Automatic Synthesis of a Global Behavior from Multiple Distributed Behaviors

Sebastian Sardina
Department of Computer Science
RMIT University
Melbourne, Australia
ssardina@cs.rmit.edu.au

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

Abstract

We consider the problem of synthesizing a *team of local behavior controllers* to realize a fully controllable target behavior from a set of available partially controllable behaviors that execute distributively within a shared partially predictable, but fully observable, environment. Available behaviors stand for existing distributed components and are represented with (finite) nondeterministic transition systems. The target behavior is assumed to be fully deterministic and stands for the collective behavior that the system as a whole needs to guarantee. We formally define the problem within a general framework, characterize its computational complexity, and propose techniques to actually generate a solution. Also, we investigate the relationship between the distributed solutions and the centralized ones, in which a single global controller is conceivable.

Introduction

A novel synthesis problem (De Giacomo & Sardina 2007) was recently proposed in which a *fully controllable* target behavior module is automatically synthesized from a library of available *partially controllable* behaviors executing within a shared *partially predictable*, but fully observable, environment. The available behaviors stand for existing accessible devices or components whereas the target behavior represents the desired but non-existing (virtual) component. The question then was: can a central system (always) guarantee a specific *deterministic* overall behavior by (partially) controlling the available components in a step-by-step manner, that is, by instructing them on which action to execute next and observing, afterwards, the outcome in the device used as well as in the environment? Such synthesis problem can be recast in a variety of forms within several sub-areas of AI, including planning (Meuleau *et al.* 1999), agent-oriented programming (Georgeff & Lansky 1987), plan coordination (Katz & Rosenschein 1993), and web-service composition (McIlraith & Son 2002; Berardi *et al.* 2005).

It is not hard to see that the above problem is of particular interest in settings where the existing components are *distributed* and *independent*, and thus, not accessible as a whole. For example, a RoboCup soccer team includes multiple players acting on their own, possibly exchanging messages with each other (Bredenfeld *et al.* 2006). Robot ecologies (Tilden 1993; Saffiotti & Broxvall 2005), the development of autonomous microrobot groups consisting of many heterogeneous members exhibiting collective behavior and intelligence, also offers an excellent domain in which coordination and task distribution towards obtaining a global behavior could be of great benefit. In all those cases, however, the set of available behaviors cannot be taken as a *library* accessible from a central system—no central entity can be assumed in such applications. In fact, the most one could realistically allow under these distributed scenarios is some sort of *local* control on the existing behavior devices, together with some kind of communication among such local controller modules. Because of this, the techniques developed by De Giacomo & Sardina (2007) are not suitable for these fully distributed scenarios, since their proposed solution requires a central coordination system, referred to as the *scheduler*, that is able to access every existing behavior.

In this paper, we study the *distributed* synthesis problem in which a team of *local behavior controllers* is automatically obtained. The objective is to guarantee a global distributed behavior starting from a set of distributed behaviors acting over a shared partially predictable but fully observable environment. A local controller is able to control the operation of the single behavior it is attached to as well as to broadcast messages into a shared channel. In addressing the distributed problem, we not only need to envision a new type of (distributed) solution, but we also have to enrich the setting considerably. For instance, in a distributed context, it is unrealistic to assume that only one behavior act at each step, and hence, many concurrent actions have to be allowed.

The contributions of this paper are threefold. First, we formally define and solve the distributed synthesis problem in the general case where behaviors and the environment are represented as arbitrary (nondeterministic) finite transition systems. We characterize its computational complexity and show that when there is a solution, there is one that is finite. Second, we study the intrinsic relation between centralized solutions and distributed ones, and prove that there is no loss when we assume a distributed setting: every coordination that can be done with a centralized controller can be done with a team of local controllers, and vice-versa. Third, we show how to obtain the smallest finite local controllers.

The setting

Based on (De Giacomo & Sardina 2007), let us start by defining the formal abstract framework for our problem.

Environment We assume a shared observable environment, which provides an abstract account of the observable effects and preconditions of actions (akin to an action theory). In giving such an account, we take into consideration that, in general, we have incomplete information about the actual effects and preconditions of actions. Thus, we allow the observable environment to be *nondeterministic* in general. In that way, the incomplete information on the actual world shows up as nondeterminism in our formalization.

Formally, an *environment* $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$ is characterized by the following four entities:[1]

- $\mathcal{A}$ is a finite set of shared actions;
- E is a finite set of possible environment states;
- $e_0 \in E$ is the initial state of the environment;
- $\delta_{\mathcal{E}} \subseteq E \times 2^{\mathcal{A}}_{\emptyset} \times E$ is the transition relation among states: $\delta_{\mathcal{E}}(e, A, e')$ holds when the environment *may evolve* from state e to state e' when the actions in the non-empty set of actions A are all (concurrently) executed.

Note that our notion of environment shares a lot of similarities with the so-called "transition system" in action languages (Gelfond & Lifschitz 1998).

Behavior A behavior is essentially a program for an agent or the logic of some available device. Such a program however leaves the selection of the set of actions to perform next to the agent itself. More precisely, at each step the program presents a choice of available sets of (concurrent) actions to the agent; the agent selects one of such sets; the actions in the selected set are executed concurrently; and so on.

Obviously, behaviors are not intended to be executed on their own, but they are executed in the environment (cf. above). Hence, we equip them with the ability of testing conditions (i.e., guards) on the environment when needed.

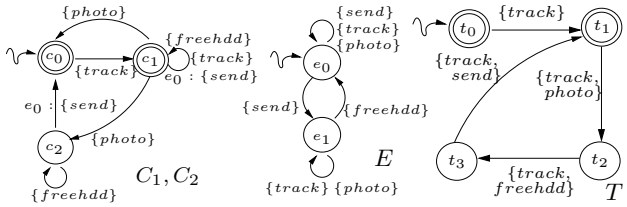
Formally, a *behavior* $\mathcal{B} = (S, s_0, G, \delta_{\mathcal{B}}, F)$ over an environment $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$ is characterized by the following give entities:

- S is a finite set of behavior states;
- $s_0 \in S$ is the single initial state of the behavior;
- G is a set of guards over the environment $\mathcal{E}$, i.e., a set of boolean functions of the form $g : E \to \{\texttt{true}, \texttt{false}\}$;
- $\delta_{\mathcal{B}} \subseteq S \times G \times 2^{\mathcal{A}}_{\emptyset} \times S$ is the behavior transition relation: $\delta_{\mathcal{B}}(s, g, A, s')$ holds when the behavior *may evolve* from state s to state s' when it (concurrently) executes all the actions in A and guard g applies in the environment;
- finally, $F \subseteq S$ is the set of states of the behavior that can be considered final, that is, the states in which the behavior can stop executing, but does not necessarily have to.

The size of a behavior is its number of states, i.e., $|\mathcal{B}| = |S|$.

Observe that, in general, behaviors are *nondeterministic* in that they may allow more than one transition with the same set of actions and compatible guards evaluating to the same truth value.[2] Thus, after making a choice of which set of actions to execute next on the behavior, one cannot be certain of which available choices there will be later on, since those will depend on what transition is actually executed—nondeterministic behaviors are only partially controllable.

We say that a behavior $\mathcal{B} = (S, s_0, G, \delta_{\mathcal{B}}, F)$, over the environment $\mathcal{E}$, is *deterministic* if there is no environment state e of $\mathcal{E}$, and no set A of actions, for which there exist two distinct transitions (s, g_1, A, s_1) and (s, g_2, A, s_2) in $\delta_{\mathcal{B}}$ such that $s_1 \neq s_2$ and $g_1(e) = g_2(e) = \texttt{true}$. Notice that given a state in a deterministic behavior and a legal set of actions, we always know exactly which is *the* next state of the behavior. In other words, deterministic behaviors are fully controllable through the selection of the set of actions to perform next, while this is not the case for nondeterministic ones.



(a) Behaviors (b) Environment (c) Target

Figure 1: A camera surveillance scenario.

The system A *system* $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ is formed by an environment $\mathcal{E}$ and n predefined nondeterministic behaviors $\mathcal{B}_i$ over $\mathcal{E}$, called the *available behaviors*. A *system configuration* is a tuple $c = (s_1, \ldots, s_n, e)$ denoting a snapshot of the system: behavior $\mathcal{B}_i$, with $i \in \{1, \ldots, n\}$, is in state s_i and the environment $\mathcal{E}$ is in state e.

Example 1 (The scenario) Consider a site surveilled by two identical cameras, C_1 and C_2, whose behaviors are represented in Fig. 1(a)—edges are labeled by expressions of the form $g : A$, where g stands for the guard and A for the set of actions (we omit g when it is equal to the boolean function `true`). Both cameras are capable of either tracking objects or taking photos—actions *track* and *photo*, respectively—but not both at the same time. Each time a photo is taken, it is stored in a local buffer which may unexpectedly become full. Since there is no way to know if the next photo will fill the buffer, action *photo* is modeled as a nondeterministic action leading the camera to state c_2 if no space is available on the local buffer and to c_0 otherwise. A camera can empty its buffer by performing a *send* action, which transfers the buffer content to a remote storage device (e.g., a hard drive). A *send* action can only be performed when space is available on such device; otherwise, the camera may ask the device for additional space by doing action *freehdd*.

The environment E keeps information about the remote storage device. The dynamics of E is depicted in Fig. 1(b): e_0 stands for the state where there is space available on the device; e_1 stands for the state where there is no space left.

The target behavior T (Fig. 1(c)) requires the ability to perform actions while keeping objects tracked, a task which could not be carried out by a single camera. In order to prevent local buffers and the remote storage device from filling up, a conservative strategy is adopted: after taking a picture, the remote device is asked for additional space and a *send* action is performed to empty the local buffer.

[1] We use abbreviation $2^{\mathcal{A}}_{\emptyset}$ to denote $2^{\mathcal{A}} - \{\emptyset\}$.

[2] Note that this kind of nondeterminism is of a *devilish* nature—the actual choice is out of the behavior control.

Behavior History A *behavior history* $h_{\mathcal{B}}$ for a given behavior $\mathcal{B} = (S, s_0, G, \delta_{\mathcal{B}}, F)$ over an environment $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$, is any finite sequence of the form $(s^0, e^0) \cdot A^1 \cdot (s^1, e^1) \cdots (s^{\ell-1}, e^{\ell-1},) \cdot A^{\ell} \cdot (s^{\ell}, e^{\ell})$, for some $\ell \geq 0$, such that for all $0 \leq k \leq \ell$ and $0 \leq j \leq \ell - 1$:

- $s^0 = s_0$ and $s^k \in S$;
- $e^0 = e_0$ and $e^k \in E$;
- $A^k \subseteq \mathcal{A}$;
- $(s^j, g, A^{j+1}, s^{j+1}) \in \delta_{\mathcal{B}}$, for some g such that $g(e^j) = \texttt{true}$, that is, behavior $\mathcal{B}$ *can* evolve from its current state s^j to state s^{j+1} w.r.t. the (current) environment state e^j;
- $(e^j, A, e^{j+1}) \in \delta_{\mathcal{E}}$, for some set of actions A such that $A^{j+1} \subseteq A$, that is, the environment can evolve from its current state e^{ℓ} to the new state $e^{\ell+1}$.

The set $\mathcal{H}_{\mathcal{B}}$ denotes the set of all behavior histories for $\mathcal{B}$.

Traces Given a behavior $\mathcal{B} = (S, s_0, G, \delta_{\mathcal{B}}, F)$ and an environment $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$, we define the *traces of $\mathcal{B}$ on $\mathcal{E}$* as the sequences of the form $t = (g^1, A^1) \cdot (g^2, A^2) \cdots$, where $g^i \in G$ and $A^i \subseteq \mathcal{A}$, such that there exists a behavior history $(s^0, e^0) \cdot A^1 \cdot (s^1, e^1) \cdot A^2 \cdots$ for $\mathcal{B}$ over $\mathcal{E}$ where $g^i(e^{i-1}) = \texttt{true}$ for all $i \geq 1$. In addition, if the trace $t = (g^1, A^1) \cdots (g^{\ell}, A^{\ell})$ is finite, then there exists a finite behavior history $(s^0, e^0) \cdot A^1 \cdots A^{\ell} \cdot (s^{\ell}, e^{\ell})$ as above such that $s^{\ell} \in F$.

The *traces of the deterministic behaviors* are of particular interest: any initial fragment of a trace leads to a single state in the behavior. Thus, the deterministic behavior itself can be seen as a specification of a (possibly infinite) set of traces.

System history Assume a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$. A *system history* is an alternating sequence of system configurations and actions of the form $h = (s_1^0, \ldots, s_n^0, e^0) \cdot [A_1^1, \ldots, A_n^1] \cdot (s_1^1, \ldots, s_n^1, e^1) \cdots (s_1^{\ell-1}, \ldots, s_n^{\ell-1}, e^{\ell-1}) \cdot [A_1^{\ell}, \ldots, A_n^{\ell}] \cdot (s_1^{\ell}, \ldots, s_n^{\ell}, e^{\ell})$, for some $\ell \geq 0$, such that:[3]

- $(s_i^0, e^0) \cdot A_i^1 \cdot (s_i^1, e^1) \cdots (s_i^{\ell-1}, e^{\ell-1}) \cdot A_i^{\ell} \cdot (s_i^{\ell}, e^{\ell}) \in \mathcal{H}_{\mathcal{B}_i}$, for all $i \in \{1, \ldots, n\}$, that is, the "projected" history relative to behavior $\mathcal{B}_i$ is in fact a behavior history for $\mathcal{B}_i$;
- at each step $k \in \{0, \ldots, \ell - 1\}$, we have that $(e^k, \bigcup_{i=1}^{n} A_i^{k+1}, e^{k+1}) \in \delta_{\mathcal{E}}$, that is, the environment can make a legal transition according to the set of actions executed in *all* behaviors.

The length of a history h, denoted as $|h|$, is the number of system configurations in h. The set $\mathcal{H}$ denotes the set of all system histories.

Behavior Controllers We study two different mechanisms for managing and controlling the available behaviors. The first one involves a central component, called the *centralized controller*, and extends the notion of "scheduler" from (De Giacomo & Sardina 2007). The centralized controller has the ability of activating-resuming zero, one, or more of the available behaviors by instructing each of them to execute some set of actions among those that are allowed in their current state (taking into account the environment). It also has the ability of keeping track (at runtime) of the current state of each available behavior. The second mechanism involves a decentralized team of *local behavior controllers*, one for each behavior. At any point in time, each behavior controller can activate, stop, and resume the behavior it is attached to as well as broadcast messages and access the whole set of broadcasted messages.

[3] In some cases it may be sensible to require that $\bigcap_{i=1}^{n} A_i^k = \emptyset$ for every $k \in \{1, \ldots, \ell\}$, especially when shared resource consumed by actions are of concern.

Centralized Controller

We first focus on the synthesis of a centralized controller. This is essentially an extension of the work in (De Giacomo & Sardina 2007) so as to allow for multiple (concurrent) actions at every step. In (De Giacomo & Sardina 2007), in contrast, only one action at the time was executed and the centralized controller was essentially a scheduler assigning such action to one of the available behaviors.

Specifically, we are interested in the following problem: given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, B_n, \mathcal{E})$ and a *deterministic* behavior, called the *target behavior* $\mathcal{B}_0$ over $\mathcal{E}$, *synthesize a centralized controller P such that the target behavior is realized by suitably assigning actions to execute to the available behaviors.*

Let us formally define our synthesis problem. Let the system be $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, where $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$ is the environment and $\mathcal{B}_i = (S_i, s_{i0}, G_i, \delta_i, F_i)$, with $i \in \{1, \ldots, n\}$, are the available behaviors. Let the target behavior be $\mathcal{B}_0 = (S_0, s_{00}, \delta_0, F_0)$. A *centralized controller* is a function $P : \mathcal{H} \times 2^{\mathcal{A}} \rightarrow (2^{\mathcal{A}})^n$ that, given a system history $h \in \mathcal{H}$ and a (non-empty) set of requested actions $A \subseteq \mathcal{A}$ to perform, returns the set of actions to be performed by each behavior such that the requested set of actions A is fully realized. Observe that some behaviors may not execute any action and thus remain still. It may also happen that several behaviors execute the same action. Variants can be easily defined where an action is executed by exactly one behavior. The results presented here would also hold for such variants.

One can define when a centralized controller realizes the target behavior—a solution to the problem—by extending the definition found in (De Giacomo & Sardina 2007) of when a *centralized controller P realizes a trace t*. We omit this definition for lack of space. Recall that since the target behavior is a deterministic transition system, its behavior is completely characterized by the set of its traces. Thus, a *centralized controller P realizes the target behavior $\mathcal{B}_0$* if it realizes all its traces (see (De Giacomo & Sardina 2007)).

The techniques proposed in (De Giacomo & Sardina 2007), based on a polynomial reduction of the problem to satisfiability of a Propositional Dynamic Logic formula (Harel, Kozen, & Tiuryn 2000), can be extended to deal with our new notion of centralized controllers.[4] As a consequence, we have the following result.

Theorem 1 *Checking the existence of a centralized controller that realizes a target behavior $\mathcal{B}_0$ relative to a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ is EXPTIME-complete.*

Observe that, in general, a centralized scheduler can have infinite states. However, the next theorem shows

[4] To avoid an exponential blowup, special care has to be put in encoding the problem into PDL satisfiability.

that if a centralized controller that realizes the target behavior does exist, then there exists one with a *finite* number of states. To ground the ideas,[5] we define a *finite (state) centralized controller* relative to a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ as a tuple $\hat{P} = (\Sigma, \sigma_0, nexts, nexta)$, where:

- Σ is the *finite* set of states of the controller;
- $\sigma_0 \in \Sigma$ is the single initial state of the controller;
- $nexta : \Sigma \times S_1 \times \ldots \times S_n \times E \times 2^{\mathcal{A}} \rightarrow (2^{\mathcal{A}})^n$ is the controller output, which instructs each available behavior to execute a given set of actions given its current state, the current states of the behaviors and that of the environment, and the set of requested actions;
- $nexts : \Sigma \times S_1 \times \ldots \times S_n \times E \times 2^{\mathcal{A}} \rightarrow \Sigma$ is the transition function of the controller which states what is the next state of $\hat{P}$ after having observed the state of the behaviors and that of the environment, and the requested actions.

It is possible to univocally define, by induction on the structure of histories, the *induced* centralized controller (of the general form above): executing the finite controller in the system amounts to execute its induced controller. For sake of simplicity, we shall blur the distinction between a finite controller and its induced one.

The following result holds for finite controllers, and can be shown by resorting to the finite model property of PDL:

Theorem 2 *If there exists a centralized controller that realizes a target behavior $\mathcal{B}_0$ relative to a system $(\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, then there exists one which is finite.*

As a matter of fact, one can give a tighter bound on the number of states required by the finite controller.

Theorem 3 *If there exists a finite centralized controller that realizes a target behavior $\mathcal{B}_0$ relative to a system $(\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, then there exists one with $|\Sigma| \leq |\mathcal{B}_0|$ and with a function nexts that is independent of the states of $\mathcal{B}_1, \ldots, \mathcal{B}_n$, i.e., $nexts : \Sigma \times E \times 2^{\mathcal{A}} \rightarrow \Sigma$.*

Observe that such bound on the number of states of the controller is tight. Indeed, it is easy to find cases in which there exists no controller with less states than the target behavior.

Finally, we observe that the PDL reduction technique mentioned above, as the one in (De Giacomo & Sardina 2007), can be used to actually generate a finite centralized controller. In fact, from a finite model of the PDL formula, one can easily extract the *finite* centralized controller by defining *nexta* and *nexts* on the basis of the truth-values of the propositions in the model. In addition, one can apply the construction devised for Theorem 3 so as to get a controller with a minimal (in the above sense) number of states.

Example 2 (Centralized controller) Figure 2 depicts the centralized controller which realizes the target behavior T from Figure 1(c). Functions *nexta* and *nexts* are represented together as edges labelled with pairs I/O, where $I = \langle s_1, s_2, e, A \rangle$ and $O = \langle A_1, A_2 \rangle$, with the following meaning: s_i is the current state of camera C_i ($i \in \{1, 2\}$); e is the environment current state; A is the set of requested actions to be performed; and A_i is the set of actions assigned to camera C_i ($i \in \{1, 2\}$).

[5] Other representations for the controller are also possible, tough none can be exponentially more succinct than the one adopted here.

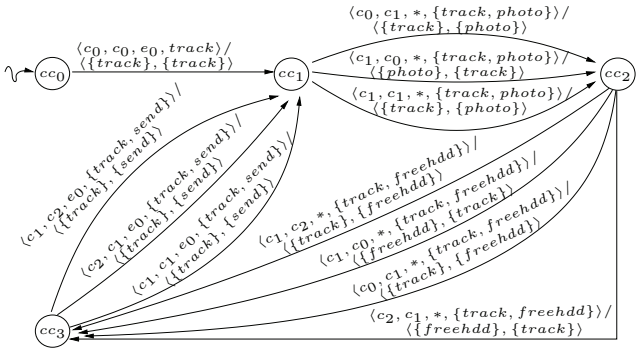


Figure 2: Centralized controller for the example scenario.

Distributed Controllers

We next turn to the case in which a centralized controller is not implementable and, as a result, we have to rely on distributed controllers. The main difficulty here is that such controllers have no access to the complete history, but only to the local history of the behavior they are controlling. To overcome that we allow distributed controllers to communicate through message broadcasting. This is essential to make them able to cooperate in order to realize the target behavior.

Messages We assume to have a set of possible messages $\mathcal{M}$ (more precisely, message types) that can be broadcasted. That is, we do not put a priori limits to the information that the distributed controllers can exchange. Later, though, we will see that a finite set of messages is sufficient.

Extended Local Behavior History We extend the notion of local histories to incorporate messages from $\mathcal{M}$. An *extended behavior history* $h^+_{\mathcal{B}}$ for a given a behavior $\mathcal{B} = (S, s_0, G, \delta_{\mathcal{B}}, F)$ over an environment $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$, is any finite sequence of the form $(s^0, e^0, M^0) \cdot A^1 \cdot (s^1, e^1, M^1) \cdots (s^{\ell-1}, e^{\ell-1}, M^{\ell-1}) \cdot A^\ell \cdot (s^\ell, e^\ell, M^\ell)$ such that the following constraints hold:

- $(s^0, e^0) \cdot A^1 \cdot (s^1, e^1) \cdots (s^{\ell-1}, e^{\ell-1}) \cdot A^\ell \cdot (s^\ell, e^\ell) \in \mathcal{H}_{\mathcal{B}}$, that is, we get a behavior history for $\mathcal{B}$ when the set of broadcasted messages are projected out;
- $M^0 = \emptyset$ and $M^k \subseteq \mathcal{M}$, for all $k \in \{0, \ldots, \ell\}$.

Observe that the behavior itself puts no constraints on the messages that are broadcasted at each step. However, we'll see that the local controller will. The set $\mathcal{H}^+_{\mathcal{B}}$ denotes the set of all *extended* local behavior histories for $\mathcal{B}$ (over $\mathcal{E}$).

Local Controllers A local controller is a module that can be (externally) attached to a behavior in order to control its operation. It has the ability of activating-resuming its controlled behavior by instructing it to execute a set of actions. Also, the controller has the ability of broadcasting messages after observing how the attached behavior evolved w.r.t. the delegated set of actions, and to access all messages broadcasted by the other local controllers at every step. Lastly, the controller has full observability on the environment.

Formally, a *local behavior controller* for behavior $\mathcal{B}$ is a pair of functions $\mathcal{C} = (P, B)$ of the following form:

$$P : \mathcal{H}^+_{\mathcal{B}} \times 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}; \qquad B : \mathcal{H}^+_{\mathcal{B}} \times 2^{\mathcal{A}} \times S \rightarrow 2^{\mathcal{M}}.$$

Function P states what actions $A' \subseteq A$ to delegate to the attached behavior at local extended behavior history $h^+_{\mathcal{B}}$ when actions A were requested. Function B states what messages, if any, are to be broadcasted under the same circumstances and the fact that the attached behavior has just moved to state s after executing actions A'. We attach one local controller $\mathcal{C}_i$ to each available behavior $\mathcal{B}_i$ in system $\mathcal{S}$.

In general, local controllers can have infinite states, though, as for central controllers, we will be particularly interested in finite state ones. A *finite (state) local behavior controller* for a behavior $\mathcal{B}$, and relative to environment $\mathcal{E}$ and a set of messages $\mathcal{M}$, is a tuple $\hat{\mathcal{C}} = (\Sigma, \hat{\mathcal{M}}, \sigma_0, \mathit{nexts}, \mathit{nexta}, \mathit{nextm})$, such that:

- Σ is the *finite* set of states of the controller;
- $\hat{\mathcal{M}} \subseteq \mathcal{M}$ is a *finite* set of messages;
- $\sigma_0 \in \Sigma$ is the single initial state of the controller;
- $\mathit{nexta} : \Sigma \times S \times E \times 2^{\hat{\mathcal{M}}} \times 2^{\mathcal{A}} \rightarrow 2^{\mathcal{A}}$ is the action output of the controller, which observes the state of the controlled behavior, the state of the environment, the messages broadcasted, the actions requested for execution, and delegates some of these to its controlled behavior;
- $\mathit{nextm} : \Sigma \times S \times E \times 2^{\hat{\mathcal{M}}} \times 2^{\mathcal{A}} \times S \rightarrow 2^{\hat{\mathcal{M}}}$ is the message output of the controller, which observes the state of the controlled behavior, the state of the environment, the messages broadcasted, the actions requested, and the state of the controlled behavior resulting from executing the selected subset of these actions, and states what messages, if any, are to be broadcasted by the controller;
- $\mathit{nexts} : \Sigma \times S \times E \times 2^{\hat{\mathcal{M}}} \times 2^{\mathcal{A}} \rightarrow \Sigma$ states what is the next state of $\hat{\mathcal{C}}$ after having observed the state of the controlled behavior, the state of the environment, the messages broadcasted, and the actions requested.

As with finite central controllers, one can univocally define the *induced* local controller (of the general form above) of a finite local controller such that running the finite local controller in the system amounts to execute its induced version. Once again, we blur the two notions in the following.

Extended System History We now extend system histories to include messages from $\mathcal{M}$. Assume then a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$. An *extended system history* is an alternating sequence of system configurations and actions of the form $h^+ = (s^0_1, \ldots, s^0_n, e^0, M^0) \cdot [A^1_1, \ldots, A^1_n] \cdot (s^1_1, \ldots, s^1_n, e^1, M^1) \cdots (s^{\ell-1}_1, \ldots, s^{\ell-1}_n, e^{\ell-1}, M^{\ell-1}) \cdot [A^\ell_1, \ldots, A^\ell_n] \cdot (s^\ell_1, \ldots, s^\ell_n, e^\ell, M^\ell)$ such that:

- $(s^0_1, \ldots, s^0_n, e^0) \cdot [A^1_1, \ldots, A^1_n] \cdot (s^1_1, \ldots, s^1_n, e^1) \cdots (s^{\ell-1}_1, \ldots, s^{\ell-1}_n, e^{\ell-1}) \cdot [A^\ell_1, \ldots, A^\ell_n] \cdot (s^\ell_1, \ldots, s^\ell_n, e^\ell) \in \mathcal{H}$, that is, we get a system history after projecting out all broadcasted messages M^i;
- $M^0 = \emptyset$ and $M^k \subseteq \mathcal{M}$, for all $k \in \{0, \ldots, \ell\}$.

Notice that messages are shared by the local history of each available behavior: they all see the same messages. The set $\mathcal{H}^+$ shall denote the set of all extended system histories. Also, if $h^+ \in \mathcal{H}^+$ is as above, then $h^+|_i$ denotes the corresponding extended local *projected history* w.r.t. behavior $\mathcal{B}_i$, that is, $(s^0_i, e^0, M^0) \cdot A^1_i \cdot (s^1_i, e^1, M^1) \cdot \cdots \cdot A^\ell_i \cdot (s^\ell_i, e^\ell, M^\ell)$.

Local Controller Synthesis A *distributed controller* is a set of local controllers, one for each available behavior. The problem we are interested in is the following: given a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, a set of messages $\mathcal{M}$, and a *deterministic* target behavior $\mathcal{B}_0$ over $\mathcal{E}$, *synthesize a distributed controller, i.e., a team of n local controllers, such that the target behavior is realized by concurrently running all behaviors under the control of their respective controllers*.

More precisely, let $\mathcal{M}$ be a set of messages, and let the system be $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, where $\mathcal{E} = (\mathcal{A}, E, e_0, \delta_{\mathcal{E}})$ and $\mathcal{B}_i = (S_i, s_{i0}, G_i, \delta_i, F_i)$, for $i \in \{1, \ldots, n\}$. Let the target behavior be $\mathcal{B}_0 = (S_0, s_{00}, \delta_0, F_0)$. Since the target behavior is a deterministic transition system, its behavior is fully characterized by the set of its traces, that is, by the set of infinite action sequences that are faithful to its transitions, and of finite sequences that in addition lead to a final state.

So, given a trace $t = (g^1, A^1) \cdot (g^2, A^2) \cdots$ of the target behavior, we say that *a distributed controller* $\mathcal{T} = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ *realizes the trace* t iff for all ℓ and for all extended system histories $h^\ell \in \mathcal{H}^\ell_{t,\mathcal{T}}$ ($\mathcal{H}^\ell_{t,\mathcal{T}}$ is defined below) such that $g^{\ell+1}(e^\ell) = \texttt{true}$ in the last environment state e^ℓ of h^ℓ, we have that $\mathsf{Ext}_{t,\mathcal{T}}(h^\ell, A^{\ell+1})$ is nonempty, where $\mathsf{Ext}_{t,\mathcal{T}}(h, A)$ is the set of $(|h|+1)$-length extended system histories of the form $h \cdot [A_1, \ldots, A_n] \cdot (s^{|h|+1}_1, \ldots, s^{|h|+1}_n, e^{|h|+1}, M^{|h|+1})$ such that:

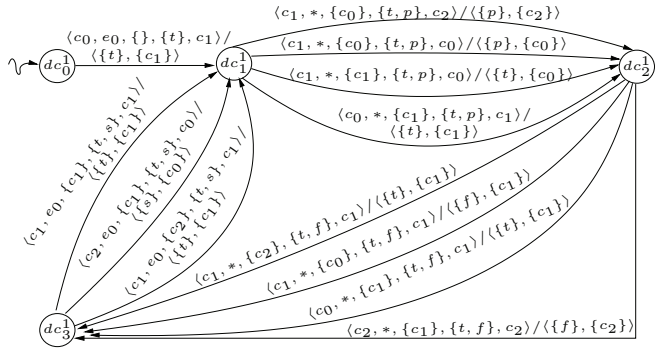
- $(s^{|h|}_1, \ldots, s^{|h|}_n, e^{|h|}, M^{|h|})$ is the last configuration in h;
- $A = \bigcup_{i=1}^n A_i$, that is, the requested set of actions A is fulfilled by putting together all the actions executed by every behavior;
- $P_i(h|_i, A) = A_i$ for all $i \in \{1, \ldots, n\}$, that is, the local controller $\mathcal{C}_i$ instructed behavior $\mathcal{B}_i$ to execute actions A_i;
- $(s^{|h|}_i, g, A_i, s^{|h|+1}_i) \in \delta_i$ with $g(e^{|h|}) = \texttt{true}$, that is, behavior $\mathcal{B}_i$ can evolve from its current state $s^{|h|}_i$ to state $s^{|h|+1}_i$ with respect to the (current) environment state $e^{|h|}$;
- $(e^{|h|}, A, e^{|h|+1}) \in \delta_{\mathcal{E}}$, that is, the environment can evolve from its current state $e^{|h|}$ to state $e^{|h|+1}$;
- $M^{|h|+1} = \bigcup_{i=1}^n B_i(h|_i, A, s^{|h|+1})$, that is, the set of broadcasted messages is the union of all messages broadcasted by each local controller.

The set $\mathcal{H}^k_{t,\mathcal{T}}$ of all histories that implement the first k actions of trace t and is prescribed by $\mathcal{T}$ is defined as follows:

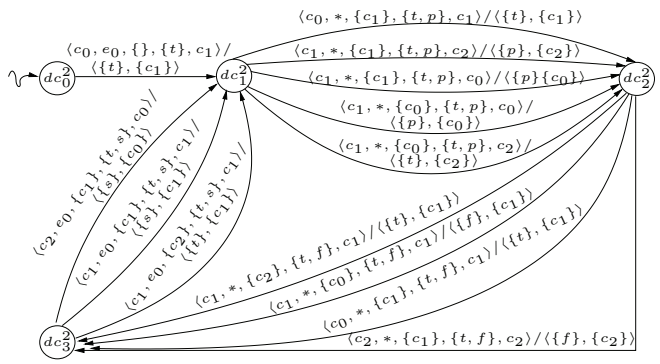
- $\mathcal{H}^0_{t,\mathcal{T}} = \{(s_{10}, \ldots, s_{n0}, e_0, \{\})\}$;
- $\mathcal{H}^{k+1}_{t,\mathcal{T}} = \bigcup_{h^k \in \mathcal{H}^k_{t,\mathcal{T}}} \mathsf{Ext}_{t,\mathcal{T}}(h^k, A^{k+1})$, for every $k \geq 0$;

In addition, as before, if the trace t is finite and ends after m actions, and all its guards are satisfied all along, we require that all histories in $\mathcal{H}^m_{t,\mathcal{T}}$ end with all behaviors in a *final state*. Finally, we say that a *distributed controller* $\mathcal{T} = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ *realizes the target behavior* $\mathcal{B}_0$ if it realizes all its traces (recall that $\mathcal{B}_0$ is deterministic).

In order to understand the above definitions, let us observe that, intuitively, the team of local controllers realizes a trace if, as long as the guards in the trace are satisfied, they can globally perform all actions prescribed by the trace—each of the local controllers instructs its behavior to do some of



(a) Behavior controller for camera C_1.



(b) Behavior controller for camera C_2.

Figure 3: Distributed controllers for the surveillance scenario. Notation t, f, s and p is used to abbreviate actions $track$, $freehdd$, $send$ and $photo$, respectively.

them. In order to do so, each local controller can use the history of its behavior together with the (global) messages that have been broadcasted so far. In some sense, implicitly through such messages, each local controller gets information on the global system history in order to take the right decision. Furthermore, at each step, each local controller broadcasts messages; such messages will be used in the next step by all behavior controllers to choose how to proceed.

Example 3 (Distributed controllers) Figure 3 depicts the distributed controllers for cameras C_1 and C_2. Functions *nexta*, *nextm*, and *nexts* are (compactly) represented as edges labelled with pairs I/O, where $I = \langle s, e, M_{rec}, A, s'\rangle$ and $O = \langle \bar{A}, M_{sent}\rangle$: s is the current state of the camera the distributed controller is attached to; e is the current state of the environment; M_{rec} is the set of all messages the distributed controller has received; A is the set of requested actions to be performed; $\bar{A}$ is the set of actions the distributed controller assigns to its attached camera; s' is the state of the camera after performing the assigned actions $\bar{A}$; and lastly, M_{sent} is the set of all messages broadcasted by the controller.

Distributed vs Centralized Controllers

We now investigate the relationship between central controllers and distributed ones. The main result on this relationship is that in going from centralized controllers to distributed ones we do not lose generality.

Theorem 4 *Let $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ be a system and let $\mathcal{B}_0$ be the target behavior. Then, there exists a distributed controller that realizes $\mathcal{B}_0$ iff there exists a centralized controller that realizes $\mathcal{B}_0$.*

The crux of the proof of this theorem (omitted here for space reasons) is that, through the suitable use of messages, one can emulate the ability of accessing the states of each of the available behaviors—such states are not directly observable in the distributed case. Interestingly, the proof shows that it is sufficient for the set of messages $\mathcal{M}$ to be finite.

As an immediate consequence of Theorem 4 and Theorem 1, we get a computational complexity characterization of the synthesis problem in the distributed case as well.

Theorem 5 *Checking the existence of a distributed controller that realizes a target behavior $\mathcal{B}_0$ relative to a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ is EXPTIME-complete.*

Next, we turn our attention to the relationship between *finite* central controllers and *finite* distributed controllers, i.e., teams of finite local controllers. The main result we get is the following.

Theorem 6 *If there exists a finite central controller P that realizes $\mathcal{B}_0$ relative to a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, then there is a finite distributed controller $\mathcal{T} = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ that realizes $\mathcal{B}_0$ relative to $\mathcal{S}$.*

PROOF. Assume $P = (\Sigma, \sigma_0, \mathit{nexts}, \mathit{nexta})$ realizes $\mathcal{B}_0$ relative to $\mathcal{S}$. We define the finite state local controller $\mathcal{C}_i = (\Sigma_i, \hat{\mathcal{M}}, p_{i0}, \mathit{nexts}_i, \mathit{nexta}_i, \mathit{nextm}_i)$ as follows:

- $\Sigma_i = \Sigma$;
- $\hat{\mathcal{M}} = \{\text{“}j : s\text{”} \mid s \in S_j,\ j = \{1, \ldots, n\}\}$;
- $p_{i0} = \sigma_0$;
- $\mathit{nexta}_i(\sigma, s_i, e, \{\text{“}1 : s_1\text{”}, \ldots, \text{“}n : s_n\text{”}\}, A) = A_i$ iff $\mathit{nexta}(\sigma, s_1, \ldots, s_n, e, A) = [A_1, \ldots, A_i, \ldots, A_n]$;
- $\mathit{nextm}_i(\sigma, s, e, M, A, s') = \text{“}i : s'\text{”}$;
- $\mathit{nexts}_i(\sigma, s_i, e, \{\text{“}1 : s_1\text{”}, \ldots, \text{“}n : s_n\text{”}\}, A) = \mathit{nexts}(\sigma, s_1, \ldots, s_n, e, A)$. □

From the above theorem, we get the analog of Theorem 2 for distributed local controllers.

Theorem 7 *If there exists a distributed controller $\mathcal{T}$ realizing a target behavior $\mathcal{B}_0$ relative to a system $(\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$ and a set of messages $\mathcal{M}$, then there exists one that is finite.*

As a matter of fact, the construction in the proof of Theorem 6 gives us a way of obtaining a distributed controller: generate a finite centralized controller realizing the target behavior, and then apply the construction above to get the team of finite local controllers. Interestingly, if we start from a minimal finite centralized controller satisfying the size condition in Theorem 3, then what we get is a team of “optimal” finite local controllers. Indeed, we get the following: *(i)* the number of states of the local controllers is $|\Sigma_i| \leq |\mathcal{B}_0|$; *(ii)* the total number of different messages in the solution is bounded by the size of the available behaviors, i.e., $|\hat{\mathcal{M}}| \leq |\mathcal{B}_1| + \cdots + |\mathcal{B}_n|$; and *(iii)* the size of each message is bounded by $\log n \cdot \log |\mathcal{B}_i|$, and at each point there are at most n broadcasted messages of such size—this gives us bounds on the channel required for the messages exchange. We observe that all these bounds are tight and one cannot do better than this in general. It is easy to find cases in which

we do need $|\mathcal{B}_0|$ number of states, $|\mathcal{B}_1| + \cdots + |\mathcal{B}_n|$ messages, and n broadcasted messages of size $\log n \cdot \log |\mathcal{B}_i|$, or otherwise we lose the ability of generating a distributed controller, even when a centralized one does exist.

We now look at the reverse relationship, that is, how we can get a central controller from a distributed one.

Theorem 8 *If there exists a finite distributed controller $\mathcal{T} = (\mathcal{C}_1, \ldots, \mathcal{C}_n)$ that realizes the target behavior $\mathcal{B}_0$ relative to a system $\mathcal{S} = (\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{E})$, then there is a finite central controller P that realizes $\mathcal{B}_0$ relative to $\mathcal{S}$.*

PROOF. Let $\mathcal{C}_i = (\Sigma_i, \hat{\mathcal{M}}, \sigma_{i0}, \mathit{nexts}_i, \mathit{nexta}_i, \mathit{nextm}_i)$, for $i \in \{1, \ldots, n\}$. We define the finite state central controller $P = (\Sigma, \sigma_0, \mathit{nexts}, \mathit{nexta})$ relative to system $\mathcal{S}$ as follows:

- $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$;
- $\sigma_0 = \langle p_{10}, \ldots, p_{n0} \rangle$;
- $\mathit{nexta}(\langle \sigma_1, \ldots, \sigma_n \rangle, s_1, \ldots, s_n, e, A) = [\mathit{nexta}_1(\sigma_1, s_1, e, M, A), \ldots, \mathit{nexta}_n(\sigma_n, s_n, e, M, A)]$;
- $\mathit{nexts}(\langle \sigma_1, \ldots, \sigma_n \rangle, s_1, \ldots, s_n, e, A) = \langle \mathit{nexts}_1(\sigma_1, s_1, e, M, A), \ldots, \mathit{nexts}_n(\sigma_n, s_n, e, M, A) \rangle$. □

Looking at the construction used in the theorem above, we get other interesting bounds: while the central controller obtained from the local controllers is, in general, of the size of the Cartesian product of the states of the local controllers, in the case where the local controllers are all *minimal* (i.e., their states and state transitions correspond to those of the target behavior), the obtained central controller can be shrunk to the same number of states (of the target) by projecting out states that are not reachable from the initial state σ_0. Furthermore, if one starts from a *minimal* central controller (as in Theorem 3), generates the corresponding team of local controllers using the construction for Theorem 6, and finally generates back a central controller using the construction of Theorem 8, one gets back the original central controller.

Conclusion

In this paper, we investigated the distributed version of the synthesis problem introduced in (De Giacomo & Sardina 2007). As a result, the solutions proposed here are palatable to a much wider range of cases, including those where we have multiple independent agents and a centralized solution, as that in (De Giacomo & Sardina 2007), is not conceivable.

We observe that the kind of problems we dealt with are special forms of reactive process synthesis, both for the centralized (Pnueli & Rosner 1989) and distributed (Pnueli & Rosner 1990) cases. The main distinction is, apart from the specific formal settings, the kind of design specification to be realized (a target behavior in our case). It is well known that, in general, distributed solutions are much harder to get than centralized ones (Pnueli & Rosner 1990; Kupferman & Vardi 2001). This is not the case in our approach since we allow for equipping local controllers with autonomous message exchange capabilities, even if such capabilities are not present in the behaviors that they control. Such a capability is at the core of the reason why one can always get a distributed controller from a centralized one.

We close the paper by pointing out that the context in which the distributed synthesis problem has been tackled in this paper can be seen as "ideal": (i) there exists a shared and fully reliable messaging channel with no a priori size bound; (ii) the environment is shared and fully observable by every local behavior controller; and (iii) behaviors are allowed to synchronize at every step just before the next action to be performed. In several practical cases, however, it would be necessary to address variations of this ideal context. For instance, in distributed settings where behaviors are geographically far apart, one should consider behavior controllers with local (partial) observability of the environment or even consider different environments all together. Also, when communication is limited and unreliable, robust solutions in which a behavior can be "replaced" upon failure are desired. Finally, in cases with diverse devices acting at very different rates, more asynchronous accounts are of interest. These and other variations remain for future study.

Acknowledgments

This work was funded by the European FET basic research project FP6-7603 Thinking Ontologies (TONES) and the Australian Research Council and Agent Oriented Software under the grant LP0560702. The authors would also like to thank the anonymous reviewers for their comments.

References

Berardi, D.; Calvanese, D.; De Giacomo, G.; Hull, R.; and Mecella, M. 2005. Automatic composition of transition-based semantic web services with messaging. In *Proc. of VLDB-05*, 613–624.

Bredenfeld, A.; Jacoff, A.; Noda, I.; and Takahashi, Y., eds. 2006. *RoboCup '05: Robot Soccer WC IX*, volume 4020 of *LNCS*.

De Giacomo, G., and Sardina, S. 2007. Automatic synthesis of new behaviors from a library of available behaviors. In *Proc. of IJCAI-07*, 1866–1871.

Gelfond, M., and Lifschitz, V. 1998. Action languages. *Electronic Transactions of AI (ETAI)* 2:193–210.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive reasoning and planning. In *Proc. of AAAI-87*, 677–682.

Harel, D.; Kozen, D.; and Tiuryn, J. 2000. *Dynamic Logic*. The MIT Press.

Katz, M. J., and Rosenschein, J. S. 1993. The generation and execution of plans for multiple agents. *Computers and Artificial Intelligence* 12(1):5–35.

Kupferman, O., and Vardi, M. Y. 2001. Synthesizing distributed systems. In *Proc. of LICS-01*, 389–398.

McIlraith, S., and Son, T. C. 2002. Adapting Golog for programming the semantic web. In *Proc. of KR*, 482–493.

Meuleau, N.; Peshkin, L.; Kim, K.-E.; and Kaelbling, L. P. 1999. Learning finite-state controllers for partially observable environments. In *Proc. of UAI-99*, 427–436.

Pnueli, A., and Rosner, R. 1989. On the synthesis of a reactive module. In *Proc. of POPL-89*, 179–190.

Pnueli, A., and Rosner, R. 1990. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS-90*, 746–757.

Saffiotti, A., and Broxvall, M. 2005. PEIS ecologies: Ambient intelligence meets autonomous robotics. In *Proc. of the Int. Conf. on Smart Objects and Amb. Intell.*, 275–280.

Tilden, M. W. 1993. The evolution of functional roboecologies. *ARS Electronica* 93:195–200.