imtlucca.it/, last accessed 2020-04-16.

ADELFE [11] is a methodology for developing self-organising cooperative and thus collective multi-agent systems. Its main ingredients are the characterisation of the system environment and the identification and treatment of so-called non-cooperative situations. ADELFE 3.0 [55] proposes an iterative process that integrates modelling, programming and simulation techniques. The method supports the UML Unified Process and tailors it to the specificity of self-organising systems. In [56] ADELFE is combined with Tropos4AS to obtain a goal-oriented development method for self-organising multi-agent systems.

Autonomous Requirements Engineering [67] (ARE) is a goal-oriented method for systematically eliciting so-called autonomy requirements. These are the self*-objectives of the system which are derived by applying a model for generic autonomy requirements to any system goal and its environmental constraints. The ARE method was applied to unmanned space missions [68] as well as to ensembles such as a peer-to-peer cloud [67].

The paper in this special section "The SOTA approach to engineering collective adaptive systems" [4] by Dhaminda Abeywickrama, Nicola Bicocchi, Marco Mamei, and Franco Zambonelli presents SOTA, a goal-oriented requirements engineering and modelling method for describing the overall domain and the requirements of a collective adaptive system. SOTA focuses on modelling and analysing functional and non-functional requirements of self-adaptation, and enables the early verification of requirements, the identification of knowledge requirements for self-adaptation, and the identification of the most suitable architectural patterns for self-adaptation. A more detailed and formal counterpart of SOTA specifications is the general ensemble model GEM [41].

The invariant refinement method IRM [19] supports the transition from early high-level requirements—specified, e.g. in SOTA and ARE, see [4,67]—to software architecture. IRM captures goals and requirements as invariants that describe desired system states over time. High-level invariants are iteratively decomposed into more specific sub-invariants up to the level that they can be operationalised by autonomous components and component ensembles.

Methods based on feedback control loops An early reference model for adaptive systems is the MAPE-K architecture introduced by IBM [42]. It comprises a control loop with the four phases *Monitor*, *Analyse*, *Plan*, *Execute*, all of which exploit the *Knowledge* of the system itself and its environment. The research roadmap for self-adaptive systems [29] suggests a life cycle based on MAPE-K and proposes the use of a process modelling language to describe the self-adaptation workflows and feedback control loops. The "generic life cycle for context-aware adaptive systems" [43] is also based on MAPE-K and addresses foreseen and unforeseen evolution of the environment. In [16] feedback loops are considered as first-class entities and are explored from a control engineering perspective. Similarly, in [64] a bio-cybernetic loop serves as basis for a three-layered software architecture for adaptive systems.

The ensemble development life cycle EDLC [39] deals with awareness and adaptation in a "runtime feedback control loop." It is an agile process whose development phases are arranged in two "wheels": at "design" time, the classical development phases—requirements engineering, modelling and programming, verification and validation—are iterated; at "runtime", the entities of the ensemble iterate the "runtime feedback control loop" consisting of monitoring, awareness, and self-adaptation. The connection between design time and runtime is established by the deployment or hot update of the system and in the other direction by the runtime system providing feedback to the design cycle. The construction of collective adaptive systems using EDLC is supported by eight engineering principles [9]. EDLC has been used in the development of several ensemble systems covering swarm robots [74], peer-to-peer cloud [54], and e-mobility applications [18].

Automated synthesis Automated synthesis of programs is a lifely and important research area (for an overview see [36]). Collective adaptive systems are typically too complex to be automatically generated but some crucial parts can be synthesised or generated.

Autonomous systems are subject to faults caused by the interaction with the environment. The paper [33] proposes an algorithm for automatically synthesising runtime monitors for fault detection and recovery strategies for controller synthesis. The approach is currently under implementation in the BIP [7] framework.

The gap between simulated hardware and real hardware implementations is addressed in [70]. It proposes a novel architecture for realising collective adaptive systems on hardware devices for real-world scenarios. Hardware devices are equipped with self-descriptions and these self-descriptions are used to automatically find appropriate devices for the tasks required by the system or the users.

Modern energy systems comprise several local community energy systems [23] which produce, consume, and sell energy. In [23] community energy systems are modelled as so-called electronic institutions [60,63] and two optimisation methods are proposed to automatically find appropriate policies for selling and consuming energy. The optimisation methods are based on genetic programming and reinforcement learning. The results show that the evolved policies clearly outperform the initially human-designed policy and enable the energy system to remain sustainable over time.

The paper in this special section "Learning-based coordination model for spontaneous self-composition of reliable

services in a distributed system" [50] by Houssem Ben Mahfoudh, Giovanna Di Marzo Serugendo, Nabil Naja, and Nabil Abdennadher studies decentralised services which are built and composed on-demand and arise from the interaction of multiple sensors and devices. The bio-inspired SAPERE coordination model [75] is extended to a learning-based coordination model which accommodates spontaneous self-composition of services in fully distributed scenarios using multiple nodes. Through learning, agents progressively update their behaviours and initial results show that the collective adaptive system are able to converge towards reliable composition of services, in terms of both functionality and expected quality of services.

3 Methods and theories for modelling and analysing CAS

Process algebraic, logical, and model-based methods are used for modelling dynamically reconfigurable architectures and ensemble requirements.

As of now there exist a large number of instances of Process Description Languages (PDLs), but overall they share a common ground: they can be generally described as *action-based formalisms* relying on behavioural operators that support *compositionality* and *abstraction*, where the meaning of process terms is formally defined via a set of structural operational semantic rules that can be used to associate a state transition graph to each process term. Such state transition graphs are then used as the basis for model checking properties expressed with one of the many temporal logics that have been proposed. After the success of such logics in specifying and verifying qualitative properties of system, other richer logics and tools for considering also quantitative properties have been proposed, to perform also probabilistic, stochastic or statistical model checking: see, e.g. [46] and [47]. For an extensive overview of such logics and associated tools, we refer the reader to [6] and references therein.

In parallel with the above approach, *Architecture description languages* (ADLs) have been developed and used to model the architecture of software-intensive systems. With the term *architecture* we refer to the description of the components that comprise a system, the behavioural specifications of those components, and the patterns and mechanisms for their interactions. However, as pointed out in surveys on the main ADLs [27,34,59] their semantics is not always fully formal and early analysis of system architectures is often difficult if not impossible.

Below we describe some of the logic-based approaches that have been more concerned with the specification and verification of collective adaptive systems. Then we will consider (architectural) languages and their models.

Logic-based methods and analysis techniques Complex requirements of collective adaptive systems are specified concisely by modal logics that exploit dynamic, spatial and temporal concepts.

Spatial Temporal Logics have been introduced in order to specify spatial aspects of computation that are very important in systems distributed in physical space and thus in many collective adaptive systems.

These logics offer a topology-based approach to formal verification of spatial properties depending upon physical space based on a spatial logic, We extend the framework with a spatial surrounded operator, a propagation operator and with some collective operators.

A spatial logic (SLCS), stemming from the tradition of topological interpretations of modal logics, dating back to earlier logicians such as Tarski, where modalities describe neighbourhood is introduced in [25]. SLCS extends the classical framework with a spatial surrounded operator, a propagation operator and with some collective operators which are interpreted over arbitrary sets of points instead of individual points in space. A variant of SLCS, named SSTL, with two new spatial modalities: the somewhere operator and a novel bounded version of the topological surrounded operator is introduced in [26] and is used in [25] to detect problems in vehicle location data for city buses and, e.g. spot the undesirable phenomenon of "clumping" which occurs when there is not enough separation between subsequent buses serving the same route.

Properties of dynamically changing ensembles can be described in dynamic logics. In [61] a differential dynamic logic (dL) is proposed to lay the foundations for developing adaptive cyber-physical system models by rigorously specifying their behaviour: the logic is then accompanied by a continuous monitoring to safeguard the decisions of learning agents and guide run-time decisions. In [38] a dynamic logic is presented for specifying the global behaviour of systems by desired and forbidden interaction scenarios.

Languages and models Several formalisms for specialised component models and languages for collective adaptive systems have been proposed, in particular in the context of the ASCENS project [73]. One of the main objectives of these formalisms is the actual formation of ensembles of components forming a collective system with the idea that ensembles are formed according to the overall goals of the system and possibly to the environmental conditions.

The Service Component Ensemble Language (SCEL) [32] has been designed to deal with collective adaptation. It brings together programming abstractions to directly address aggregations (how different components interact to form ensembles and systems), behaviours (how components progress) and knowledge manipulation according to specific policies. These ingredients constitute the basis of so-called attribute-

based communication, a novel paradigm that enables groups of partners to interact by considering the predicates over the (dynamic) values of the attributes they expose. SCEL is equipped with an operational semantics that permits verification of properties of systems. SCEL has also been the main source of inspiration for CARMA, a language recently defined to support specification and analysis of collective adaptive systems. CARMA [13,48] combines the lessons learned from SCEL and the long tradition of stochastic process algebras [31].

The Helena approach [37] proposes a role-based method for modelling collaborations using a UML-like notation and is founded on a rigorous formal semantics. Helena focuses on the description of the behaviour of each role as well as on the behaviour on the ensemble level. In Helena, a component instance explicitly indicates which of the ensembles it belongs to.

The DEECo (Dependable Emergent Ensembles of Components) component model [21] aims at providing the relevant software engineering abstractions that ease the programmers' tasks. A component in DEECo features an execution model based on the MAPE-K autonomic loop. Like SCEL, it consists of well-defined knowledge and processes that are executed periodically in a soft real-time manner. The component concept is complemented by the first-class ensemble concept. An ensemble stands as the only communication mechanism between DEECo components. In DEECo ensembles are specified explicitly and ensemble formation employs a "greedy" strategy, in that a component is a member of each of the ensemble instances, the membership condition of which it satisfies, and that at the same time, this condition is solely based on the component's current knowledge.

Another formalism developed outside ASCENS is BIP [7]. This formalism aims at combining the approach based on PDLs and ADLs by relying on a logic language for modelling the interaction of components. Within the framework, components are obtained as the superposition of three layers that give name to the framework itself: *Behaviour*, *Interaction* between transitions of the behaviour in the form of a set of *connectors*, and *Priority* to describe the scheduling policies of interactions. The framework is designed to support *incremental* construction of systems starting from atomic components combined through a parameterised binary *composition* operator.

The two papers in this special section address reconfiguration modelling and formal modelling of scenario evolution.

The paper "The DReAM framework for dynamic reconfigurable architecture modelling: theory and applications" [49], Alessandro Maggi, Joseph Sifakis, and Rocco De Nicola introduces a new framework for modelling dynamic reconfigurable architectures. A system is understood as a dynamically changing set of typed components and a system architecture is characterised by a set of coordination constraints. The DReAM language is based on the Propositional Interaction Logic [12] of the BIP framework and extends it with operations for data exchange between components and by coordination terms for regulating interactions and reconfigurations among a set of components. Static architectures are modelled by Interaction Logic formulas. The main ingredient for modelling dynamic architectures is the new notion of 'motif' which is defined by a DReAM coordination term. The DReAM framework is implemented as a Java API together with an execution engine.

In "Adapting quality assurance to adaptive systems: the scenario coevolution paradigm" [35] by Thomas Gabor, Andreas Sedlmeier, Thomy Phan, Fabian Ritz, Marie Kiermeier, Lenz Belzner, Bernhard Kempter, Cornel Klein, Horst Sauer, Reiner Schmid, Jan Wieghardt, Marc Zeller, and Claudia Linnhoff-Popien take a more abstract view and present a formal framework for adaptation and testing of adaptive systems using scenarios; it also discusses how such a framework can be used for increasing the trustworthiness of complex adaptive systems. In particular, the work extends the system model and the notion of adaptation space of [41] of the ASCENS project [74] by abstract definitions of self-adaptation and scenarios. A main contribution is also the concept of scenario coevolution for making quality assurance self-adaptive in order to match the capabilities of the self-adaptive system-under-test.

4 Techniques for programming and operating CAS

In collective adaptive systems interactions are typically performed following an opportunistic pattern and might involve several selected partners among a huge number of alternatives. It is thus very costly or infeasible to rely on channels, addresses or component identities for selecting such partners. A number of alternative programming frameworks have been proposed with different intercommunication strategies and abstractions.

ECA programming Unlike traditional programming languages, Event-Condition-Action (ECA) programs consist of a small set of rules, each of which specifies an action that is performed when an event occurs and a specific condition holds [10]. This event-driven model allows for the specification of robust programs in which the static data model is separated from the dynamic part. Tools have been developed to support runtime execution and static/dynamic analysis of ECA programs to guarantee satisfaction of invariants [65,66].

Attribute-based programming The "attribute-based" paradigm allows for specifying communicating systems by abstracting from channels and identities. Systems are

described as sets of parallel components equipped with attributes encoding relevant features such as role, status, and position that can be modified at runtime. Communication links among components are dynamically established through predicates over their attributes. Attribute-based programming was first used in SCEL [32] and has been then refined in AbC [1,2]. This more abstract calculus concentrates on primitives and interaction mechanisms that are crucial for dealing with CAS and relies on a notion of implicit, anonymous, and non-blocking multicasting. An AbC system is rendered as a set of parallel components, each equipped with a set of attributes, termed as attribute environment, and with a behaviour, termed as a process. The attribute values can be modified by internal actions and the behaviour of a component is parameterised to these values.

Aggregate programming A similar approach is adopted in the "aggregate programming" paradigm [8]: systems are coherent collections of interacting devices, whose coordination and composition are hidden to programmers. In aggregate programming, a program is specified over a region of the computational environment expressing e.g. network structure, continuous space and time. Programs manipulate data computed by the individual devices in that region, via local sensing and interactions with (spatial or network) neighbours. Several models and programming languages supporting encapsulation, modulation, and composition of services have been already proposed, e.g. the Field Calculus and the language Protelis [69].

The last part of this special session is exactly dedicated to setting up appropriate programming frameworks to support a programming style that would ease programming, evolution, management, and security of collective adaptive systems. It also addresses testing methods and fault handling techniques for self-organising adaptive and autonomic systems.

The paper "A distributed API for coordinating AbC programs" [3] by Yehia Abd Alrahman and Giulio Garbi concentrates on the interaction models for CAS where environmental conditions largely influence interactions. In particular, it considers a possible implementation of message exchange based on anonymous multicast communication as advocated in AbC. The paper describes an efficient and distributed coordination infrastructure for AbC and proves its correctness with respect to the formal operational semantics of ABC. The proposed approach guarantees that the individual components are infrastructure agnostic. Thus the code of a component does not specify how messages are routed in the infrastructure but rather what properties a target component must satisfy. The actual communication infrastructure is implemented through a Go API, named GoAt, and an Eclipse plugin that enables one to program in a high-level syntax which can be automatically used to generate matching Go code. The approach is illustrated through a non trivial case study implementing a distributed graph colouring algorithm. The performance of the tree-based coordination infrastructure is also evaluated against two alternative ones based on clusters and rings.

The paper "A language and framework for dynamic component ensembles in smart systems" [20] by Tomáš Bureš, Ilias Gerostathopoulos, Petr Hnetynka, František Plášil, Filip Krijt, Jiří Vinárek, and Jan Kofron presents the "Trait-based COmponent Ensemble Language" TCOEL. This specification and architecture description language builds on the experience on using DEECo [21] within the same group and is based on the consideration that in complex real-life systems ensembles may overlap, be nested, and dynamically formed and dismantled in a distributed environment. TCOEL is built on top of Scala and supports the combination the ensemble specifications with existing libraries in Java and other JVM-based languages available on multiple platforms. Ensembles are constructed by integrating domain-independent concepts (such a number and type of the members of an ensemble) with domain-dependent features, so-called traits, similar to mixins in object-oriented languages. Examples for traits are spatial constraints, predictions of certain values, or statistical tests. For evaluating the approach it is shown how a RoboCup Rescue application can be specified and simulated in TCOEL.

The last paper of this section, ''Toward autonomically composable and context-dependent access control specification through ensembles" [5] by Rima Al Ali, Tomáš Bureš, Petr Hnetynka, Jan Matejek, František Plášil, and Jiří Vinárek is concerned with the possibility of extending the concept of ensemble and exploiting it to define dynamic access control rules to govern interactions in a system of evolving autonomic components. Autonomic component ensembles [22] are enhanced with autonomously composable and context-dependent access control rules that follow the dynamicity and context-dependence of the components of a system. These are the so-called (dynamic) security ensemble rules that describe the allowed interactions in the system and follow the system during its evolution. The rules are complemented with an approach to match ensembles to the current state of the system so as to resolve the binding of dynamic access control rules to current situations and to generate the set of actions permitted at a given moment given the components in the system and their context. Care is taken for deriving the state of components that are not controlled by the system (e.g. humans and 3rd party components) and thus are not directly observable. For the specification of the rules, an internal Scala-based DSL has been proposed, but for end-user usage, an external a user friendly DSL is also offered.

5 Concluding remarks

Collective adaptive systems pose many research challenges which range from engineering adaptivity to novel requirements for the classical topics of modelling, analysing, testing, safety, security, and performance. Adaptivity is probably the most challenging topic for further evolution of our current technologies. But adaptivity is not only concerned with technical challenges and solutions. Even more important are the questions of acceptance and trust as well as those concerned with liability and, in general, with law. More research will be needed to master the rigorous engineering of such collective adaptive systems, to make them reliable and trustworthy. The story continues …., in the mean time, enjoy this special section of STTT.

Acknowledgements Open Access funding provided by Projekt DEAL. As editors of this special section we would like to thank all authors for their valuable contributions and all reviewers for their careful evaluations and constructive comments. We are also grateful to the ISoLA chairs, Tiziana Margaria and Bernhard Steffen, for giving us the opportunity to organise this special section and we thank them and STTT for providing us with the very helpful Online Journal Service (OJS) system.

References

1. Abd Alrahman, Y., De Nicola, R., Loreti, M.: A calculus for collective-adaptive systems and its behavioural theory. Inf. Comput. **268**, 104457 (2019)
2. Abd Alrahman, Y., De Nicola, R., Loreti, M.: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. **192**, 102428 (2020)
3. Abd Alrahman, Y., Garbi, G.: A distributed API for coordinating AbC programs. In this issue
4. Abeywickrama, D., Bicocchi, N., Mamei, M., Zambonelli, F.: The SOTA approach to engineering collective adaptive systems. In this issue
5. Al Ali, R., Bureš, T., Hnetynka, P., Matejek, J., Plášil, F., Vinárek, J.: Towards autonomically composable and context-dependent access control specification through ensembles. In this issue. (2018)
6. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)
7. Basu, A., Bensalem, S., Bozga, M., Bourgos, P., Sifakis, J.: Rigorous system design: the BIP approach. In: MEMICS 2011, Volume 7119 of Lecture Notes in Computer Science 7119, pp. 1–19. Springer, Berlin (2012)
8. Beal, J., Viroli, M.: Aggregate programming: from foundations to applications. In: Bernardo, M., De Nicola, R., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems—SFM 2016, Volume 9700 of Lecture Notes in Computer Science, pp. 233–260. Springer, Berlin (2016)
9. Belzner, L., Hölzl, M.M., Koch, N., Wirsing, M.: Collective autonomic systems: towards engineering principles and their foundations. Trans. Found. Mastering Chang. **1**, 180–200 (2016)
10. Berndtsson, M., Mellin, J.: ECA rules. In: Liu, L., Özsu, T. (eds.) Encyclopedia of Database Systems, 2nd edn, pp. 959–960. Springer, Berlin (2018)
11. Bernon, C., Gleizes, M.-P., Migeon, F., Di Marzo Serugendo, G.: Engineering self-organising systems. Self-organising Softw, **2011**, 283–312 (2011)
12. Bliudze, S., Sifakis, J.: The algebra of connectors–structuring interaction in BIP. IEEE Trans. Comput. **57**(10), 1315–1330 (2008)
13. Bortolussi, L., De Nicola, R., Galpin, V., Gilmore, S., Hillston, J., Latella, D., Loreti, M., Massink, M.: Collective adaptive resource-sharing Markovian agents. In: Quantitative Analysis of Programming Languages, Volume 194 of EPTCS, pp. 16–31 (2015)
14. Bortolussi, L., De Nicola, R., Gast, N., Gilmore, S., Hillston, J., Massink, M., Tribastone, M.: A quantitative approach to the design and analysis of collective adaptive systems. In: 1st FoCAS Workshop on Fundamentals of Collective Adaptive Systems (2013)
15. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: an agent-oriented software development methodology. JAAMAS **8**(3), 203–236 (2004)
16. Brun, Y., Di Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H.M., Litoiu, M., Müller, H. A., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. In: Software Engineering for Self-Adaptive Systems, pp. 48–70 (2009)
17. Bucchiarone, A., Dulay, N., Lavygina, A., Marconi, A., Raik, H., Russo, A.: An approach for collective adaptation in socio-technical systems. In: SASO Workshops, pp. 43–48 (2015)
18. Bureš, T., De Nicola, R., Gerostathopoulos, I., Hoch, N., Kit, M., Koch, N., Monreale, G.V., Montanari, U., Pugliese, R., Šerbedžija, N., Wirsing, M., Zambonelli, F.: A life cycle for the development of autonomic systems: the e-mobility showcase. In: SASO Workshops, pp. 71–76 (2013)
19. Bureš, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M., Plášil, F.: The invariant refinement method. In: [73], pp. 405–428 (2015)
20. Bureš, T., Gerostathopoulos, I., Hnetynka, P., Plášil, F., Krijt, F., Vinárek, J., Kofron, J.: A language and framework for dynamic component ensembles in smart systems. In this issue
21. Bureš, T., Plášil, F., Kit, M., Tuma, P., Hoch, N.: Software abstractions for component interaction in the internet of things. Computer **49**(12), 50–59 (2016)
22. Bureš, T., Plášil, F., Kit, M., Tuma, P., Hoch, N.: Software abstractions for component interaction in the internet of things. IEEE Comput. **49**(12), 50–59 (2016)
23. Cardoso, R.P., Rossetti, R.J.F. , Hart, E., Kurka, D.B., Pitt, J.: Engineering sustainable and adaptive systems in dynamic and unpredictable environments. In: [53], pp. 221–240 (2018)
24. Cheng, B., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modelling approach to develop requirements of an adaptive system with environmental uncertainty. In: MODELS '09, Volume 5795 of Lecture Notes in Computer Science, pp. 468–483. Springer, Berlin (2009)
25. Ciancia, V., Gilmore, S., Grilletti, G., Latella, D., Loreti, M., Massink, M.: Spatio-temporal model checking of vehicular movement in public transport systems. STTT **20**(3), 289–311 (2018)
26. Ciancia, V., Latella, D., Loreti, M., Massink, M.: Model checking spatial logics for closure spaces. Log. Methods Comput. Sci. **12**, 4 (2016)

27. Clements, P.C.: A survey of architecture description languages. In: IWSSD '96, vol. 8, p. 16. IEEE Computer Society Press (1996)
28. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci Comput Program **20**(1–2), 3–50 (1993)
29. de Lemos, R., Giese, H., Müller, H.A., Shaw, M., Andersson, J., Litoiu, M., Schmerl, B.R., Tamura, G., Villegas, N.M., Vogel, T., Weyns, D., Baresi, L., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Desmarais, R., Dustdar, S., Engels, G., Geihs, K., Göschka, K.M., Gorla, A., Grassi, V., Inverardi, P., Karsai, G., Kramer, J., Lopes, A., Magee, J., Malek, S., Mankovski, S., Mirandola, R., Mylopoulos, J., Nierstrasz, O., PezzèC, M., Prehofer, W., Schäfer, R., Schlichting, D., Smith, J.P., Sousa, L., Tahvildari, K.Wong, Wuttke, J.: Software engineering for self-adaptive systems: a second research roadmap. In: Software engineering for self-adaptive systems, pp. 1–32 (2010)
30. De Nicola, R., Jähnichen, S., Wirsing, M.: Rigorous engineering of collective adaptive ensembles—track introduction. In: [53], pp. 3–12 (2018)
31. De Nicola, R., Latella, D., Loreti, M., Massink, M.: A uniform definition of stochastic process calculi. ACM Comput. Surv. **46**(1), 5:1–5:35 (2013)
32. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. ACM Trans. Auton. Adapt **9**(2), 7:1–7:29 (2014)
33. Dragomir, I., Iosti, S., Bozga, M., Bensalem, S.: Designing systems with detection and reconfiguration capabilities: a formal approach. In: [53], pp. 155–171 (2018)
34. Fuxman, A.D.: A survey of architecture description languages. In: Reports from CSC 2018 Automatic Verification (2000)
35. Gabor, T., Sedlmeier, A., Phan, T., Ritz, F., Kiermeier, M., Belzner, L., Kempter, B., Klein, C., Sauer, H., Schmid, R., Wieghardt, J., Zeller, M., Linnhoff-Popien, C.: The scenario coevolution paradigm: adaptive quality assurance for adaptive systems. In this issue
36. Gulwani, S., Polozov, O., Singh, R.: Program synthesis. Found. Trends Program. Lang. **4**(1–2), 1–119 (2017)
37. Hennicker, R., Klarl, A.: Foundations for ensemble modeling—the Helena approach—handling distributed systems with elaborate ensemble architectures. In: Specification, Algebra, and Software, pp. 359–381 (2014)
38. Hennicker, R., Wirsing, M.: Dynamic logic for ensembles. In: [53], pp. 32–47 (2018)
39. Hölzl, M.M., Koch, N., Puviani, M., Wirsing, M., Zambonelli, F.: The ensemble development life cycle and best practices for collective autonomic systems. In: [73], pp. 325–354 (2015)
40. Hölzl, M.M., Rauschmayer, A., Wirsing, M.: Engineering of software-intensive systems: state of the art and research challenges. In: [71], pp. 1–44 (2008)
41. Hölzl, M.M., Wirsing, M.: Towards a system model for ensembles. In: Formal Modeling: Actors, Open Systems, Biological Systems, Number 7000 in Lecture Notes in Computer Science, pp. 241–261. Springer, Berlin (2011)
42. IBM. An architectural blueprint for autonomic computing. Technical report, IBM Corporation (2005)
43. Inverardi, P., Mori, M.: Software lifecycle process to support consistent evolutions. Softw. Eng. Self-Adaptive Syst. **2010**, 239–264 (2010)
44. Jähnichen, S., Wirsing, M.: Rigorous engineering of collective adaptive systems—-track introduction. In: [52], pp. 535–538 (2016)
45. Kernbach, S., Schmickl, T., Timmis, J.: Collective adaptive systems: challenges beyond evolvability. CoRR arXiv:1108.5643 (2011)
46. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) Formal Methods for Performance Evaluation, SFM 2007, Advanced Lectures, Volume 4486 of Lecture Notes in Computer Science, pp. 220–270. Springer, Berlin (2007)
47. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical Model Checking, pp. 478–504. Springer, Berlin (2019)
48. Loreti, M., Hillston, J.: Modelling and analysis of collective adaptive systems with CARMA and its tools. In: Bernardo, M., De Nicola, R., Hillston, J. (eds.) Formal Methods for the Quantitative Evaluation of Collective Adaptive Systems—SFM 2016 Lecture Notes in Computer Science 9700, pp. 83–119. Springer, Berlin (2016)
49. Maggi, A., Sifakis, J., De Nicola, R.: The DReAM framework for dynamic reconfigurable architecture modelling: theory and applications. In this issue
50. Mahfoudh, H.B., Di Marzo Serugendo, G., Naja, N., Abdennadher, N.: Learning-based coordination model for spontaneous self-composition of reliable services in a distributed system. In this issue
51. Margaria, T., Steffen, B. (eds.): Leveraging applications of formal methods, verification and validation. In: Technologies for Mastering Change—6th International Symposium, ISoLA 2014, Part I, Volume 8802 of Lecture Notes in Computer Science. Springer, Berlin (2014)
52. Margaria, T., Steffen, B. (eds.): Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques—7th International Symposium, ISoLA 2016, Part I, Volume 9952 of Lecture Notes in Computer Science (2016)
53. Margaria, T., Steffen, B. (eds.): Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques—8th International Symposium, ISoLA 2018, Part III. Lecture Notes in Computer Science, vol. 11246. Springer, Berlin (2018)
54. Mayer, P., Velasco, J., Klarl, A., Hennicker, R., Puviani, M., Tiezzi, F., Pugliese, R., Keznikl, J., Bures, T.: The autonomic cloud. In: [73], pp. 495–512 (2015)
55. Mefteh, W., Migeon, F., Gleizes, M.-P., Gargouri, F.: Adelfe 3.0 design: building adaptive multi agent systems based on simulation a case study. In: ICCCI, vol. 1, pp. 19–28 (2015)
56. Morandini, M., Migeon, F., Gleizes, M.-P., Maurel, C., Penserini, L., Perini, A.: A goal-oriented approach for modelling self-organising mas. ESAW **2009**, 33–48 (2009)
57. Morandini, M., Penserini, L., Perini, A.: Modelling self-adaptivity: a goal-oriented approach. In: Second IEEE International Conference on Self-adaptive and Self-organizing Systems, pp. 469–470 (2008)
58. Nakagawa, H., Ohsuga, A., Honiden, S.: Constructing self-adaptive systems using a KAOS model. In: SASO Workshops, pp. 132–137 (2008)
59. Ozkaya, M., Kloukinas, C.: Are we there yet? Analyzing architecture description languages for formal analysis, usability, and realizability. In: Demirörs, O., Türetken, O. (eds.) 39th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2013, Santander, Spain, September 4–6, 2013, pp. 177–184. IEEE Computer Society, Washington (2013)
60. Pitt, J., Schaumeier, J., Artikis, A.: Axiomatization of socio-economic principles for self-organizing institutions. ACM Trans. Auton. Adapt. Syst. **7**(4), 1–39 (2012)
61. Platzer, A.: The logical path to autonomous cyber-physical systems. In: Quantitative Evaluation of Systems, 16th International Conference, QEST 2019, Volume 11785 of Lecture Notes in Computer Science, pp. 25–33. Springer, Berlin (2019)
62. Rao, A.S., Georgeff, M.P.: Modeling rational agents within a bdi-architecture. In: Proceedings of Knowledge Representation and Reasoning, pp. 473–484 (1991)
63. Rodríguez-Aguilar, J.A., Martín, F.J., Garcia, P., Noriega, P., Sierra, C.: Towards a formal specification of complex social structures in

multi-agent systems. In: Padget, J. (ed.) Collaboration Between Human and Artificial Societies 1999. Lecture Notes in Computer Science, vol. 1624, pp. 284–300. Springer, Berlin (1999)
64. Šerbedžija, N., Fairclough, S.: Biocybernetic loop: from awareness to evolution. In IEEE Evolutionary Computation 2009, pp. 2063–2069. IEEE (2009)
65. Vannucchi, C., Cacciagrano, D.R., Corradini, F., Culmone, R., Mostarda, L., Raimondi, F., Tesei, L.: A formal model for ECA rules in intelligent environments. In: Intelligent Environment (Workshops), pp. 56–65. IEEE, Washington (2016)
66. Vannucchi, C., Diamanti, M., Mazzante, G., Cacciagrano, D. R., Corradini, F., Culmone, R., Gorogiannis, N., Mostarda, L., Raimondi, F.: vIRONy:, a tool for analysis and verification of ECA rules in intelligent environments. In: Intelligent Environment, pp. 92–99. IEEE (2017)
67. Vassev, E., Hinchey, M.: Engineering requirements for autonomy features. In: [73], pp. 379–403 (2015)
68. Vassev, E., Hinchey, M.: Capturing autonomy features for unmanned spacecraft with ARE, the autonomy requirements engineering approach. Innov. Syst. Softw. Eng. **12**(2), 95–107 (2016)
69. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From distributed coordination to field calculus and aggregate computing. J. Log. Algebraic Methods Program. **109**, 100486 (2019)
70. Wanninger, C., Eymüller, C., Hoffmann, A., Kosak, O., Reif, W.: Synthesizing capabilities for collective adaptive systems from self-descriptive hardware devices—bridging the reality gap. In: [53], pp. 94–108 (2018)
71. Wirsing, M., Banâtre, J.-P., Hölzl, M.M., Rauschmayer, A. (eds.): Software-Intensive Systems and New Computing Paradigms - Challenges and Visions. Lecture Notes in Computer Science, vol. 5380. Springer, Berlin (2008)
72. Wirsing, M., De Nicola, R., Hölzl, M.M.: Rigorous engineering of autonomic ensembles—track introduction. In: [51], pp. 96–98 (2014)
73. Wirsing, M., Hölzl, M.M., Koch, N., Mayer, P. (eds.): Software Engineering for Collective Autonomic Systems—The ASCENS Approach. Lecture Notes in Computer Science, vol. 8998. Springer, Berlin (2015)
74. Wirsing, M., Hölzl, M.M., Tribastone, M., Zambonelli, F.: ASCENS: Engineering autonomic service-component ensembles. In: FMCO 2011, Volume 7542 of Lecture Notes in Computer Science, pp. 1–24 (2013)
75. Zambonelli, F., Castelli, G., Ferrari, L., Mamei, M., Rosi, A., Di Marzo Serugendo, G., Risoldi, M., Tchao, A.-E., Dobson, S., Stevenson, G., Ye, J. Nardini, E., Omicini, A., Montagna, S., Viroli, M.: Self-aware pervasive service ecosystems. In: The European Future Technologies Conference and Exhibition, Procedia Computer Science, vol. 7, pp. 197–199 (2011)