Composition of Nondeterministic Services for LTL$_f$ Task Specification

Giuseppe De Giacomo[1,2], Marco Favorito[3] and Luciana Silo[2,4]

[1] University of Oxford, UK
[2] Sapienza University of Rome, Italy
[3] Banca d'Italia, Italy
[4] Camera dei Deputati, Italy

Abstract
In this paper, we study the composition of services so as to obtain runs satisfying a task specification in Linear Temporal Logic on finite traces (LTL$_f$). We study the problem in the case services are nondeterministic and the LTL$_f$ specification can be exactly met. To do so, we combine techniques from LTL$_f$ synthesis, service composition *à la* Roman Model and reactive synthesis.

Keywords
Service Composition, Linear Temporal Logic on finite traces, LTL$_f$ Synthesis

1. Introduction

The service-oriented computing (SOC) paradigm uses services to support the development of rapid, low-cost, interoperable, evolvable, and massively distributed applications. Services are considered autonomous, platform-independent entities that can be described, published, discovered, and loosely coupled in novel ways [1]. Service composition, i.e. the ability to generate new, more useful services from existing ones, is an active field of research in the SOC area and has been actively investigated for over a decade. Particularly interesting in this context is the so-called Roman Model [2, 3, 4, 5] where services are *conversational*, i.e., have an internal state and are modeled as finite state machines (FSM), where at each state the service offers a certain set of actions, and each action changes the state of the service in some way. The designer is interested in generating a new service, called *target*, from the set of existing services specified using an FSM, too. The goal is to see whether the target can be satisfied by properly orchestrating the work of the component service and building a scheduler called the orchestrator that will use actions provided by existing services to implement action requests.

In this paper, we consider a variant of the Roman Model where the composition is *task-oriented*, and this makes it more similar to Planning in AI [6, 7, 8]. Specifically, we are given a task, and we want to synthesize an orchestrator that, on the one hand, reactively chooses actions to form a sequence that satisfies the task and, on the other hand, delegates each action

OVERLAY 2023: 5th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, November 7, 2023, Rome, Italy
giuseppe.degiacomo@cs.ox.ac.uk (G. D. Giacomo); marco.favorito@bancaditalia.it (M. Favorito); silo@diag.uniroma1.it (L. Silo)

to an available service in such a way that at the end of the sequence, all services are in their final states. We consider the available services as nondeterministic, in the sense that when the orchestrator delegates to them an action, they will change state in a nondeterministic (devilish vs. angelic) way, as studied, e.g. in [3]. We draw from the work on declarative process modeling in Business Process Management (BPM) in which the task specification is expressed in Linear Temporal Logic on finite traces (LTL_f) [9] with the so-called DECLARE assumption that only one action can be selected at each point in time [10]. This gives us a rich way to specify dynamic tasks that extend over time. We give a formal definition of composition and a provably correct technique to actually solve the composition problem and obtain the orchestrator. The technique is readily implementable. The solution technique is based on finding a winning strategy for a two-player game over a particular DFA game, as done, for example, in LTL_f synthesis [11]. Although this paper has a foundational nature, we observe that these kinds of task-oriented compositions are increasingly becoming important in smart manufacturing [12].

2. Composition of Nondeterministic Services for LTL_f Tasks

In this section we present our service composition in the case the available service are nondeterministic. Unlike the classical Roman model, we do not have an explicit specification of the target service to realize, but rather, a high-level specification of a task to accomplish expressed as an LTL_f formula.

2.1. Nondeterministic Services Framework

In the Roman Model [2], each *(available) service* is defined as a tuple $\mathcal{S} = \langle \Sigma, A, \sigma_0, F, \delta \rangle$ formed respectively by: a finite set Σ of service states, a finite set A of services' actions, an initial state σ_0, a set of final states F (i.e., states in which the computation may stop, but does not necessarily have to), and a transition relation $\delta \subseteq \Sigma \times A \times \Sigma$. For convenience, we define $\delta(\sigma, a) = \{\sigma' \mid (\sigma, a, \sigma') \in \delta\}$, and we assume that for each state $\sigma \in \Sigma$ and each action $a \in A$, there exist $\sigma' \in \Sigma$ such that $(\sigma, a, \sigma') \in \delta$ (possibly σ' is an error state σ_u that will never reach a final state). Actions in A denote interactions between service and clients. The behaviour of each available service is described in terms of a finite transition system that uses only actions from A. Consider a task specification φ expressed in LTL_f over the set of propositions A, and consider a community of n services $\mathcal{C} = \{\mathcal{S}_1, \dots, \mathcal{S}_n\}$. An *infinite trace of* $\mathcal{C}$ is an infinite alternating sequence of the form $t = (\sigma_{10} \dots \sigma_{n0}), (a_1, o_1), (\sigma_{11} \dots \sigma_{n1}), (a_2, o_2) \dots$ where for every $0 \leq k$, we have (i) $\sigma_{ik} \in \Sigma_i$ for all $i \in \{1, \dots, n\}$, (ii) $o_k \in \{1, \dots, n\}$, (iii) $a_k \in A$, and (iv) for all i, $\sigma_{i,k+1} = \delta_i(\sigma_{ik}, a_{k+1})$ if $o_{k+1} = i$, and $\sigma_{i,k+1} = \sigma_{ik}$ otherwise. A *history of* $\mathcal{C}$ is a finite prefix of a trace of $\mathcal{C}$. Given a trace t, we call $\mathsf{states}(t)$ *sequence of states* of t, i.e. $\mathsf{states}(t) = (\sigma_{10} \dots \sigma_{n0}), (\sigma_{11} \dots \sigma_{n1}), \cdots$. The *choices* of a trace t, denoted with $\mathsf{choices}(t)$, is the sequence of actions in t, i.e. $\mathsf{choices}(t) = (a_1, o_1), (a_m, o_m), \dots$. Moreover, we define the *action run* of a trace t, denoted with $\mathsf{actions}(t)$, the projection of $\mathsf{choices}(t)$ only to the components in A. Due to nondeterminism, there might be many traces of $\mathcal{C}$ associated with the same action run. states, choices and actions are defined also on history h, in a similar way.

An *orchestrator* is a function $\gamma : (\Sigma_1 \times \cdots \times \Sigma_n)^* \to A \times \{1 \dots n\}$ that, given a sequence of states, returns the action to perform, and the service (actually the service index) that will perform it. Given a trace t, with $\mathsf{histories}(t)$, we denote the set of prefixes of the trace t that ends with a services state configuration. A trace t is an *execution* of an orchestrator γ over $\mathcal{C}$ if for all $k \geq 0$, we have $(a_{k+1}, o_{k+1}) = \gamma((\sigma_{10} \dots \sigma_{n0}) \dots (\sigma_{1k} \dots \sigma_{nk}))$. If we consider $\mathcal{T}_{\gamma,\mathcal{C}}$ be the set of such executions we can have many executions for the same orchestrator, despite the orchestrator being a deterministic function (due nondeterminism of the services). If $h \in \mathsf{histories}(t)$ for some (infinite) execution $t \in \mathcal{T}_{\gamma,\mathcal{C}}$, we call h a finite execution of γ over $\mathcal{C}$. We say that some finite execution h is *successful*, denoted with $\mathsf{successful}(h)$, if the following two conditions hold: (1) $\mathsf{actions}(h) \models \varphi$, and (2) all service state $\sigma_i \in \mathsf{last}(\mathsf{states}(h))$ are such that $\sigma_i \in F_i$. If for execution $t \in \mathcal{T}_{\gamma,\mathcal{C}}$ there exist a finite prefix history $h \in \mathsf{histories}(t)$ such that $\mathsf{successful}(h)$, we say that t is successful. Finally, we say that an orchestrator γ *realizes the* LTL$_f$ *specification* φ *with* $\mathcal{C}$ if, for all traces $t \in \mathcal{T}_{\gamma,\mathcal{C}}$, t is successful. Since the orchestrator at every step chooses the action and the (index of the) service to which the action is delegated, it guarantees the (complete) sequence of actions that satisfy the LTL$_f$ task specification. Hence, when the orchestrator stops, all services are left in their final states.

The composition problem is: given the pair $(\mathcal{C}, \varphi)$, where φ is an LTL$_f$ task specification over the set of propositions A, and $\mathcal{C}$ is a community of n services $\mathcal{C} = \{\mathcal{S}_1, \dots, \mathcal{S}_n\}$, compute, if it exists, an orchestrator γ that realizes φ.

2.2. Nondeterministic Services Solution Technique

To synthesize the orchestrator we rely on a game-theoretic technique: (*i*) we build a game arena where the *controller* (the orchestrator) and the *environment* (the service community) play as adversaries; (*ii*) we synthesize a strategy for the controller to win the game whatever the environment does; (*iii*) from this strategy we will build the actual orchestrator. Specifically, we proceed according to the following steps.

Step (1) First, from the LTL$_f$ task specification we compute the equivalent Nondeterministic Finite Automaton (NFA) of an LTL$_f$ formula $\mathcal{A}_\varphi = (A, Q, q_0, F, \delta)$ using the LTL$_f$2NFA algorithm [13]. Only one action is executed at each time instant.

Step (2) From this NFA we define a *controllable Deterministic Finite Automaton (*DFA*)* on the alphabet $A \times Q$, $\mathcal{A}_{\mathsf{act}} = (A \times Q, Q, q_0, F, \delta_{\mathsf{act}})$, where everything is as in $\mathcal{A}_\varphi$ except δ_{act}, with which can give the control of the transition to the controller. This means that for every sequence of actions $a_1, \dots, a_n$ accepted by the NFA $\mathcal{A}_\varphi$, there exists a corresponding alternating sequence $q_0, a_1, \dots, q_n$ accepted by the DFA $\mathcal{A}_{\mathsf{act}}$, and viceversa. In other words, when we project out the Q-component from the accepted sequences of $\mathcal{A}_{\mathsf{act}}$, we get a sequence of actions satisfying φ.

Step (3) Then, we compute the *product of such* DFA $\mathcal{A}_{\mathsf{act}}$ *with the services*, obtaining the *composition* DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ again extending the alphabet with new symbols, which this time are under the control of the environment. Intuitively, the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ over alphabet A' is a synchronous cartesian product between the NFA $\mathcal{A}_\varphi$ and the service $\mathcal{S}_i$ chosen by the current symbol $(a, q, i, \sigma) \in A'$. The "angelic" nondeterminism of $\mathcal{A}_\varphi$ and the "devilish" nondeterminism coming from the services is cancelled by moving the choice of the next NFA state and the next system service state in the alphabet A'. It can be shown that there is a relationship between the accepting runs of the DFA $\mathcal{A}_{\varphi,\mathcal{C}}$ and the set of successful executions of some orchestrator γ over community $\mathcal{C}$ for the specification φ.

Step (4) The DFA obtained is the arena over which we play the so-called DFA *game* [11]. It is a game between two players: the environment and the controller. $\mathcal{X}$ is the set of uncontrollable symbols (under the control of the environment), $\mathcal{Y}$ is the set of controllable symbols (under the control of the controller). This is a classical problem and can be solved as follows. First, we define the *controllable preimage* $PreC(\mathcal{E})$ of a set $\mathcal{E}$ of states of $\mathcal{G}$ as the set of states s s.t. there exists a choice for symbols $\mathcal{Y}$ s.t. for all choices of symbols $\mathcal{X}$, game $\mathcal{G}$ progresses to states in $\mathcal{E}$. Then, we define the set $Win(\mathcal{G})$ of winning states of a DFA game $\mathcal{G}$, i.e., the set formed by the states from which the controller can win the DFA game $\mathcal{G}$. Specifically, $Win(G)$ is defined as the least-fixpoint, making use of approximates $Win_k(\mathcal{G})$ denoting all states where the controller wins in at most k steps: (1) $Win_0(\mathcal{G}) = F$ (the final states of $\mathcal{G}$); and (2) $Win_{k+1}(\mathcal{G}) = Win_k(\mathcal{G}) \cup PreC(Win_k(\mathcal{G}))$. Then, $Win(\mathcal{G}) = \bigcup_k Win_k(\mathcal{G})$. Computing $Win(\mathcal{G})$ requires linear time in the number of states in $\mathcal{G}$. Indeed, after at most a linear number of steps $Win_{k+1}(\mathcal{G}) = Win_k(\mathcal{G}) = Win(\mathcal{G})$. It can be shown that a DFA game $\mathcal{G}$ admits a winning strategy iff $s_0 \in Win(\mathcal{G})$, and the resulting strategy is a transducer $T = (\mathcal{X} \times \mathcal{Y}, Q', q'_0, \delta_T, \theta_T)$ formed by: the input alphabet $\mathcal{X} \times \mathcal{Y}$, the set of states Q', the initial state q'_0, the transition function $\delta_T : Q' \times \mathcal{X} \to Q'$ s.t. $\delta_T(q, X) = \delta'(q, (X, \theta(q))$, and $\theta_T : Q \to \mathcal{Y}$ is the output function defined as $\theta_T(q) = Y$ s.t. if $q \in Win_{i+1}(\mathcal{G}) \setminus Win_i(\mathcal{G})$ then $\forall X.\delta(q, (X, Y)) \in Win_i(\mathcal{G})$ [11]. Given a strategy in the form of a transducer T, we can obtain an orchestrator that realizes the specification as follows. Let the *extended transition function* δ^*_T of T is $\delta^*_T(q, \varepsilon) = q$ and $\delta^*_T(q, wa) = \delta_T(\delta^*_T(q, w), a)$. Then, for every sequence w of length $m \geq 0$ $w = (X_1, Y_1) \dots (X_m, Y_m)$, where for each index k, Y_k and X_k are of the form (a_k, q_k, o_k) and $\sigma_{o_k,k}$ respectively, we define the orchestrator $\gamma_T((\sigma_{10} \dots \sigma_{n0}), (\sigma_{11} \dots \sigma_{o_1,1} \dots \sigma_{n1}), \dots (\sigma_{1m} \dots \sigma_{o_k,m} \dots \sigma_{nm})) = (a_{m+1}, o_{m+1})$, where $(a_{m+1}, q_{m+1}, o_{m+1}) = \theta_T(\delta^*_T(q_0, w))$. We can reduce the problem of service composition for LTL$_f$ task specifications to solving the DFA game over $\mathcal{A}_{\varphi,\mathcal{C}}$ with uncontrollable symbols $\mathcal{X} = \bigcup_i \Sigma_i$ and controllable symbols $\mathcal{Y} = A \times Q \times \{1, \dots, n\}$. It can be shown that the *service composition realizability* problem with community $\mathcal{C}$ for the satisfaction of an LTL$_f$ task specification φ can be solved by checking whether $q'_0 \in Win(\mathcal{A}_{\varphi,\mathcal{C}})$, and that the problem can be solved in at most exponential time in the size of the formula, in at most exponential time in the number of services, and in polynomial time in the size of the services.

3. Conclusion and Future Works

In this paper, we have studied an advanced form of task-oriented compositions of nondeterministic services. In future works, we want to analyze the composition in stochastic settings. In this case, we will model nondeterminism using probability distributions over the services' successor states by considering the objective of maximizing the satisfaction probability of the specification. The services will be considered stochastic, in the sense that delegated actions change the service state according to a probability distribution, as studied, e.g., in [14, 4]. The solution technique will solve a bi-objective lexicographic optimization [15] over a special Markov Decision Process [16], allowing to minimize the services' utilization costs while guaranteeing maximum probability of task satisfaction. Moreover, since service composition has become very relevant in smart manufacturing [17, 18], we want to expand the use of it in a Digital Twins (DT) scenario as in [12].

Acknowledgements

This work has been partially supported by the EU H2020 project AIPlan4EU (No. 101016442), the ERC-ADG White- Mech (No. 834228), the EU ICT-48 2020 project TAILOR (No. 952215), the PRIN project RIPER (No. 20203FFYLK), and the PNRR MUR project FAIR (No. PE0000013).

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: State of the art and research challenges, Computer 40 (2007) 38–45.

[2] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, Automatic composition of e-services that export their behavior, in: ICSOC, 2003.

[3] D. Berardi, D. Calvanese, G. De Giacomo, M. Mecella, Composition of services with nondeterministic observable behavior, in: ICSOC, 2005.

[4] R. I. Brafman, G. De Giacomo, M. Mecella, S. Sardina, Service composition in stochastic settings, in: AIxIA, 2017.

[5] G. De Giacomo, M. Mecella, F. Patrizi, Automated service composition based on behaviors: The Roman model, in: Web services foundations, 2014.

[6] H. Geffner, B. Bonet, A Concise Introduction to Models and Methods for Automated Planning, Morgan & Claypool Publishers, 2013.

[7] S. A. McIlraith, T. C. Son, Adapting golog for composition of semantic web services, in: KR, Morgan Kaufmann, 2002, pp. 482–496.

[8] M. Pistore, A. Marconi, P. Bertoli, P. Traverso, Automated composition of web services by planning at the knowledge level, in: IJCAI, 2005, pp. 1252–1259.

[9] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: IJCAI, 2013.

[10] M. Pesic, H. Schonenberg, W. M. Van der Aalst, Declare: Full support for loosely-structured processes, in: EDOC, 2007.

[11] G. De Giacomo, M. Y. Vardi, Synthesis for LTL and LDL on finite traces, in: IJCAI, 2015.

[12] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, L. Silo, Digital twin composition in smart manufacturing via markov decision processes, Comput. Ind. 149 (2023).

[13] R. I. Brafman, G. De Giacomo, F. Patrizi, Ltlf/ldlf non-markovian rewards, in: AAAI, 2018.

[14] N. Yadav, S. Sardina, Decision theoretic behavior composition, in: AAMAS, 2011.

[15] D. Busatto-Gaston, D. Chakraborty, A. Majumdar, S. Mukherjee, G. A. Pérez, J.-F. Raskin, Bi-objective lexicographic optimization in markov decision processes with related objectives, arXiv preprint arXiv:2305.09634 (2023).

[16] M. L. Puterman, Markov Decision Processes, 1994.

[17] G. De Giacomo, P. Felli, B. Logan, F. Patrizi, S. Sardiña, Situation calculus for controller synthesis in manufacturing systems with first-order state representation, Artif. Intell. 302 (2022) 103598.

[18] G. De Giacomo, M. Y. Vardi, P. Felli, N. Alechina, B. Logan, Synthesis of orchestrations of transducers for manufacturing, in: AAAI, 2018.