A Formal Approach to Autonomic Systems Programming: The SCEL Language
(Long Abstract)

Rocco De Nicola(✉)

IMT – Institute for Advanced Studies, Lucca, Italy
rocco.denicola@imtlucca.it

Abstract. Software-intensive cyber-physical systems have to deal with massive numbers of components, featuring complex interactions among components and with humans and other systems. Often, they are designed to operate in open and non-deterministic environments, and to dynamically adapt to new requirements, technologies and external conditions. This class of systems has been named ensembles and new engineering techniques are needed to address the challenges of developing, integrating, and deploying them. In the paper, we briefly introduce SCEL (Software Component Ensemble Language), a kernel language that takes a holistic approach to programming autonomic computing systems and aims at providing programmers with a complete set of linguistic abstractions for programming the behavior of autonomic components and the formation of autonomic components ensembles, and for controlling the interaction among different components.

Software-intensive cyber-physical systems have to deal with massive numbers of components, featuring complex interactions among components and with humans and other systems. Often, they are designed to operate in open and non-deterministic environments, and to dynamically adapt to new requirements, technologies and external conditions. This class of systems has been named *ensembles*. Sometimes, ensembles are assembled from systems that are independently controlled and managed, while their interaction "mood" might be cooperative or competitive; then one has to deal with systems coalitions or so-called *systems of systems*. Due to their inherent complexity, today's engineering methods and tools do not scale well with such systems. Therefore, new engineering techniques are needed to address the challenges of developing, integrating, and deploying them.

A possible answer to the problems posed by such complex systems is to make them able to self-manage by continuously monitoring their behavior and their working environment and by selecting the actions to perform to best deal with the current status of affairs. Self-management could be exploited also to face situations in which humans intervention is limited or even absent and components have to collaborate to achieve specific goals. This requires increasing systems' self-management capabilities and guaranteeing what now are known as

I. Lanese and E. Madelaine (Eds.): FACS 2014, LNCS 8997, pp. 24–28, 2015.
DOI: 10.1007/978-3-319-15317-9_2

*self-** properties (self-configuration, self-healing, self-optimization, self-protection) of *autonomic computing*.

The main challenges posed to language designers by these classes of systems are:

- to devise appropriate abstractions and linguistic primitives to deal with the large dimension of systems,
- to guarantee systems adaptation to (possibly unpredicted) changes of the working environment,
- to take into account evolving requirements,
- to control the emergent behaviors resulting from complex interactions.

During the invited talk, we proposed facing these challenges by taking as starting point the notions of *autonomic components* and *autonomic components ensembles* and defining programming abstractions to model their evolutions and their interactions. These notions are the means we propose to use to structure systems into well-understood, independent and distributed building blocks that interact and adapt in different ways.

Autonomic components are entities with dedicated knowledge units and resources; awareness is guaranteed by providing them with information about their state and behavior via their knowledge repositories. These repositories can be also used to store and retrieve information about the working environment of components, and can thus be used to adapt components' behavior to the perceived changes. Each component is equipped with an *interface*, consisting of a collection of *attributes*, describing different component's features such as its identity, functionalities, spatial coordinates, group memberships, trust level, response time.

Attributes play a crucial rôle, they are used by components to dynamically organize themselves into ensembles. Indeed, one of the main novelties of our approach is the way sets of partners are selected for interaction and thus how ensembles are formed. Communication partners of a specific component can be not only selected by using their identities, but also by exploiting the attributes in their interfaces. Predicates over such attributes are used to specify the targets of communication actions, to guarantee a sort of *attribute-based* communication. In this way, the formation rule of ensembles is endogenous to components: members of an ensemble are connected by the interdependency relations defined through predicates. An *autonomic-component ensembles* is therefore not a rigid fixed network but rather a highly flexible structure where components' linkages are dynamically established.

In the talk, we presented SCEL (Software Component Ensemble Language), a kernel language that takes a holistic approach to programming autonomic computing systems and aims at providing programmers with a complete set of linguistic abstractions for programming the behavior of autonomic components and the formation of autonomic components ensembles, and for controlling the interaction among different components. These abstractions permit describing autonomic systems in terms of *Behaviors*, *Knowledge* and *Aggregations*, according to specific *Policies* depicted in Fig. 1 and described below.

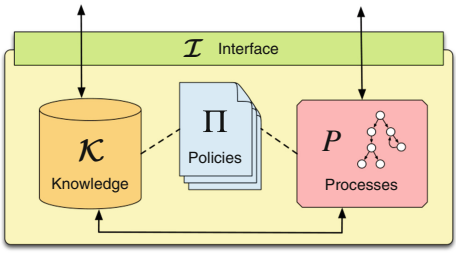

Fig. 1. SCEL component

- *Behaviors* describe how computations progress; they are modeled as processes executing actions, in the style of process calculi.
- *Knowledge* repositories provide the high-level primitives to manage pieces of information coming from different sources. Each knowledge repository is equipped with operations for *adding*, *retrieving*, and *withdrawing* knowledge items.
- *Aggregations* describe how different elements are brought together to form components and to construct the software architecture of components ensembles. Composition and interaction are implemented by exploiting the attributes exposed in components interfaces.
- *Policies* control and adapt the actions of the different components for guaranteeing accomplishment of specific tasks or satisfaction of specific properties.

Components, by accessing and manipulating their own knowledge repository or the repositories of other components, acquire information about their status (*self-awareness*) and their environment (*context-awareness*) and can perform *self-adaptation*, initiate *self-healing* actions to deal with system malfunctions, or install *self-optimizing* behaviors. All these *self-** properties, as well as *self-configuration*, can be naturally expressed by exploiting SCEL's higher-order features, namely the capability to store/retrieve (the code of) processes in/from the knowledge repositories and to dynamically trigger execution of new processes. Moreover, by implementing appropriate security policies, e.g. limiting information flow or external actions, components can set up *self-protection* mechanisms.

To show expressiveness and effectiveness of SCEL's design, we briefly introduced a Java implementation of the proposed abstractions and showed how it had been exploited for programming the robotics scenario that was used as a running example for describing achievements and potentials of the proposed approach.

The results presented in the talk have been developed within the EU-FET project Ascens [2] and most of them are presented in [9]. Other important features have been described in various other papers to which the reader is referred to for details about specific results and for references to related work. In particular:

- jRESP, the Java Run-time Environment for SCEL Programs that provides an API for using SCEL's linguistic constructs in JAVA programs is described in [11]. There, it is also discussed how jRESP can be exploited to perform statistical model checking of SCEL programs.
- Policies and their integration with the run time environment are studied in [12,13]. A full instantiation of the SCEL language, called PSCEL (Policed SCEL) that relies on modeling knowledge by means of distributed tuple spaces (à la Klaim [7]) and on FACPL for specifying policies is introduced in [13]. In [12], jRESP is extended to encompass PSCEL and thus to deal also with policies.
- Knowledge handling mechanisms alternative to distributed *tuple spaces* that are instead based on *constraints* are studied in [15]. It is discussed how soft constraints can be exploited to deal with partial knowledge and guarantee multi-criteria optimization.
- Quantitative variants are considered in [10]. There, a stochastic version of SCEL is introduced that enriches terms with information about actions duration, and can be used to support quantitative analysis of autonomic systems. Investigation of these issues will continue in QUANTICOL [16] another EU-FET project.
- Adaptation patterns and the possibility of modeling them via the SCEL abstractions are considered in [6]. Modelling of self-expression in SCEL is instead considered in [5].
- The extension of SCEL with reasoning capabilities that are guaranteed by external reasoners is presented in [3]. There, the solid semantics foundations of SCEL is also exploited to develop MISSCEL, an implementation of SCEL's operational semantics in MAUDE to pave the way towards using the rich verification tool set of this framework.
- In [8] it is instead shown how the SPIN model checker can be used to prove properties of SCEL programs by translating them into Promela, the input language of SPIN.
- Specific case studies taken from the automotive and cloud computing scenarios are considered in [4,14].
- A core calculus with attribute-based communication obtained by distilling the key concepts of SCEL is presented [1] with the aim of initiating fundamental studies to understand the full impact of this novel communication paradigm.

Acknowledgement. As evident from the list of the papers mentioned in the bibliography below, design, implementation and exploitation of SCEL has been a collective effort. I would like to thank all involved colleagues for their fundamental contributions. I take the occasion to thank also Ivan Lanese and Eric Madelaine for giving me the possibility of presenting our work at the FACS conference and in the proceedings.

References

1. Abd Alrahman, Y., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: Proceedings of SAC 2015 (2015, to appear)
2. ASCENS. Autonomic service-components ensemble, a FET-EU project. http://www.ascens-ist.eu/ (2014). Accessed 28 Nov 2014
3. Belzner, L., De Nicola, R., Vandin, A., Wirsing, M.: Reasoning (on) service component ensembles in rewriting logic. In: Iida, S., Meseguer, J., Ogata, K. (eds.) Specification, Algebra, and Software. LNCS, vol. 8373, pp. 188–211. Springer, Heidelberg (2014)
4. Bures, T., De Nicola, R., Gerostathopoulos, I., Hoch, N., Kit, M., Koch, N., Valentina Monreale, G., Montanari, U., Pugliese, R., Serbedzija, N. B., Wirsing, M., Zambonelli, F.: A life cycle for the development of autonomic systems: the e-mobility showcase. In: Proceedings of SASOW, pp. 71–76. IEEE (2013)
5. Cabri, G., Capodieci, N., Cesari, L., De Nicola, R., Pugliese, R., Tiezzi, F., Zambonelli, F.: Self-expression and dynamic attribute-based ensembles in SCEL. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014, Part I. LNCS, vol. 8802, pp. 147–163. Springer, Heidelberg (2014)
6. Cesari, L., De Nicola, R., Pugliese, R., Puviani, M., Tiezzi, F., Zambonelli, F.: Formalising adaptation patterns for autonomic ensembles. In: Fiadeiro, J.L., Liu, Z., Xue, J. (eds.) FACS 2013. LNCS, vol. 8348, pp. 100–118. Springer, Heidelberg (2014)
7. De Nicola, R., Ferrari, G.L., Pugliese, R.: Klaim: a kernel language for agents interaction and mobility. IEEE Trans. Softw. Eng. **24**(5), 315–330 (1998)
8. De Nicola, R., Lluch Lafuente, A., Loreti, M., Morichetta, A., Pugliese, R., Senni, V., Tiezzi, F.: Programming and verifying component ensembles. In: Bensalem, S., Lakhneck, Y., Legay, A. (eds.) From Programs to Systems. LNCS, vol. 8415, pp. 69–83. Springer, Heidelberg (2014)
9. De Nicola, R., Loreti, M., Pugliese, R., Tiezzi, F.: A formal approach to autonomic systems programming: the SCEL language. TAAS **9**(2), 7 (2014)
10. Latella, D., Loreti, M., Massink, M., Senni, V.: Stochastically timed predicate-based communication primitives for autonomic computing. In: Bertrand, N., Bortolussi, L. (eds.) Proceedings of QAPL 2014, Electronic Proceedings in Theoretical Computer Science, EPTCS, 2014, pp. 1–16. ISSN: 2075–2180, doi:10.4204/EPTCS.154.1
11. Loreti, M.: jRESP: a run-time environment for SCEL programs. Technical report, Sept 2014. http://rap.dsi.unifi.it/scel/ and http://code.google.com/p/jresp/
12. Loreti, M., Margheri, A., Pugliese, R., Tiezzi, F.: On programming and policing autonomic computing systems. In: Margaria, T., Steffen, B. (eds.) ISoLA. LNCS, vol. 8802, pp. 164–183. Springer, Heidelberg (2014)
13. Margheri, A., Pugliese, R., Tiezzi, F.: Linguistic abstractions for programming and policing autonomic computing systems. In: UIC/ATC, pp. 404–409. IEEE (2013)
14. Mayer, P., Klarl, A., Hennicker, R., Puviani, M., Tiezzi, F., Pugliese, R., Keznikl, J., Bures, T.: The autonomic cloud: a vision of voluntary, peer-2-peer cloud computing. In: Proceedings of SASOW, pp. 89–94. IEEE (2013)
15. Montanar, U., Pugliese, R., Tiezzi, F.: Programming autonomic systems with multiple constraint stores. In: De Nicola, R., Hennicker, R. (eds.) Software, Services and Systems. LNCS. Springer, Heidelberg (2015)
16. QUANTICOL. A quantitative approach to management and design of collective and adaptive behaviours, a FET-EU project. http://blog.inf.ed.ac.uk/quanticol/. Accessed 28 Nov 2014