Building Virtual Behaviors from Partially Controllable Available Behaviors in Nondeterministic Environments

Giuseppe De Giacomo and **Fabio Patrizi**
Sapienza Universita' di Roma
Roma, Italy
{degiacomo,patrizi}@dis.uniroma1.it

Sebastian Sardina
RMIT University
Melbourne, Australia
sebastian.sardina@rmit.edu.au

Abstract

The composition problem involves how to coordinate a set of available modules (e.g., concrete devices installed in a smart house, such as video cameras, lights, blinds, etc.) so as to implement a desired but non-existent target complex component (e.g., a complex entertainment house system). This paper summarizes the results in (De Giacomo, Patrizi, and Sardina 2013), by formally defining the problem within an AI context, characterizing its complexity, and identifying effective techniques to solve it. Related results are also briefly discussed.

Introduction

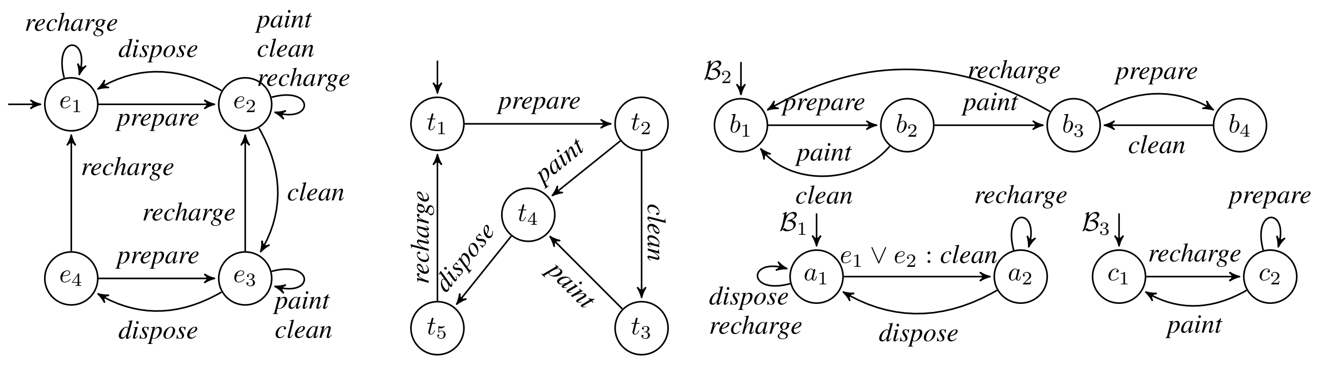
With computers now embedded in everyday devices and environments like mobile phones, cars and planes, houses, offices, and factories, the trend is to build complex systems from a collection of simple components. For example, complex entertainment systems within a smart house can be "realised" (i.e., implemented) by suitably coordinating the behaviour of a plethora of simple devices and artifacts—lights, phones, game consoles, TVs, music systems, etc.—installed in the house. Such embedded systems can provide services that range from simple tasks, such as "turn on the lights in the bedroom," to more complex ones, such as "bring me a cup of coffee" or "handle house intruder" (by tracking and taking pictures of the intruder, toggling lights rapidly, and alerting the owner by email or phone).

The problem of automatically synthesizing such a controller-coordinator for a desired target (complex) system is called the *behaviour composition* problem. Informally, the problem amounts to realizing an abstract desired target behavior module (e.g., a smart house system or complex web-service) by reusing and re-purposing a set of accessible modules implementing certain concrete behaviors (e.g., installed artifacts in the house or available web-services). The question then is whether it is possible, and if so synthesise a controller, to coordinate and execute the existing available behavior modules so that it appears as if the target module is being run. Behaviours here refer to the *operational logic* of a system (e.g., a vacuum cleaner, microwave, or web-service) and are generally represented using transition systems.

This paper summarizes the results on behavior composition presented in (De Giacomo, Patrizi, and Sardina 2013) and discusses further related developments. Such work formally defines the composition problem in an AI context, characterizes its complexity, developes effective techniques to solve it, and identifies links with several areas of Computer Science and AI. When looking at behavior composition from an AI perspective, actual controllability of available behaviors becomes a prominent issue. While one can instruct a behavior module to carry out an action, the actual outcome of the action may not always be foreseen a priori, though it can possibly be observed after execution. While the work presented here is based on revisiting a certain stream of work in service composition (Berardi et al. 2003), the issue of dealing with partial controllability (of behaviors) becomes central one.

Behavior composition is strongly related to several forms of (advanced) automated planning, in particular, to *planning for temporally extended goals* (Bacchus and Kabanza 1998) as well as *fully-observable non-deterministic (FOND) planning* (Daniele, Traverso, and Vardi 2000). The former investigates techniques for building finite or infinite plans that satisfy linear- or branching-time specifications, while the latter studies the planning problem in the context of actions whose effects cannot be fully determined a priori. Indeed, the composition problem requires an advanced conditional plan (with loops) that always guarantees all possible target requests to be "served," which is, ultimately, a (temporal) invariant property. What is more, as later proved by (Ramirez, Yadav, and Sardina 2013), such plans amount to strong-cyclic policies, the general solution concepts for FOND planning. Even more, the solutions obtained via the simulation technique developed in this work are akin to the so-called *universal plans* (Schoppers 1987), that is, plans representing every possible solution.

We shall first present the formal definition of the problem and its computational complexity. We then provide our technique based on the formal notion of simulation for synthesizing the most general kind of solutions, called *controller generators*, and show how these can deal with behavior failures. After that, we demonstrate how we can resort, in practice, to existing platforms for synthesis via model checking by recasting the composition task as a safety game. We close by discussing recent developments and open challenges.



(a) Environment $\mathcal{E}$. (b) Target arm $\mathcal{T}$. (c) Available arms $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$.

Figure 1: The painting arms system $\mathcal{S} = \langle \mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3, \mathcal{E} \rangle$ and the target arm $\mathcal{T}$. Missing guards denote set $\{e_1, e_2, e_3, e_4\}$.

The Problem

We explain the behavior composition problem using the *painting arms scenario* depicted in Figure 1. The overall aim of the system is to process blocks, one at a time, by first preparing a block, then optionally cleaning it if necessary (by using water from a special tank), then painting the block (by using paint from another tank), and finally disposing the end product. This desired process is represented by target behavior $\mathcal{T}$. Importantly, after the block is disposed for collection, the process always makes sure the water and paint tanks are recharged before processing the next block.

Unfortunately, the desired arm $\mathcal{T}$ *does not* exist in reality. Nonetheless, there are three different actual arms available: a cleaning-disposing arm $\mathcal{B}_1$ able to clean and dispose blocks; an arm $\mathcal{B}_2$ capable of preparing, cleaning, and painting blocks; and an arm $\mathcal{B}_3$ that can paint and prepare blocks. All three arms are able to trigger the tanks' recharge operation. Notice that arm $\mathcal{B}_2$ behaves non-deterministically when it comes to painting a block. This captures the modeler's incomplete information about $\mathcal{B}_2$'s internal logic.

All modules are meant to execute on a shared non-deterministic environment $\mathcal{E}$ capturing the dynamics of the domain. For example, blocks can be painted or cleaned *only* after they have been prepared, and the water tank can contain water (states e_1 and e_2) or be empty (states e_3 and e_4). Observe that $\mathcal{B}_1$ uses a cleaning action implementation which requires water available, though *clean* can still be performed by other means—as done by arm $\mathcal{B}_2$—and hence it is still legal from environment state e_3.

Formally, a <u>behavior</u> over an environment $\mathcal{E}$ with set of states E is a tuple $\mathcal{B} = \langle B, b_0, G, F, \varrho \rangle$, where:

- B is the finite set of *behavior states*;
- $b_0 \in B$ is the *behavior initial state*;
- $G \subseteq E$ is a set of *guards over* $\mathcal{E}$;
- $F \subseteq B$ is the set of *final states* (where $\mathcal{B}$ can be stopped);
- $\varrho \subseteq B \times G \times \mathcal{A} \times B$ is the *behavior transition relation*.

We write $b \xrightarrow{g,a} b'$ in $\mathcal{B}$ to denote $\langle b, g, a, b' \rangle \in \varrho$: action a can be executed by $\mathcal{B}$ in state b when the environment is in a state e such that $e \in g$, which may lead the behavior to successor state b'. A behavior $\mathcal{B}$ is <u>deterministic</u> if there are no two transitions $b \xrightarrow{g_1,a} b'$ and $b \xrightarrow{g_2,a} b''$ in $\mathcal{B}$ such that $b' \neq b''$ and $g_1 \cap g_2 \neq \emptyset$. Behaviors $\mathcal{B}_1$ and $\mathcal{B}_3$ are deterministic, but not $\mathcal{B}_2$ due to the *paint* transition in b_2.

An <u>available system</u> is a tuple $\mathcal{S} = \langle \mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E} \rangle$, where $\mathcal{B}_i$'s are all the available behaviors over the shared environment $\mathcal{E}$. Available behaviors and environment could be *non-deterministic*, and hence partially controllable. Informally, the <u>*behavior composition task*</u> is stated as follows:

> *Given a system $\mathcal{S}$ and a deterministic target behavior $\mathcal{T}$, is it possible to (partially) control the available behaviors in $\mathcal{S}$ in a step-by-step manner—by instructing them on which action to execute next and observing, afterwards, the outcome in the behavior used—so as to "realize" the desired target behavior?*

In other words, can we adequately control the system so that it appears as if one was actually executing the target module?

Formally, a <u>*controller*</u> for target $\mathcal{T}$ on system $\mathcal{S}$ is a partial function $C : \mathcal{H}_\mathcal{S} \times \mathcal{A} \mapsto \{1, \ldots, n\}$, which, given a history $h \in \mathcal{H}_\mathcal{S}$ of the available system (where $\mathcal{H}_\mathcal{S}$ is, basically, the set of all finite traces of the asynchronous product of the available behaviors) and a requested (target-compatible) action, returns the index of an available behavior to which the action in question is delegated for execution. Intuitively, a controller (fully) realizes a target behavior if for every trace (i.e., run) of the target, at every step, the controller returns the index of an available behavior that can perform the requested action. Formally, one first defines when a controller C <u>*realizes a trace*</u> of the target $\mathcal{T}$. Then, a controller C realizes the target behavior $\mathcal{T}$ *iff* it realizes all its traces. In that case, C is said to be a <u>*composition*</u> for target $\mathcal{T}$ on system $\mathcal{S}$. Being a sort of conditional planning problem, it is not surprising that deciding whether there is a composition of a target module in an available system, even when the system is fully deterministic, is EXPTIME-complete.

Composition via Simulation

One of the most significant results is that one can rely on the notion of *simulation relation* (Milner 1971) as a formal tool for solution characterization. Intuitively, a transition system S_1 *simulates* another transition system S_2, if S_1 is able to "match", step by step, all of S_2 moves during execution. At each step of execution, S_2 performs a transition among those allowed in its current state. If, regardless of how S_2 happens to execute, system S_1 can, at each step, choose a transition that "matches" the one executed by S_2, then S_1 simulates S_2. In other words, system S_1 can "mimic" system S_2.

The first step towards a simulation-based account for behavior composition is to define the notions of *enacted* modules. A module is enacted when it is run in the environment of concern. So, the *enacted target* $\mathcal{T}_{\mathcal{E}}$ represents the target behavior $\mathcal{T}$ when "enacted" in the environment $\mathcal{E}$, and is the transition system obtained by taking the syncrhnous product of $\mathcal{T}$ with $\mathcal{E}$. A state $\langle t, e\rangle$ of $\mathcal{T}_{\mathcal{E}}$ represents that the target behavior is in state t and the environment in state e. Similarly, the *enacted available system* $\mathcal{S}_{\mathcal{E}}$ is defined as the synchronous product of the environment $\mathcal{E}$ with the synchronous product of all available behaviors $\mathcal{B}_1, \ldots, \mathcal{B}_n$. A state $\langle b_1, \ldots, b_n, e\rangle$ in $\mathcal{S}_{\mathcal{E}}$ provides a snapshot of the available system stating that behavior $\mathcal{B}_i$ is in state b_i and the environment is in state e.

The second step involves adapting the simulation notion to handle the intrinsic "devilish" nature of the non-determinism in available behaviors and the environment—their evolutions cannot be totally controlled. In words, enacted system state $\langle b_1, \ldots, b_n, e\rangle$ *ND-simulates* enacted target state $\langle t, e'\rangle$ when *(i)* the states share the same environment component (i.e., $e = e'$); *(ii)* if the target is in a final state (i.e., t is final in $\mathcal{T}$), so is the system (i.e., each b_i is final in $\mathcal{B}_i$); and *(iii)* for all actions the (enacted) target behavior can execute from t, there exists a *witness* behavior $\mathcal{B}_k$ that can execute the same action in state b_k while guaranteeing, regardless of non-determinism, preservation of the ND-simulation property for successor target and system states.

The main result (Theorem 1) then states the following:

A composition controller exists for target $\mathcal{T}$ on system $\mathcal{S}$ iff *the initial state of enacted system $\mathcal{S}_{\mathcal{E}}$ ND-simulates the initial state of enacted target $\mathcal{T}_{\mathcal{E}}$.*

Put it differently, there is a way for $\mathcal{S}$, when enacted on $\mathcal{E}$, to mimic the target module $\mathcal{T}$, when enacted on $\mathcal{E}$. Besides being extremely clean, the simulation-based solution characterization has the following advantages:

1. It facilitates the use of effective techniques to compute simulation relations (Henzinger, Henzinger, and Kopke 1995)—modulo slight modifications—to solve the composition problem.
2. It provides a crisp complexity characterization (Theorem 2): composition existence can be checked in polynomial time in the number of states of the available behaviors, of the environment, and of the target behavior, and in exponential time in the number of available behaviors.
3. It eases the computation (via local checks) of the so-called *controller generator CG*, a finite state transducer that, given an action a (compliant with the target behavior), outputs, through its function ω, the set of *all* available behaviors that can perform a next and preserve ND-simulation. Importantly, such controller generator encompasses and can generate *all and only those* controllers that are compositions (Theorem 3): every controller obtained from *CG* is a composition and every composition can be obtained from *CG*.
4. It supports the systematic handling of several types of *failures* (Section 4). Firstly, controller generators can be used to generate *just-in-time* composition controllers, that is, controllers generated on-the-fly as the target and system are executed. Such controllers provide *reactive adaptability* to temporary unavailability of available behaviors and unexpected state change of behaviors and environment. Secondly, when one or more available behaviors become permanently unavailable or a new available behavior is added to the available system $\mathcal{S}$, a *parsimonious refinement* of the solution (i.e., controller generator) at hand can be performed (Theorems 7 and 8), thus avoiding a (re)computation of a new solution from scratch.

Composition as a Safety Game

The task of computing an ND-simulation relation, and thus that of synthesizing a composition, can be cast as that of checking whether a *winning strategy* exists in a so-called *safety game*. The benefit of this approach is the availability of actual systems (e.g., TLV (Pnueli and Shahar 1996), JTLV (Bloem et al. 2011)) capable of computing, in a space-efficient manner, the winning strategy of a given game.

A *safety-game* is a game played by two opponents, *system* and *environment*,[1] controlling the values assigned to a finite set of variables ranging over finite domains. The set $\mathcal{X}$ of variables is partitioned into the subsets $\mathcal{X}_s$ and $\mathcal{X}_e$, controlled, respectively, by the system and the environment. The game starts with a fixed initial assignment to all variables and every turn consists of an environment's move followed by a move of the system. The moves available at each step to the system and environment players are governed by two *transition relations*, ρ_s and ρ_e, respectively. Since such moves depend, in general, on the (current) values of all *all* variables (including those not controlled by the acting player) a player can affect, via its move, the options available next to the other player. The goal of the system is to keep the game inside a "*safe*" area, that is, a set of states where the assignments to variables satisfy a certain criterion, while that of the environment is to prevent this. In other words, the system's objective is to enforce an *invariant*. The safe area is represented by the so-called *safety-goal* formula. For instance, to guarantee that the values assigned to variables v_1 and v_2 are always different, one can use the formula $v_1 \neq v_2$. A game is said to be *winning for the system* if the system has a *strategy*—a function from the history of visited states to the next system's move—which guarantees the game to stay in the safe area, no matter how the environment plays.

The search for a system's winning strategy (in fact, all such strategies) is performed through a fixpoint computation which isolates the fragment of state space where the system can force the game to stay in the safe area. Through a reduction, this procedure can be exploited to compute an ND-simulation relation (in fact, the largest one).

[1]Despite the matching names, this environment is unrelated with the shared environments that behaviors interact with.

The idea behind te reduction is as follows. The environment player encodes *(a)* the enacted target behavior, which selects, at runtime, the action to perform next according to internal logic; and *(b)* the asynchronous execution of the available behaviors, synchronously combined with the shared environment. This approach is motivated by the fact that both the action request and the actual evolution of the available system are *not* under the system player's control. Technically, the environment player is able to control a "requested action" variable ranging over all actions available to the target and a set of variables capturing the state of all behaviors and the shared environment. The transition relation ρ_e imposes the rules on such control to mimic the enacted target and the available system. In turn, the system player is able to control a distinguished "delegation" variable which ranges over all available behaviors and states which one is to be activated at a given point (to fulfill the active request). Formally, the transition relation ρ_s allows the delegation of the current requested action to *any* behavior, even one that is not able, in its current state, to perform the action.

Finally, the goal formula expresses the fact that *it is always the case that the action currently selected by the environment is actually executable by the behavior that the system player has delegated the action to*. That is, the goal requires that all actions requested by the target behavior, according to its transition relation, can be delegated to some available behavior so that all possible future requests can be successfully delegated.

Discussion

The framework summarized in this peper can be seen as a *core* account for behavior composition, that can be extended in a number of directions. Several extensions, including distributed composition, multiple target composition, composition under partial observability, or composition with data or high-level programs were proposed (see Section 7).

An important recent development concerns the settings in which there are no composition solutions. For example, removing any of the three processing arms in Figure 1 or just removing state b_4 from module $\mathcal{B}_2$ would render the target arm $\mathcal{T}$ unrealizable. In those cases, a mere "no solution" answer may be highly unsatisfactory: one would prefer accounts for the "best" possible approach to the composition instance. Again relying on the ND-simulation notion, (Yadav and Sardina 2012; Yadav et al. 2013) proposed a solution concept, and a corresponding effective technique, as the *alternative target module* that is *closest* to, though probably less powerful than, the original target and is fully realizable. Importantly, such alternative target is provably unique.

We close by noting that besides automated planning, and synthesis in general, the behavior composition problem is of interest for several other areas of CS and AI, including intelligent multi-agents (e.g., coordination of agents' teams or plans), robot-ecologies and ambient intelligence (Bordignon et al. 2007) (e.g., to achieve advanced functionalities from a plethora of simple devices), and web-service composition (Berardi et al. 2003).

Acknowledgements The authors would like to acknowledge the support from the Sapienza Award 2013 "SPIRITLETS: SPIRITLET–based Smart spaces," and that of the Australian Research Council (under grant DP120100332).

References

Bacchus, F., and Kabanza, F. 1998. Planning for temporally extended goals. *Annals of Mathematics and Artificial Intelligence* 22(1-2):5–27.

Berardi, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Mecella, M. 2003. Automatic composition of e-Services that export their behavior. In *Proceedings of the International Joint Conference on Service Oriented Computing (ICSOC)*, 43–58.

Bloem, R.; Jobstmann, B.; Piterman, N.; Pnueli, A.; and Sa'ar, Y. 2011. Synthesis of reactive(1) designs. *Journal of Computer and System Sciences* 1–28.

Bordignon, M.; Rashid, J.; Broxvall, M.; and Saffiotti, A. 2007. Seamless integration of robots and tiny embedded devices in a PEIS-ecology. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3101–3106.

Daniele, M.; Traverso, P.; and Vardi, M. 2000. Strong cyclic planning revisited. *Recent Advances in AI Planning* 35–48.

De Giacomo, G.; Patrizi, F.; and Sardina, S. 2013. Automatic behavior composition synthesis. *Artificial Intelligence Journal* 196:106–142.

Henzinger, M. R.; Henzinger, T. A.; and Kopke, P. W. 1995. Computing simulations on finite and infinite graphs. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*, 453–462.

Milner, R. 1971. An algebraic definition of simulation between programs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 481–489.

Pnueli, A., and Shahar, E. 1996. A platform for combining deductive with algorithmic verification. In *Proceedings of the International Conference on Computer Aided Verification (CAV)*, 184–195.

Ramirez, M.; Yadav, N.; and Sardina, S. 2013. Behavior composition as fully observable non-deterministic planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 180–188.

Schoppers, M. J. 1987. Universal plans for reactive robots in unpredictable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1039–1046.

Yadav, N., and Sardina, S. 2012. Qualitative approximate behavior composition. In *Proceedings of the European Conference on Logics in Artificial Intelligence (JELIA)*, volume 7519 of *LNCS*, 450–462.

Yadav, N.; Felli, P.; Giuseppe, D.; and Sardina, S. 2013. Supremal realizability of behaviors with uncontrollable exogenous events. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1176–1182.