Supervisory Synthesis of Configurable Behavioural Contracts with Modalities

Davide Basile, Maurice Beek, Pierpaolo Degano, Axel Legay, Gian-Luigi Ferrari, Stefania Gnesi, Felicita Di Giandomenico

Supervisory Synthesis of Configurable Behavioural Contracts with Modalities

D. Basile[1](✉), M.H. ter Beek[1], P. Degano[2], A. Legay[3], G.-L. Ferrari[2], S. Gnesi[1], and F. Di Giandomenico[1]

[1] ISTI–CNR, Pisa, Italy
[2] University of Pisa, Pisa, Italy
[3] UCLouvain, Louvain-la-Neuve, Belgium

Abstract. Service contracts characterise the desired behavioural compliance of a composition of services, typically defined by the fulfilment of all service requests through service offers. Contract automata are a formalism for specifying behavioural service contracts. Based on the notion of synthesis of the most permissive controller from Supervisory Control Theory, a safe orchestration of contract automata can be computed that refines a composition into a compliant one. This short paper summarises the contributions published in [8], where we endow contract automata with two orthogonal layers of variability: (i) at the structural level, constraints over service requests and offers define different configurations of a contract automaton, depending on which requests and offers are selected or discarded; and (ii) at the behavioural level, service requests of different levels of criticality can be declared, which induces the novel notion of semi-controllability. The synthesis of orchestrations is thus extended to respect both the structural and the behavioural variability constraints. Finally, we show how to efficiently compute the orchestration of all configurations from only a subset of these configurations. A recently redesigned and refactored tool supports the developed theory.

Extended Abstract

A contract automaton [4] represents a single service (a *principal*) or a multi-party composition of services [11,2]. Each principal's goal is to reach an accepting state by matching its service request actions with corresponding service offer actions of other principals. An orchestration is synthesised from the principals to only allow finite executions in agreement, i.e., each request action a is fulfilled by an offer action $\overline{a}$. Technically, such an orchestration is synthesised as the *most permissive controller* (mpc) known from Supervisory Control Theory (SCT) [23,15].

Automata $\mathcal{A}_1$ and $\mathcal{A}_2$ in Fig. 1(left) interact on a service action a. Their composition $\mathcal{A}_1 \otimes \mathcal{A}_2$ in Fig. 1(right) models two possible ways to fulfill service request a from $\mathcal{A}_1$ by matching it with a service offer $\overline{a}$ of $\mathcal{A}_2$, represented as $(a, \overline{a})$. Assume that a must be matched with $\overline{a}$ to obtain agreement, and that for some reason the state ϟ is to be avoided in favour of state ✓. In most automata-based formalisms, including the contract automata of [4,7], this is typically not allowed by the notion of *uncontrollability*, and thus the resulting mpc is empty.

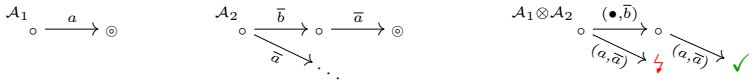

Fig. 1. Two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ and a possible composition $\mathcal{A}_1 \otimes \mathcal{A}_2$

In [8], we introduce a way to express that a must *eventually* be matched, rather than *always*, by defining contract automata in which it is possible to orchestrate the composition of $\mathcal{A}_1$ and $\mathcal{A}_2$ such that the result is similar to the composition $\mathcal{A}_1 \otimes \mathcal{A}_2$ depicted in Fig. 1 but *without state* ↯, i.e., a is only matched with $\overline{a}$ *after* the occurrence of an unmatched service offer $\overline{b}$ of $\mathcal{A}_2$, i.e., $(\bullet, \overline{b})$.

Technically, in [8] we extend contract automata with action modalities to distinguish *permitted* from *necessary* service requests (borrowed from [7]). Permitted and necessary request actions differ in that the latter *must* be fulfilled, while the former *may* also be omitted. As in [7], we assume service offer actions to be always permitted because a service contract may *always* withdraw its offers that are not needed to reach an agreement. Furthermore, we endow contract automata with two orthogonal *variability* mechanisms.

The first variability mechanism concerns constraints operating on the entire service contract, i.e., at the *structural* level, to define different configurations. This is important because services are typically reused in configurations that vary over time and need to be adapted to changing environments. Such configurations are characterised by which service actions are *mandatory* and which *forbidden*. The *valid* configurations are those respecting all structural constraints. We follow the well-established paradigm of Software Product Line Engineering (SPLE), which aims at efficiently managing a family of highly configurable systems to allow for mass customisation [22,1]. To compactly represent a *product line*, i.e., the set of *valid* product configurations, we use a so-called *feature constraint*, a propositional formula φ whose atoms are features [19,10,16] and we identify features as service actions (offers as well as requests). Usually, in SPLE, each feature is either selected or discarded to configure a product, i.e., all variability is resolved and the interpretation of the atoms of φ is *total*. Instead, we consider as valid those products (called *sub-families* in SPLE terms) that are defined by a *partial* assignment satisfying φ. This enables to synthesise the orchestration of an entire product line by considering a few valid products only (those such that their union contains all possible behaviour of the product line's orchestration), rather than computing *all* the valid ones (and retaining unnecessary complexity due to duplicated behaviour). This is one of the main results of [8].

The second variability mechanism is defined inside service contracts, i.e., at the *behavioural* level, to declare necessary request actions to be either *urgent* or *lazy*. These modalities drive the orchestrator to fulfill *all the occurrences* of an urgent action, which is the classical notion of uncontrollability from SCT, while it is required to fulfill *at least one occurrence* of lazy actions, which is the novel notion of *semi-controllability* useful for orchestration synthesis. The simplistic example above has no urgent action; the only necessary one is the lazy request a. Intuitively, the matching of a lazy request may be delayed whereas this is not the case for urgent requests. Obviously, a must not be forbidden, either directly or because it is not part of any valid configuration.

To effectively use the variability mechanisms, we refine the classical synthesis algorithm from SCT [23]. We compute the orchestrations of a single *valid* configuration, i.e., including all mandatory and none of the forbidden actions, besides fulfilling all the necessary and the maximal number of permitted requests (i.e., if the orchestration were to fulfill another permitted request, then one of the other requirements would no longer be fulfilled).

Summarising, the main contributions of [8] are as follows:

1. A novel formalism for behavioural service contracts, called *Featured Modal Contract Automata* (FMCA), which offers support for both structural and behavioural variability not available before in the literature.
2. The new notion of *semi-controllability* (related to lazy actions), which refines both the notion of controllability (related to permitted actions) and that of uncontrollability (related to urgent actions) as used in classical synthesis algorithms from SCT. This new notion is fundamental to handle different service requests in the orchestration synthesis for FMCA.
3. A revised algorithm for synthesising an orchestration of services for a single valid product configuration. Each FMCA $\mathcal{A}$ is a pair made of an automaton and a feature constraint φ, which is related to the automaton in the following way. The labels on the arcs of the automaton identify the actions for requests and offers, a subset of which corresponds to all features in φ. The FMCA $\mathcal{A}$ is said to *respect* a product p whenever all features declared *mandatory* (*forbidden*, respectively) by p correspond to actions that are reachable (unreachable, respectively) from the initial state of $\mathcal{A}$.
4. An algorithm to compute the orchestration of an entire product line by joining the orchestrations of a small selected subset of valid product configurations, *without* computing the orchestration for each of its valid product configurations. Since the number of valid product configurations is known to be exponential in the number of features [13], only using few of them greatly improves performance and guarantees scalability of the novel framework of contract automata presented in [8]. The algorithm is thus more efficient than the standard ones available in the literature (e.g., cf. [12]).
5. The open-source prototypical Contract Automata Tool [5] extended to include FMCA is briefly surveyed and evaluated (cf. [6] for more details of the FMCA tool). It exploits FeatureIDE [21], an open-source framework for feature-oriented software development based on Eclipse, offering a variety of feature model editing and management tools.

The research on the formalism and its associated tool has evolved since [8]. As reported in [3], the tool has recently been redesigned according to the principles of model-based systems engineering [24,18] and of writing clean and readable code [20,14], and it has been refactored using lambda expressions and Java Streams as available in Java 8 [17,25], exploiting parallelism. Also, the abstract parametric synthesis algorithm from [9] has been implemented. The current version is available at https://github.com/davidebasile/ContractAutomataTool and previous implementations are still available in other branches of the repository.

References

1. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer (2013)
2. Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in behavioural contracts: A brief survey. In: Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 103–121 (2015)
3. Basile, D., ter Beek, M.H.: A clean and efficient implementation of choreography synthesis for behavioural contracts. In: COORDINATION. LNCS, vol. 12717 (2021)
4. Basile, D., Degano, P., Ferrari, G.L.: Automata for specifying and orchestrating service contracts. Log. Meth. Comput. Sci. **12**(4) (2016)
5. Basile, D., Degano, P., Ferrari, G.L., Tuosto, E.: Playing with our CAT and communication-centric applications. In: FORTE. LNCS, vol. 9688, pp. 62–73 (2016)
6. Basile, D., Di Giandomenico, F., Gnesi, S.: FMCAT: Supporting dynamic service-based product lines. In: SPLC. pp. 3–8. ACM (2017)
7. Basile, D., Di Giandomenico, F., Gnesi, S., Degano, P., Ferrari, G.L.: Specifying variability in service contracts. In: VaMoS. pp. 20–27. ACM (2017)
8. Basile, D., ter Beek, M.H., Degano, P., Legay, A., Ferrari, G.L., Gnesi, S., Di Giandomenico, F.: Controller synthesis of service contracts with variability. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102344
9. Basile, D., ter Beek, M.H., Pugliese, R.: Synthesis of orchestrations and choreographies: Bridging the gap between supervisory control and coordination of services. Log. Methods Comput. Sci. **16**(2) (2020)
10. Batory, D.S.: Feature models, grammars, and propositional formulas. In: SPLC. LNCS, vol. 3714, pp. 7–20 (2005)
11. ter Beek, M.H., Bucchiarone, A., Gnesi, S.: Web service composition approaches: From industrial standards to formal methods. In: ICIW. IEEE (2007)
12. ter Beek, M.H., Reniers, M.A., de Vink, E.P.: Supervisory controller synthesis for product lines using CIF 3. In: ISoLA. LNCS, vol. 9952, pp. 856–873 (2016)
13. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. **35**(6), 615–636 (2010)
14. Boswell, D., Foucher, T.: The Art of Readable Code. Reilly (2011)
15. Caillaud, B., Darondeau, P., Lavagno, L., Xie, X. (eds.): Synthesis and Control of Discrete Event Systems. Springer (2002)
16. Czarnecki, K., Wąsowski, A.: Feature diagrams and logics: There and back again. In: SPLC. pp. 23–34. IEEE (2007)
17. Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., Lea, D.: Java Concurrency in Practice. Addison-Wesley (2006)
18. Henderson, K., Salado, A.: Value and benefits of model-based systems engineering (MBSE): Evidence from the literature. Syst. Eng. **24**(1), 51–66 (2021)
19. Mannion, M.: Using first-order logic for product line model validation. In: SPLC. LNCS, vol. 2379, pp. 176–187 (2002)
20. Martin, R.C.: Clean Code. Prentice Hall (2008)
21. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer (2017)
22. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer (2005)
23. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25**(1), 206–230 (1987)
24. Tockey, S.: How to Engineer Software: A Model-Based Approach. Wiley (2019)
25. Warburton, R.: Java 8 Lambdas: Pragmatic Functional Programming. Reilly (2014)