Safe Run-time Adaptation of Distributed Applications

A Choreography-driven Approach

Mila Dalla Preda[1], Ivan Lanese[2], Jacopo Mauro[2], Maurizio Gabbrielli[2], and Saverio Giallorenzo[2]

[1] Dipartimento di Informatica - Univ. of Verona
[2] Dipartimento di Informatica - Univ. of Bologna / INRIA

Abstract. Proving that distributed applications are well-behaved, e.g. are deadlock free, is a complex problem. The problem becomes even harder if those applications dynamically adapt to face needs which were unexpected when the application was deployed or even started. We present a framework where the adaptive application is specified by a global description, called an Adaptive Interaction-Oriented Choreography (AIOC), which is deadlock free by construction. Adaptation is enacted by rules, defined and applied at run-time, which replace a region of the AIOC, involving potentially multiple participants, with a new AIOC fragment. We automatically derive from the AIOC a distributed prototype implementing it. We prove that, under static and efficiently verifiable syntactic conditions on the AIOC and on the adaptation rules, the distributed prototype is compliant with the behavior specified by the AIOC, for all the possible, dynamically changing, environment conditions and sets of adaptation rules.

1 Introduction

Nowadays, applications are distributed, and composed by different entities which engage in complex protocols to reach a common goal. For instance, in the Price Inquiry Scenario of an online purchase, a buyer and a seller exchange messages according to a given pattern. We can describe the code for `buyer` and `seller` with a *Process-Oriented Choregraphy* (POC):

Buyer Description
```
prod = getInput();
priceReq : prod to seller;
offer : price from seller
```

Seller Description
```
priceReq : prods from buyer;
prices = priceB(prods);
offer : prices to buyer
```

The `buyer` locally executes the input (using function `getInput`) of the name of a product (s)he is interested in into local variable `prod`. Then (s)he sends a price request to the `seller` on channel `priceReq`, with the product name as a parameter. Finally, (s)he waits on channel `offer` for a price, and stores it in local variable `price`. Symmetrically, the `seller` waits for a price request from the `buyer`, computes the price `prices` of the product using function `priceB`, and answers by sending the computed price to the `buyer`.

In such a simple scenario, with only two participants and a unique possible message sequence, one can check that the application is well-behaved. In particular, each send has a matching receive, and the application is deadlock free. Note that one would obtain

a deadlock by, e.g., swapping the send and the receive inside the `buyer`. Ensuring well-behavedness becomes much more difficult for a complex application, even more if it involves more than two participants.

Now, assume that, while the application is running, the seller direction decides to stimulate business by defining new business rules. For instance, a fidelity summer card may be distributed to buyers, allowing them to get a 10% discount on their summer purchases. Such changes in business rules are standard in current economics, nevertheless updating the application to implement such a simple change raises many problems:

- which adapted code should be executed by each participant so to implement the required business rule?
- how to integrate the adapted code into the current one?
- when to activate the adapted code?
- how to ensure that the changes in the code of the two participants are coordinated?
- assuming the old application was well-behaved, how to ensure that the adapted one is well-behaved too?

Note that the good behavior of the application depends on the interactions between the two participants: it may well be the case that if a participant is updated and the other one is not, the resulting application will deadlock.

Answering the questions above at the level of POC is difficult.

We will answer them by exploiting a more abstract description of the application, called an *Interaction-Oriented Choreography* (IOC). We will then show that the POC description of the application can be automatically derived from the IOC via a *behavior-preserving projection* operation. This approach to software development is called *choreography-driven design*, and has been successfully applied to ensure the good behavior of distributed applications [5, 6, 22]. The main aim of this paper is extending choreography-driven design to cope with *unexpected adaptation needs* such as the one above, while preserving its main property, namely that the behavior of the projected POC coincides with the behavior of the IOC. To emphasize that our descriptions include adaptation we speak about *adaptive IOC* (AIOC) and *adaptive POC* (APOC).

An IOC description of the Price Inquiry Scenario above is:

```
prod@buyer = getInput();
priceReq : buyer( prod ) → seller( prods );
prices@seller = priceB(prods);
offer : seller( prices ) → buyer( price )
```

Here, the `buyer` reads from the user the product name `prod`. Then, the `buyer` engages in a communication with the `seller`: (s)he sends the name of the product to the `seller`, which stores it in a local variable `prods`. The `seller` computes the price of the product, and sends it to the `buyer`, that stores it in a local variable `price`. At the IOC level the expected development of the interaction is much more clear than at the POC level, since communications are syntactically explicit. However, the IOC is less suitable to generate or check the code of single participants, to ensure that they indeed follow the expected protocol. For these reasons, having a behavior-preserving projection to derive the POC from the IOC is a key ingredient of the approach.

Integration of adaptation into choreography-driven design requires to introduce adaptation mechanisms both at the level of AIOC and of APOC, and to extend the projection to cope with the new mechanisms. Mainly, one has to ensure that the projection is behavior-preserving.

Adaptive Price Inquiry Scenario

```
prod@buyer = getInput();
priceReq : buyer( prod ) → seller( prods );
scope price-inquiry@seller {
 prices@seller = priceB(prods);
 offer : seller( prices) → buyer( price )
} prop { N.offers = 0}
```

Summer Card Adaptation Rule

```
rule price-inquiry
where 06.21.2013 < E.date and
 E.date < 09.21.2013 and N.offers == 0
specifies
 cardReq : seller( void ) → buyer( _ );
 cardId@buyer = getInput();
 card : buyer( cardId ) → seller( buyerId );
 if valid( buyerId )@seller {
   prices@seller = priceB(prods) * 0.9
 } else { prices@seller = priceB(prods) };
 offer : seller( prices ) → buyer( price )
```

Fig. 1. Adaptive Price Inquiry Scenario.

A main issue here is to find mechanisms to specify adaptation able to cope with adaptation needs which were not expected when the application has been deployed or even started. This is fundamental since it is not possible to foresee in advance all the possible adaptation needs, even less the possible solutions to them. For instance, in the Price Inquiry Scenario, it is not possible to know in advance all the offer modalities and conditions the sales direction may want to propose.

To deal with unexpected updates, we present two mechanisms: (i) a mechanism of scope, used inside AIOCs and APOCs to specify which code may be adapted, and, (ii) a set of rules specifying when the code should be adapted and incorporating the new code. The approach is able to cope with unforeseen adaptation needs since the set of adaptation rules is dynamic, i.e. can be changed at any moment to face a new adaptation need. Ensuring that these unpredictable changes of the set of rules do not break the approach, and the fact that the projection is behavior preserving, is the main technical challenge we had to face.

Fig. 1 shows an AIOC making adaptable the Price Inquiry Scenario (on the left), and an adaptation rule enabling the Summer Card update (on the right). The AIOC includes a scope with label `price-inquiry` which specifies that the communication between the `seller` and the `buyer` may be adapted. The Summer Card rule is applicable to such a scope since it matches the scope label. In order to check whether adaptation will improve the distributed application, one has to check whether the applicability condition of the rule, defined by clause **`where`**, is satisfied. This condition may take into account the state of the application, the non functional properties of the scope, and the environment conditions. In the example, the non-functional property `N.offers = 0` indicates that no offer is currently active. Similarly, the environment variable `E.date` contains the current date. The annotation `@seller` after the name of the scope specifies that the `seller` is in charge of acting as a coordinator for the adaptation step.

The body of the rule specifies the protocol to be followed when the summer card offer is active. First, the `seller` asks the card id to the `buyer`. The `buyer` inputs the id, stores it into the variable `cardId` and sends this information to the `seller`. If the summer card id is valid then the discount is applied, otherwise the standard price is computed.

A graphical view of an adaptation step is in Fig. 2. The upper-left corner of the diagram shows an AIOC, where a scope with label ℓ, represented as a black box, is available.

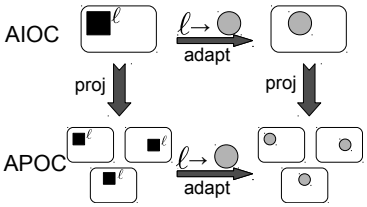

Fig. 2. Our approach, graphically.

From the AIOC, one can automatically derive the APOC in the lower-left corner using the behavior-preserving projection. In the APOC, three different participants feature a scope with label ℓ: their interactions implement the behavior specified by the AIOC scope. Assume to have an adaptation rule applicable to scopes with label ℓ, specifying that, at the AIOC level, the scope with label ℓ becomes a gray circle. We want to apply the same rule to the running APOC so that the result of:

1. adapting the AIOC and then derive the adapted APOC by projection, or
2. directly adapt the APOC

is the same. In other words, the diagram commutes. The same result holds also for normal, non adaptation, steps. As a consequence, the behavior of the APOC, under all possible sets of adaptation rules, proceed as prescribed by the abstract adaptive model given by the AIOC. As a corollary, the APOC is well-behaved.

Summarizing, this paper provides the following contributions:

- the definition of an abstract AIOC language to specify adaptive distributed applications;
- the definition and implementation of a more concrete APOC language;
- the definition of a behavior-preserving projection function to automatically derive an APOC from an AIOC;
- the proof that an APOC derived from an AIOC is compliant with the AIOC under all possible sets of adaptation rules and all possible execution environments.

As a result we ensure that adaptive distributed applications derived from AIOCs are correct by design.

Structure of the paper: Section 2 and Section 3 present the syntax and semantics of AIOCs and APOCs, respectively. Section 4 presents the correctness result. Section 5 describes AIOCJ, our prototype implementation. The paper ends with discussion and related works in Section 6. Full proofs of the results are collected in Appendix, and additional material is available in the companion technical report [10].

2 Adaptive Interaction-Oriented Choreography (AIOC)

This section defines the architectural model of the distributed adaptive applications we consider, and the syntax and semantics of the AIOC language specifying them.

We consider applications composed by participants deployed on different localities, each executing its own code and accessing its own local state, and interacting via synchronous message passing.[1] Adaptation is performed by an *adaptation middleware* that includes one or more, possibly distributed, *adaptation servers*, which are repositories of

[1] Asynchronous message passing can also be considered, but this makes the technicalities more difficult without changing the main features of the approach.

adaptation rules. The running application may interact with the adaptation middleware to look for applicable adaptation rules. The effect of an adaptation rule is to replace a scope with a new AIOC answering the adaptation need. Applicability may depend on the execution environment (including, possibly, user desires), on the application state, and on the non-functional properties of the scope.

The main novelty of our languages is thus to provide scopes and adaptation rules, thus enabling adaptation.

Our languages rely on a set $Roles$, ranged over by $r, s, \ldots$, to identify the participants (also called roles, or localities) in the choreography. Roles exchange messages over channels, also called *operations*. We distinguish two kinds of operations: *public operations*, ranged over by o, and *private operations*, ranged over by o^*. We use $o^?$ to range over both public and private operations. Public operations represent relevant communications inside the application, and we ensure that both the AIOC and the corresponding APOC perform the same public operations, in the same order. Vice versa, private operations are just used for synchronization purposes, and additional private communications may be introduced when moving from the AIOC level to the APOC level. We also assume a set of scope labels, ranged over by $l, l_i, \ldots$, allowing adaptation rules to specify the scopes they can target. We denote with $Expr$ the set of expressions, ranged over by e. We deliberately do not give a formal definition of expressions, and of their typing, since our results do not depend on it. We only require that expressions include at least values, belonging to a set Val ranged over by $v, \ldots$, and variables, belonging to a set Var ranged over by $x, y, \ldots$, and that it is always possible to evaluate a given expression (e.g., a default value is given as a result if evaluation is not possible or causes an error). Finally, we assume a subset of expressions containing boolean expressions, ranged over by b, whose evaluation always yields either `true` or `false`.

The syntax of *AIOC processes*, ranged over by $\mathcal{I}, \mathcal{I}', \ldots$, is defined as follows:

$$\begin{array}{l}\mathcal{I} ::= o^? : r_1(e) \to r_2(x) \mid \mathcal{I};\mathcal{I}' \mid \mathcal{I}|\mathcal{I}' \mid x@r = e \mid \mathbf{1} \mid \mathbf{0} \mid \\ \qquad \mathit{if}\ b@r\ \{\mathcal{I}\}\ \mathit{else}\ \{\mathcal{I}'\} \mid \mathit{while}\ b@r\ \{\mathcal{I}\} \mid \mathit{scope}\ l@r\ \{\mathcal{I}\}\ \mathit{prop}\ \{\Delta\}\end{array}$$

Interaction $o^? : r_1(e) \to r_2(x)$ means that role r_1 sends a message on operation $o^?$ to role r_2 (we assume $r_1 \neq r_2$). The sent value is obtained by evaluating e in the local state of r_1 and it is then stored in variable x in r_2. Processes $\mathcal{I};\mathcal{I}'$ and $\mathcal{I}|\mathcal{I}'$ denote sequential and parallel composition of $\mathcal{I}$ and $\mathcal{I}'$, respectively. Assignment $x@r = e$ evaluates expression e in the local state of r and assigns the resulting value to its local variable x. Conditional $\mathit{if}\ b@r\ \{\mathcal{I}\}\ \mathit{else}\ \{\mathcal{I}'\}$ and iteration $\mathit{while}\ b@r\ \{\mathcal{I}\}$ are guarded by the evaluation of boolean expression b in the local state of r. The empty process $\mathbf{1}$ defines an AIOC that can only terminate. $\mathbf{0}$ represents a deadlocked AIOC. This is needed for the definition of the operational semantics, but we expect the programmer not to use it when specifying AIOC processes. In particular, we call *initial* an AIOC process where $\mathbf{0}$ never occurs. The construct $\mathit{scope}\ l@r\ \{\mathcal{I}\}\ \mathit{prop}\ \{\Delta\}$ delimits a region $\mathcal{I}$ of the AIOC process that may be adapted in the future.

Indeed, the designer needs to foresee whether a code region would need adaptation, to allow it to expose an interface towards the adaptation middleware, thus enabling adaptation. On the other hand, the designer does not need to know in advance under which conditions adaptation should be performed, nor the code that will answer the forthcoming adaptation need. While there are no general rules defining when a code

```
1  priceOk@buyer = False; continue@buyer = True;
2  while ( not(priceOk) and continue )@buyer {
3   prod@buyer = getInput(); priceReq : buyer( prod ) → seller( prods );
4   scope price-inquiry@seller {
5     prices@seller = priceB(prods); offer : seller( prices ) → buyer( price )
6   } prop { N.offers = 0};
7   priceOk@buyer = getInput();
8   if ( not(priceOk) )@buyer {continue@buyer = getInput()} };
9  if ( priceOk )@buyer {
10  payReq : seller( payDesc(prices) ) → bank( desc );
11  scope payment@bank {
12    paymentOK@bank = True; pay : buyer( payAuth(price) ) → bank( auth );
13    ... // code for the payment
14  } prop { N.securityLevel = 1 };
15  if ( paymentOK )@bank {
16    confirm : bank( void ) → seller(_) | confirm : bank( void ) → buyer(_)
17  } else { abort : bank( void ) → buyer(_) } }
```

Fig. 3. AIOC for Buying Scenario.

region should be adaptable, a few typical cases are described below. An AIOC region dealing with business rules should normally be adaptable, since business rules frequently change. Also, an AIOC region critical for performance or security reasons should be adaptable, to enable future improvements of performances or security properties. In *scope* $l@r$ $\{\mathcal{I}\}$ *prop* $\{\Delta\}$, role r coordinates the adaptation procedure by interacting with the adaptation middleware to check whether adaptation is needed, and with the other roles inside the scope to enact the adaptation procedure. Also, l is the label of the scope, to be matched by a corresponding label in the adaptation rule.[2] Function $\Delta : DName \rightarrow Val$ describes the *non-functional properties* of the current code, such as version number or latency. E.g., $\Delta(VersionNumber) = 3.5$ means that version 3.5 is currently installed. These properties can be checked when applying an adaptation rule, to avoid applying obsolete rules. Assigning correct non-functional properties to the scopes is a responsibility of the designer.

Fig. 3 gives an example of AIOC process, extending the one discussed in the Introduction to a more realistic setting: a `buyer` orders a product from a `seller`, paying via a `bank`. The `buyer` starts the protocol by iteratively repeating the Price Inquiry Scenario from the Introduction, until either the `seller` offer is fine (`priceOk`), or the `buyer` is not interested any more in buying the product (not `continue`). These decisions are taken by interacting with the user at `buyer` via function **`getInput`** (abstracting away all the details related to user interaction). If the `seller` offer is fine, the `seller` sends to the `bank` the payment details. The `buyer` then authorizes the payment via operation `pay`. We omit the details related to the local execution of the payment at the `bank`. Since the payment may be critical for security purposes, the related communication is enclosed in a scope `payment`, thus allowing, later on, the introduction of a more refined protocol. The scope has a non-functional property `N.securityLevel` with value 1, with the intended mean-

[2] More sophisticated ways to match scopes and rules can be studied, but this is orthogonal to the main issues tackled in the paper, thus we leave this point for future work.

ing that the scope only provides basic security. After the scope successfully terminates, the protocol ends with the `bank` acknowledging the payment to the `seller` and `buyer` in parallel. If the payment was not successful the failure is notified to the `buyer` only.

Adaptation is specified by adaptation rules, defined below.

Definition 1 (Adaptation rules). *An adaptation rule is a term* rule l where $\mathcal{C}$ specifies $\mathcal{I}$*, where l is the label of the scopes to which the rule applies, $\mathcal{C}$ is a boolean predicate that specifies the applicability condition to be satisfied, and $\mathcal{I}$ is the AIOC process that will replace the scope in case the adaptation is performed.*

The predicate $\mathcal{C}$ may refer to the *execution environment* E of the application, to the non-functional properties Δ of the considered scope, and to the *local state* Σ_r of the role r coordinating the adaptation. An environment E is a function from environment names to values: $E : EName \rightarrow Val$, while a local state Σ_r is a function from variables to values: $\Sigma_r : Var \rightarrow Val$. To avoid confusion, the sets Var, $EName$, and the set of non-functional property names $DName$ are mutually disjoint. As for expressions, we do not explicitly define the syntax of predicates in $\mathcal{C}$, since our results do not rely on it. We only require the existence of a decidable predicate $\Sigma_r, E, \Delta \vdash \mathcal{C}$ to establish whether the applicability condition $\mathcal{C}$ is satisfied by a local state Σ_r, an environment E and a set of non-functional properties Δ. We denote as $\mathbf{R}$ a set of adaptation rules. We require that adaptation rules (as well as AIOCs) satisfy a well-formedness condition called *connectedness*, defined in Section 4.

An example of adaptation rule is in the Introduction, and additional examples can be found in the companion technical report [10] or on the web site [1].

AIOC processes do not execute in isolation: they are equipped with a *global state* Σ, an environment E and a set of adaptation rules $\mathbf{R}$ (at this level of abstraction, we do not need to consider if they are distributed among different adaptation servers or not).

Definition 2 (AIOC systems). *An AIOC system is a quadruple $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$, denoting an AIOC process $\mathcal{I}$ equipped with a global state Σ, an environment E and a set of adaptation rules* $\mathbf{R}$*.*

A global state Σ is a map that defines the value v of each variable x in a given role r, namely $\Sigma : Roles \times Var \rightarrow Val$. The local state of role r verifies $\forall x \in Var : \Sigma(r, x) = \Sigma_r(x)$. Expressions are always evaluated by a given role r, i.e. considering local state Σ_r. We denote the evaluation of expression e in local state Σ_r as $[\![e]\!]_{\Sigma_r}$. We assume $[\![e]\!]_{\Sigma_r}$ is always defined, and that for each boolean expression b, $[\![b]\!]_{\Sigma_r}$ is either `true` or `false`. We assume a function $\mathrm{roles}(\mathcal{I})$ that computes the roles of an AIOC process $\mathcal{I}$. We can now define the semantics of AIOC systems.

Definition 3 (AIOC systems semantics). *The semantics of AIOC systems is defined as the smallest labeled transition system (LTS) closed under the rules in Table 1, where symmetric rules for parallel composition have been omitted.*

The rules in the top part of the table model the evolution of AIOC processes, while the rules in the bottom part lift the transitions to AIOC systems. When computing, AIOC processes may make assumptions on the state Σ, the environment E and the set of rules $\mathbf{R}$ (to check whether a rule is applicable), or on the state Σ only (to evaluate an expression). Since this information is not available at the process level, the assumptions (ranged over by A) are stored in the label and checked at the system level.

AIOC processes

(INTERACTION)
$$\frac{[\![e]\!]_{\Sigma_{r_1}} = v}{o^? : r_1(e) \to r_2(x) \xrightarrow{\Sigma, o^? : r_1(v) \to r_2(x)} x@r_2 = v}$$

(ASSIGN)
$$\frac{[\![e]\!]_{\Sigma_r} = v}{x@r = e \xrightarrow{\Sigma, [v/x, r]} \mathbf{1}}$$

(END)
$$\mathbf{1} \xrightarrow{\surd} \mathbf{0}$$

(SEQUENCE)
$$\frac{\mathcal{I} \xrightarrow{\mu} \mathcal{I}' \quad \mu \neq \surd}{\mathcal{I}; \mathcal{J} \xrightarrow{\mu} \mathcal{I}'; \mathcal{J}}$$

(SEQ-END)
$$\frac{\mathcal{I} \xrightarrow{\surd} \mathcal{I}' \quad \mathcal{J} \xrightarrow{\mu} \mathcal{J}'}{\mathcal{I}; \mathcal{J} \xrightarrow{\mu} \mathcal{J}'}$$

(PARALLEL)
$$\frac{\mathcal{I} \xrightarrow{\mu} \mathcal{I}' \quad \mu \neq \surd}{\mathcal{I} \parallel \mathcal{J} \xrightarrow{\mu} \mathcal{I}' \parallel \mathcal{J}}$$

(PAR-END)
$$\frac{\mathcal{I} \xrightarrow{\surd} \mathcal{I}' \quad \mathcal{J} \xrightarrow{\surd} \mathcal{J}'}{\mathcal{I} \parallel \mathcal{J} \xrightarrow{\surd} \mathcal{I}' \parallel \mathcal{J}'}$$

(IF-THEN)
$$\frac{[\![b]\!]_{\Sigma_r} = \texttt{true}}{\mathit{if}\ b@r\ \{\mathcal{I}\}\ \mathit{else}\ \{\mathcal{I}'\} \xrightarrow{\Sigma, \tau} \mathcal{I}}$$

(IF-ELSE)
$$\frac{[\![b]\!]_{\Sigma_r} = \texttt{false}}{\mathit{if}\ b@r\ \{\mathcal{I}\}\ \mathit{else}\ \{\mathcal{I}'\} \xrightarrow{\Sigma, \tau} \mathcal{I}'}$$

(WHILE-UNFOLD)
$$\frac{[\![b]\!]_{\Sigma_r} = \texttt{true}}{\mathit{while}\ b@r\ \{\mathcal{I}\} \xrightarrow{\Sigma, \tau} \mathcal{I}; \mathit{while}\ b@r\ \{\mathcal{I}\}}$$

(WHILE-EXIT)
$$\frac{[\![b]\!]_{\Sigma_r} = \texttt{false}}{\mathit{while}\ b@r\ \{\mathcal{I}\} \xrightarrow{\Sigma, \tau} \mathbf{1}}$$

(ADAPT)
$$\frac{\mathit{rule}\ l\ \mathit{where}\ \mathcal{C}\ \mathit{specifies}\ \mathcal{I}' \in \mathbf{R} \quad \Sigma_r, E, \Delta \vdash \mathcal{C} \wedge \mathrm{roles}(\mathcal{I}') \subseteq \mathrm{roles}(\mathcal{I})}{\mathit{scope}\ l@r\ \{\mathcal{I}\}\ \mathit{prop}\ \{\Delta\} \xrightarrow{\Sigma, E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}'} \mathcal{I}'}$$

(NOADAPT)
$$\frac{\forall \mathit{rule}\ l\ \mathit{where}\ \mathcal{C}\ \mathit{specifies}\ \mathcal{I}' \in \mathbf{R}\ .\ \Sigma_r, E, \Delta \nvdash \mathcal{C} \vee \mathrm{roles}(\mathcal{I}') \not\subseteq \mathrm{roles}(\mathcal{I})}{\mathit{scope}\ l@r\ \{\mathcal{I}\}\ \mathit{prop}\ \{\Delta\} \xrightarrow{\Sigma, E, \mathbf{R}, [\texttt{no-adapt}]} \mathcal{I}}$$

AIOC systems

(EXEC)
$$\frac{\mathcal{I} \xrightarrow{A, \alpha} \mathcal{I}' \quad \alpha \neq [v/x, r] \quad A = \Sigma \vee A = \Sigma, E, \mathbf{R}}{\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\alpha} \langle \Sigma, E, \mathbf{R}, \mathcal{I}' \rangle}$$

(EXEC-TICK)
$$\frac{\mathcal{I} \xrightarrow{\surd} \mathcal{I}'}{\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\surd} \langle \Sigma, E, \mathbf{R}, \mathcal{I}' \rangle}$$

(EXEC-ASSIGN)
$$\frac{\mathcal{I} \xrightarrow{\Sigma, [v/x, r]} \mathcal{I}'}{\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\tau} \langle \Sigma[v/x, r], E, \mathbf{R}, \mathcal{I}' \rangle}$$

(EXT-UPDATE)
$$\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{E', \mathbf{R}'} \langle \Sigma, E', \mathbf{R}', \mathcal{I} \rangle$$

Table 1. AIOC semantics.

Concerning actions, $o^? : r_1(v) \to r_2(x)$ denotes an interaction where r_1 sends to r_2 a value v on operation $o^?$. Value v will be stored in x by r_2. Label $[v/x, r]$ denotes an assignment of value v to local variable x of role r, while τ denotes a silent action (generated, e.g., by guard evaluation). Label $[l, \mathcal{C}] \mapsto \mathcal{I}'$ traces the application of an adaptation rule, while $[\texttt{no-adapt}]$ traces the begin of a scope that does not perform adaptation. Label $\surd$ denotes process termination.

We use μ to range over labels. To define labels, and, in particular, to deal with assumptions on information known only at the system level, we preferred simplicity and uniformity over minimality. E.g., when evaluating e one could only trace the variables occurring in e. This would lead however to a slightly more complex semantics.

The rules dealing with standard computation should be clear from the description of labels, and are described in the companion technical report [10]. Thus, we concentrate here on the rules describing adaptation. Rule ADAPT models the application of an adaptation rule that matches the scope label l, whose applicability condition $\mathcal{C}$ is satisfied, and that involves only roles occurring in the scope. As a result, *scope* $l@r\ \{\mathcal{I}\}$ *prop* $\{\Delta\}$ is replaced by the AIOC process $\mathcal{I}'$ in the body of the rule. One could have required the scope to remain after adaptation, thus allowing further adaptations. However, our choice is more general, since we can always let $\mathcal{I}'$ be a scope with the same label l. Note that the set of roles in an adaptation rule have to be a subset of the roles participating to the scope. This limitation is due to technical reasons, which will be clearer after having seen the semantics of the APOC. Intuitively, a role not involved in the scope provides no adaptation interface specifying where to insert the new code. We considered two possibilities for mitigating this drawback. First, roles not occurring in the scope could be declared as possibly relevant for future adaptations of the scope, so that information enabling their adaptation is generated. This possibility is part of the current implementation, and extending the theory to cope with it is straightforward. Second, new roles not occurring at all in the AIOC could be added dynamically, by inserting at the APOC level an action to create a new role executing a given process. Integrating this feature requires more work, and we leave it as a future extension.

If no rule inside $\mathbf{R}$ can be applied to the scope, rule NOADAPT removes the scope boundaries and starts the execution of the body of the scope. Allowing a scope to start only if no rule applies may not always appear realistic, but it is less strong than it seems, since $\mathbf{R}$ can be changed at any moment to simulate the unavailability of some rules (rule EXT-UPDATE). Furthermore, allowing adaptation to be skipped even if an applicable rule exists would not change our results, provided that the corresponding change is made also to the APOC semantics.

Let us consider the rules to lift the transitions of AIOC processes to AIOC systems. Rule EXEC deals with most of the actions, and it checks the assumption A before removing it from the label. Rule EXEC-TICK deals with $\surd$. Rule EXEC-ASSIGN deals with the label generated by assignments. Beyond checking the assumption on the state, it updates the state by assigning the computed value v to the desired variable x of role r. Then, the label becomes τ. Rule EXT-UPDATE is not used for lifting actions of the process level, but it specifies that the environment E and the set of rules $\mathbf{R}$ may arbitrarily change at any time, since they are not under the control of the system. Notably, the new environment E' and the new set of rules $\mathbf{R}'$ are visible in the label. This allows us to ensure that the changes to the environment and to the set of rules are applied at the same time both in the AIOC and in the APOC. More fine grained ways of changing the environment and the set of rules, e.g. removing or adding one rule or setting one environment name, can be obtained as particular cases of this rule.

We define *AIOC traces*, where all the performed actions are observed, and *weak AIOC traces*, where interactions on private operations and silent actions are not visible.

Definition 4 (AIOC traces). *A (strong) trace of an AIOC system* $\langle \Sigma_1, E_1, \mathbf{R}_1, \mathcal{I}_1 \rangle$ *is a sequence (finite or infinite) of labels* $\mu_1, \mu_2, \ldots$ *such that there is a sequence of AIOC system transitions* $\langle \Sigma_1, E_1, \mathbf{R}_1, \mathcal{I}_1 \rangle \xrightarrow{\mu_1} \langle \Sigma_2, E_2, \mathbf{R}_2, \mathcal{I}_2 \rangle \xrightarrow{\mu_2} \ldots$.
A weak trace of an AIOC system $\langle \Sigma_1, E_1, \mathbf{R}_1, \mathcal{I}_1 \rangle$ *is a sequence of labels* $\mu_1, \mu_2, \ldots$

obtained by removing all the labels corresponding to private communications, i.e. of the form $o^* : a(v) \to b(x)$, and the silent labels τ from a trace of $\langle \Sigma_1, E_1, \mathbf{R}_1, \mathcal{I}_1 \rangle$.

3 Adaptive Process-Oriented Choreography (APOC)

This section describes the syntax and operational semantics of APOCs. APOCs include *processes*, ranged over by P, P', ..., describing the behavior of participants. $(P, \Gamma)_r$ denotes an *APOC role* named r, executing process P with a local state Γ. *Networks*, ranged over by $\mathcal{N}$, $\mathcal{N}'$, ..., are parallel compositions of APOC roles with different names. APOC systems, ranged over by $\mathcal{S}$, are APOC networks equipped with an environment E and a set of adaptation rules $\mathbf{R}$, namely a triple $\langle E, \mathbf{R}, \mathcal{N} \rangle$.

$$
\begin{aligned}
P ::= {} & o^? : x \textit{ from } r \mid o^? : e \textit{ to } r \mid o^* : X \textit{ to } r \mid P; P' \mid P \mid P' \mid x = e \mid \mathbf{1} \mid \mathbf{0} \mid \\
& \textit{if } b \{P\} \textit{ else } \{P'\} \mid \textit{while } b \{P\} \mid n : \textit{scope } l@r \{P\} \textit{ prop } \{\Delta\} \textit{ roles } \{S\} \mid \\
& n : \textit{scope } l@r \{P\} \\
X ::= {} & \mathtt{no} \mid P \qquad \mathcal{N} ::= (P, \Gamma)_r \mid \mathcal{N} \parallel \mathcal{N}' \qquad \mathcal{S} ::= \langle E, \mathbf{R}, \mathcal{N} \rangle
\end{aligned}
$$

Processes include input action $o^? : x$ *from* r on a specific operation $o^?$ (either public or private) of a message from role r to be stored in variable x, output action $o^? : e$ *to* r of an expression e to be sent to role r, and higher-order output action $o^* : X$ *to* r of the higher-order argument X to be sent to role r. Here X may be either an APOC process P, which is the new code for a scope in r, or a flag `no`, notifying that no adaptation is needed. $P; P'$ and $P \mid P'$ denote the sequential and parallel composition of P and P', respectively. Processes also feature assignment $x = e$ of expression e to variable x, the process $\mathbf{1}$ that can only successfully terminate, and the deadlocked process $\mathbf{0}$. We also have conditionals *if* b $\{P\}$ *else* $\{P'\}$ and cycles *while* b $\{P\}$. Finally, we have two constructs for scopes. Scope n : *scope* $l@r$ $\{P\}$ *prop* $\{\Delta\}$ *roles* $\{S\}$ may occur only inside role r, and acts as coordinator to perform (or not perform) adaptation. The shorter version n : *scope* $l@r$ $\{P\}$ is enough when the role is not the coordinator for this scope. In fact, only the coordinator needs to know the non-functional properties Δ of the scope and the set S of involved roles. Note that scopes are prefixed by an index n: the distributed execution of the corresponding AIOC constructs requires to coordinate different roles, and unique indexes are needed to avoid interference between different scopes in the same role with the same label.

3.1 Projection

Before defining the semantics of APOCs, we need to define the projection of an AIOC process onto APOC processes. Indeed, this is needed to apply an adaptation rule at the APOC level. The projection exploits some auxiliary communications to coordinate the different roles, e.g., ensuring that in a conditional they all select the same branch. To define these auxiliary communications and avoid interferences, it is convenient to annotate AIOC main constructs with unique indexes.

Definition 5 (Well-annotated AIOC). *Annotated AIOC processes are obtained by indexing every interaction, assignment, scope, if and while in an AIOC process with a*

$$\pi(\mathbf{1}, s) = \mathbf{1} \quad \pi(\mathbf{0}, s) = \mathbf{0} \quad \pi(\mathcal{I}; \mathcal{I}', s) = \pi(\mathcal{I}, s); \pi(\mathcal{I}', s) \quad \pi(\mathcal{I} \parallel \mathcal{I}', s) = \pi(\mathcal{I}, s) \mid \pi(\mathcal{I}', s)$$

$$\pi(n : o^{?} : r_1(e) \to r_1(x), s) = \begin{cases} o^{?} : e \textit{ to } r_2 & \text{if } s = r_1 \\ o^{?} : x \textit{ from } r_1 & \text{if } s = r_2 \\ \mathbf{1} & \text{otherwise} \end{cases} \quad \pi(n : x@r = e, s) = \begin{cases} x = e & \text{if } s = r \\ \mathbf{1} & \text{otherwise} \end{cases}$$

$$\pi(n : \textit{if } b@r \ \{\mathcal{I}\} \textit{ else } \{\mathcal{I}'\}, s) = \begin{cases} \text{when } s = r : \\ \textit{if } b \ \{(\Pi_{r' \in \mathrm{roles}(\mathcal{I}) \cup \mathrm{roles}(\mathcal{I}') \setminus \{r\}} o_n^* : \textit{true to } r'); \pi(\mathcal{I}, s)\} \\ \textit{else } \{(\Pi_{r' \in \mathrm{roles}(\mathcal{I}) \cup \mathrm{roles}(\mathcal{I}') \setminus \{r\}} o_n^* : \textit{false to } r'); \pi(\mathcal{I}', s)\} \\ \text{when } s \in \mathrm{roles}(\mathcal{I}) \cup \mathrm{roles}(\mathcal{I}') \setminus \{r\} : \\ o_n^* : x_n \textit{ from } r; \textit{if } x_n \ \{\pi(\mathcal{I}, s)\} \textit{ else } \{\pi(\mathcal{I}', s)\} \\ \text{otherwise : } \mathbf{1} \end{cases}$$

$$\pi(n : \textit{while } b@r \ \{\mathcal{I}\}, s) = \begin{cases} \text{when s = r :} \\ \textit{while } b \ \{(\Pi_{r' \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} o_n^* : \textit{true to } r'); \pi(\mathcal{I}, s); \\ \quad \Pi_{r' \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} o_n^* : \_ \textit{ from } r'\}; \Pi_{r' \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} o_n^* : \textit{false to } r' \\ \text{when } s \in \mathrm{roles}(\mathcal{I}) \setminus \{r\} : \\ o_n^* : x_n \textit{ from } r; \textit{while } x_n \ \{\pi(\mathcal{I}, s); o_n^* : \mathtt{ok} \textit{ to } r; o_n^* : x_n \textit{ from } r\} \\ \text{otherwise : } \mathbf{1} \end{cases}$$

$$\pi(n : \textit{scope } l@r \ \{\mathcal{I}\} \textit{ prop } \{\Delta\}, s) = \begin{cases} \text{when } s = r : \\ n : \textit{scope } l@r \ \{\pi(\mathcal{I}, s)\} \textit{ prop } \{\Delta\} \textit{ roles } \{\mathrm{roles}(\mathcal{I})\} \\ \text{when } s \in \mathrm{roles}(\mathcal{I}) \setminus \{r\} : n : \textit{scope } l@r \ \{\pi(\mathcal{I}, s)\} \\ \text{otherwise : } \mathbf{1} \end{cases}$$

Table 2. Process-projection function π.

natural number $n \in \mathbb{N}$. *This results in the following grammar:*

$$\mathcal{I} ::= n : o^{?} : r_1(e) \to r_2(x) \mid \mathcal{I}; \mathcal{I}' \mid \mathcal{I} | \mathcal{I}' \mid n : x@r = e \mid n : \textit{while } b@r \ \{\mathcal{I}\} \mid$$
$$n : \textit{if } b@r \ \{\mathcal{I}\} \textit{ else } \{\mathcal{I}'\} \mid \mathbf{1} \mid \mathbf{0} \mid n : \textit{scope } l@r \ \{\mathcal{I}\} \textit{ prop } \{\Delta\}$$

An AIOC process is well-annotated if all its indexes are distinct.

We now define the *process-projection function* that derives APOC processes from AIOC processes. If the AIOC process is not annotated, we add indexes to make it well-annotated. Given an annotated AIOC process $\mathcal{I}$ and a role s, the projected APOC process $\pi(\mathcal{I}, s)$ is defined by structural induction on $\mathcal{I}$ in Table 2. In most of the cases the projection is trivial. For instance, the projection of an interaction is an output on the sender role, an input on the receiver, and $\mathbf{1}$ on any other role. For a conditional $n : \textit{if } b@r \ \{\mathcal{I}\} \textit{ else } \{\mathcal{I}'\}$, role r locally evaluates the condition and then sends its value to the other roles using auxiliary communications. Similarly, in a cycle $n : \textit{while } b@r \ \{\mathcal{I}\}$ role r communicates the evaluation of the condition to the other roles. Also, after a cycle has terminated it waits for the other roles to terminate, and then starts a new cycle. In both the conditional and the cycle, indexes are used to choose names for auxiliary operations: the choice is coherent among the different roles, and interference between different cycles or conditionals is avoided.

In the companion technical report [10] one can find the APOC processes obtained by projecting the AIOC for the Buying scenario on `buyer`, `seller` and `bank`.

(IN)
$$o^? : x \textit{ from } r \xrightarrow{o^?(x \leftarrow v)@r} x = v$$

(OUT)
$$\frac{[\![e]\!]_\Gamma = v}{o^? : e \textit{ to } r \xrightarrow{\Gamma, \overline{o^?}\langle v\rangle@r} \mathbf{1}}$$

(OUT-ADAPT)
$$o^? : X \textit{ to } r \xrightarrow{\overline{o^?}\langle X\rangle@r} \mathbf{1}$$

(ONE)
$$\mathbf{1} \xrightarrow{\surd} \mathbf{0}$$

(ASSIGN)
$$\frac{[\![e]\!]_\Gamma = v}{x = e \xrightarrow{\Gamma, [v/x]} \mathbf{1}}$$

(SEQUENCE)
$$\frac{P \xrightarrow{\delta} P' \quad \delta \neq \surd}{P; Q \xrightarrow{\delta} P'; Q}$$

(SEQ-END)
$$\frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\delta} Q'}{P; Q \xrightarrow{\delta} Q'}$$

(PARALLEL)
$$\frac{P \xrightarrow{\delta} P' \quad \delta \neq \surd}{P \mid Q \xrightarrow{\delta} P' \mid Q}$$

(PAR-END)
$$\frac{P \xrightarrow{\surd} P' \quad Q \xrightarrow{\surd} Q'}{P \mid Q \xrightarrow{\surd} P' \mid Q'}$$

(IF-THEN)
$$\frac{[\![b]\!]_\Gamma = \texttt{true}}{\textit{if } b \textit{ } \{P\} \textit{ else } \{P'\} \xrightarrow{\Gamma, \tau} P}$$

(IF-ELSE)
$$\frac{[\![b]\!]_\Gamma = \texttt{false}}{\textit{if } b \textit{ } \{P\} \textit{ else } \{P'\} \xrightarrow{\Gamma, \tau} P'}$$

(WHILE-UNFOLD)
$$\frac{[\![b]\!]_\Gamma = \texttt{true}}{\textit{while } b \textit{ } \{P\} \xrightarrow{\Gamma, \tau} P; k : \textit{while } e \textit{ } \{P\}}$$

(WHILE-EXIT)
$$\frac{[\![b]\!]_\Gamma = \texttt{false}}{\textit{while } b \textit{ } \{P\} \xrightarrow{\Gamma, \tau} \mathbf{1}}$$

(LEAD-ADAPT)
$$\frac{\textit{rule } l \textit{ where } \mathcal{C} \textit{ specifies } \mathcal{I} \in \mathbf{R} \quad \Gamma, E, \Delta \vdash \mathcal{C} \quad \mathcal{I}' = \mathrm{freshIndex}(\mathcal{I}, n) \quad \mathrm{roles}(\mathcal{I}) \subseteq S}{n : \textit{scope } l@r \textit{ } \{P\} \textit{ prop } \{\Delta\} \textit{ roles } \{S\} \xrightarrow{\Gamma, E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}} \Pi_{r_i \in S \setminus \{r\}} o^*_{l,n} : \pi(\mathcal{I}', r_i) \textit{ to } r_i; \pi(\mathcal{I}', r); \Pi_{r_i \in S \setminus \{r\}} o^*_{l,n} : \_ \textit{ from } r_i}$$

(LEAD-NOADAPT)
$$\frac{\forall \textit{ rule } l \textit{ where } \mathcal{C} \textit{ specifies } \mathcal{I} \in \mathbf{R} \,.\, \Gamma, E, \Delta \not\vdash \mathcal{C} \vee \mathrm{roles}(\mathcal{I}) \not\subseteq S}{n : \textit{scope } l@r \textit{ } \{P\} \textit{ prop } \{\Delta\} \textit{ roles } \{S\} \xrightarrow{\Gamma, E, \mathbf{R}, [\texttt{no-adapt}]} \Pi_{r_i \in S \setminus \{r\}} o^*_{l,n} : \mathrm{no} \textit{ to } r_i; P; \Pi_{r_i \in S \setminus \{r\}} o^*_{l,n} : \_ \textit{ from } r_i}$$

(ADAPT)
$$n : \textit{scope } l@r \textit{ } \{P\} \xrightarrow{o^*_{l,n}(\_ \leftarrow P')@r} P'; o^*_{l,n} : \mathtt{ok} \textit{ to } r$$

(NOADAPT)
$$n : \textit{scope } l@r \textit{ } \{P\} \xrightarrow{o^*_{l,n}(\_ \leftarrow \mathrm{no})@r} P; o^*_{l,n} : \mathtt{ok} \textit{ to } r$$

Table 3. APOC processes semantics.

We now define the projection $\mathrm{proj}(\mathcal{I}, \Sigma)$ to derive an APOC network from an AIOC process $\mathcal{I}$ and a global state Σ. The function proj is based on the process-projection π. We denote with $\|_{i \in I}\, \mathcal{N}_i$ the parallel composition of networks $\mathcal{N}_i$ for each $i \in I$.

Definition 6 (Projection). *The projection of an AIOC process $\mathcal{I}$ with global state Σ is the APOC network defined by* $\mathrm{proj}(\mathcal{I}, \Sigma) = \|_{s \in \mathrm{roles}(\mathcal{I})}\, (\pi(\mathcal{I}, s), \Sigma_s)_s$

3.2 APOC semantics

Definition 7 (APOC systems semantics). *The semantics for of APOC systems is defined as the smallest LTS closed under the rules in Tables 3 and 4. Symmetric rules for parallel composition have been omitted.*

Table 3 defines the transitions of APOC processes. As for the AIOCs, information not available at process level is guessed, and checked when the desired level is reached.

Assumptions may include the local state Γ, the environment E and the set of rules $\mathbf{R}$ (to check rule applicability), or the local state Γ only (to evaluate an expression). Concerning actions, $\overline{o^?}\langle v\rangle@r$ defines the output from role r on operation $o^?$ of value v. Label $o^?(x \leftarrow v)@r$ denotes the input on $o^?$ that receives value v from r and stores it into variable x (the value v is guessed). Assignment label $[v/x]$ traces the substitution to be applied to the local state. Silent action τ is produced, e.g., by the evaluation of a boolean expression. Label $\overline{o^?}\langle X\rangle@r$ denotes the higher-order output on operation $o^?$ of X from role r. Here X is either a process P or a flag `no`. Label $o^*_{l,k} : X$ *from* r is the corresponding input, waiting for a process P or a flag `no` on operation $o^*_{l,k}$. Label $[l, \mathcal{C}] \mapsto \mathcal{I}'$ denotes the application of the adaptation *rule* l *where* $\mathcal{C}$ *specifies* $\mathcal{I}'$. Label $[\texttt{no-adapt}]$ specifies that no adaptation is needed. Label $\surd$ denotes termination. We use δ to range over labels at the level of APOC processes, β at the level of APOC roles, χ at the level of APOC networks, and χ and η at the level of APOC systems.

As for AIOCs, the rules dealing with standard computation should be clear from the description of labels, and are described in the companion technical report [10]. Thus, we concentrate here on the rules describing adaptation. Rule LEAD-ADAPT concerns the role r coordinating the adaptation of a scope. Role r checks whether an applicable rule exists, by evaluating the applicability condition and verifying that only roles involved in the scope are needed. If adaptation is needed, r transforms the AIOC $\mathcal{I}$ into $\mathcal{I}'$ using function $\mathrm{freshIndex}(\mathcal{I}, n)$. This has a double aim. On the one hand, it changes the indexes n of scopes into fresh indexes, to avoid clashes with indexes in the target APOC. On the other hand, it renames all the operations by adding to them the index n of the scope. We assume to this end to extend the set of operations, without changing the semantics. We write the new operations as $n \cdot o^?$, where $o^?$ is an old one. Note that complementary input and output are still complementary after the transformation. In case of adaptation, r generates the processes to be executed by the roles in S using the process-projection function π. The processes are sent via higher-order communications to the participants that should execute them. Then, r starts its own updated code, i.e., $\pi(\mathcal{I}', r)$. Finally, auxiliary communications are used to synchronize the end of the execution of the adapted process (here $\_$ denotes an unused variable to store the synchronization message `ok`). The auxiliary communications are needed to ensure that the update is performed in a coordinated way, i.e. the roles agree on when the scope is started and terminated, and on whether adaptation is performed or not. Rule LEAD-NOADAPT instead defines the behavior of the coordinator role r when no adaptation rule is applicable: in this case r sends a message containing a flag `no` to each other involved role, notifying them that no adaptation has to be performed. End of scope synchronization is as above. Rules ADAPT and NOADAPT define the behavior of adaptation scopes for the other roles. The scope waits for a message from the coordinator role. If the message content is `no`, the current body of the scope is executed. If it is a process P', P' itself is executed instead of the scope.

Then, in Table 4, we have the rules that lift the APOC process actions to APOC roles, networks, and systems. Rule LIFT deals with τ labels and adaptation labels. The guess on the local state Γ is checked, and Γ is removed from the label. Rule LIFT-ASSIGN deals with assignment: the guess on Γ is checked, and the state is changed to keep into account the performed assignment. The label becomes a τ. Rule LIFT-TICK deals with termination. Rules LIFT-COMM and LIFT-OUT deal with input and output

APOC roles

(LIFT)
$$\frac{P \xrightarrow{\Gamma,\beta} P' \quad \beta = \tau \ \vee \ \beta = E, \mathbf{R}, [\texttt{no-adapt}] \ \vee \ \beta = E, \mathbf{R}, [\texttt{l}, \mathcal{C}] \mapsto \mathcal{I}'}{(P, \Gamma)_r \xrightarrow{\beta} (P', \Gamma)_r}$$

(LIFT-ASSIGN)
$$\frac{P \xrightarrow{\Gamma,[v/x]} P'}{(P, \Gamma)_r \xrightarrow{\tau} (P', \Gamma[v/x])_r}$$

(LIFT-TICK)
$$\frac{P \xrightarrow{\surd} P'}{(P, \Gamma)_r \xrightarrow{\surd} (P', \Gamma)_r}$$

(LIFT-OUT)
$$\frac{P \xrightarrow{\Gamma,\overline{o^?}\langle v\rangle@r_1} P'}{(P, \Gamma)_{r_2} \xrightarrow{\beta:r_2} (P', \Gamma)_{r_2}}$$

(LIFT-COMM)
$$\frac{P \xrightarrow{\beta} P' \quad \beta = o^?(x \leftarrow v)@r_1 \ \vee \ \beta = o^*_{l,k}(\_ \leftarrow X)@r_1 \ \vee \ \beta = \overline{o^?}\langle X\rangle@r_1}{(P, \Gamma)_{r_2} \xrightarrow{\beta:r_2} (P', \Gamma)_{r_2}}$$

APOC networks

(SYNCH)
$$\frac{\mathcal{N} \xrightarrow{\overline{o^?}\langle v\rangle@r_2:r_1} \mathcal{N}' \quad \mathcal{N}'' \xrightarrow{o^?(x\leftarrow v)@r_1:r_2} \mathcal{N}'''}{\mathcal{N} \parallel \mathcal{N}'' \xrightarrow{o^?:r_1(v)\rightarrow r_2(x)} \mathcal{N}' \parallel \mathcal{N}'''}$$

(EXT-PARALLEL)
$$\frac{\mathcal{N} \xrightarrow{\chi} \mathcal{N}' \quad \chi \neq \surd}{\mathcal{N} \parallel \mathcal{N}'' \xrightarrow{\chi} \mathcal{N}' \parallel \mathcal{N}''}$$

(SYNCH-ADAPT)
$$\frac{\mathcal{N} \xrightarrow{\overline{o^?}\langle X\rangle@r_2:r_1} \mathcal{N}' \quad \mathcal{N}'' \xrightarrow{o^?(\_\leftarrow X)@r_1:r_2} \mathcal{N}'''}{\mathcal{N} \parallel \mathcal{N}'' \xrightarrow{o^?:r_1(X)\rightarrow r_2(\_)} \mathcal{N}' \parallel \mathcal{N}'''}$$

(EXT-PAR-END)
$$\frac{\mathcal{N} \xrightarrow{\surd} \mathcal{N}' \quad \mathcal{N}'' \xrightarrow{\surd} \mathcal{N}'''}{\mathcal{N} \parallel \mathcal{N}'' \xrightarrow{\surd} \mathcal{N}' \parallel \mathcal{N}'''}$$

APOC systems

(EXEC)
$$\frac{\mathcal{N} \xrightarrow{\chi} \mathcal{N}' \quad \chi \in \{o^? : r_1(v) \rightarrow r_2(x), o^? : r_1(X) \rightarrow r_2(\_), \tau, \surd\}}{\langle E, \mathbf{R}, \mathcal{N}\rangle \xrightarrow{\chi} \langle E, \mathbf{R}, \mathcal{N}'\rangle}$$

(EXEC-ADAPT)
$$\frac{\mathcal{N} \xrightarrow{E,\mathbf{R},\eta} \mathcal{N}' \quad \eta = [\texttt{l}, \mathcal{C}] \mapsto \mathcal{I}' \vee \eta = [\texttt{no-adapt}]}{\langle E, \mathbf{R}, \mathcal{N}\rangle \xrightarrow{\eta} \langle E, \mathbf{R}, \mathcal{N}'\rangle}$$

(EXT-UPDATE)
$$\langle E, \mathbf{R}, \mathcal{N}\rangle \xrightarrow{E',\mathbf{R}'} \langle E', \mathbf{R}', \mathcal{N}\rangle$$

Table 4. APOC semantics.

actions. The information on the participant doing the action is added to the label. Only in rule LIFT-OUT the guess on the state has to be checked. At APOC networks level, rule SYNCH synchronizes an output with the corresponding input, producing an interaction. Rule SYNCH-ADAPT is similar, but it deals with higher-order interactions. The labels of these transitions store the information on the occurred communication: label $o^? : r_1(v) \rightarrow r_2(x)$ denotes an interaction on operation $o^?$ from role r_1 to role r_2 where the value v is sent by r_1 and then stored by r_2 in variable x. Label $o^? : r_1(X) \rightarrow r_2(\_)$ denotes a similar interaction, but concerning a higher-order value X. No receiver variable is specified, since the received value becomes part of the code of the receiving process. Rule EXT-PARALLEL allows a network inside a parallel composition to compute. Rule EXT-PAR-END synchronizes the termination of parallel networks.

Transitions are lifted to APOC systems by rules EXEC and EXEC-ADAPT. In particular, rule EXEC-ADAPT checks the assumptions on the environment and on the set

of rules, and removes the corresponding information from the label. Notably, input and output actions are not lifted: they are meant for internal communication, and are not relevant when considering the whole system. Furthermore, there are no correspondent labels in AIOCs. Finally, rule EXT-UPDATE allows the environment and the set of rules to change arbitrarily.

Definition 8 (APOC traces). *A (strong) trace of an APOC system $\langle E_1, \mathbf{R}_1, \mathcal{N}_1 \rangle$ is a sequence (finite or infinite) of labels $\chi_1, \chi_2, \ldots$ such that there is a sequence of transitions $\langle E_1, \mathbf{R}_1, \mathcal{N}_1 \rangle \xrightarrow{\chi_1} \langle E_2, \mathbf{R}_2, \mathcal{N}_2 \rangle \xrightarrow{\chi_2} \ldots$.*
A weak trace of an APOC system $\langle E_1, \mathbf{R}_1, \mathcal{N}_1 \rangle$ is a sequence of labels $\chi_1, \chi_2, \ldots$ obtained by removing all the labels corresponding to private communications, i.e. of the form $o^* : a(v) \to b(x)$ or $o^* : r_1(X) \to r_2(\_)$, and the silent labels τ, from a trace of $\langle E_1, \mathbf{R}_1, \mathcal{N}_1 \rangle$. Furthermore, all the operations of the form $n \cdot o^?$ are replaced by $o^?$.

In the companion technical report [10] one can find a sample execution of the APOC obtained by projecting the AIOC for the Buying scenario.

4 Correctness of the adaptation

In the previous sections we have presented AIOCs, APOCs, and described how to derive an APOC from a given AIOC. This section presents the main technical result of the paper, namely the correctness of the projection. Correctness here means that the weak traces of the AIOC coincide with the weak traces of the projected APOC. Unfortunately, this does not hold for arbitrary AIOCs, but only for AIOCs and adaptation rules which are *connected*. We provide below a syntactic characterization of connectedness, and we prove in Theorem 2 that projections of connected AIOCs, adapted using connected AIOC rules, have the expected set of weak traces. To formally define the connectedness conditions we introduce, in Table 5, the auxiliary functions transI and transF, that given an AIOC process compute sets of pairs representing senders and receivers of initial and final interactions. We represent one such pair as $r_1 \to r_2$. Actions located at r are represented as $r \to r$. We also assume a function op that given an AIOC process returns the set of signatures of its interactions, where the signature of interaction $o^? : r_1(e) \to r_2(x)$ is $o^? : r_1 \to r_2$.

Definition 9 (Connectedness). *An AIOC process $\mathcal{I}$ is connected if it satisfies:*

connectedness for sequence: *each sub term of the form $\mathcal{I}'; \mathcal{I}''$ satisfies $\forall r_1 \to r_2 \in \mathrm{transF}(\mathcal{I}'), \forall s_1 \to s_2 \in \mathrm{transI}(\mathcal{I}'') \,.\, \{r_1, r_2\} \cap \{s_1, s_2\} \neq \emptyset$;*
connectedness for parallel: *each sub term of the form $\mathcal{I}' \parallel \mathcal{I}''$ satisfies $\mathrm{op}(\mathcal{I}') \cap \mathrm{op}(\mathcal{I}'') = \emptyset$.*

If $\mathcal{I}$ is connected then rule l where $\mathcal{C}$ specifies $\mathcal{I}$ is a connected adaptation rule.

Intuitively, connectedness for sequence ensures that the APOC network obtained by projecting a sequence $\mathcal{I}; \mathcal{I}'$ executes first the actions in $\mathcal{I}$, and then the ones in $\mathcal{I}'$.

Connectedness for parallel is needed to ensure that different interactions with the same signature do not interfere (cfr. rule SYNCH in Table 4): we require that two such interactions never occur in parallel AIOC processes.

Connectedness can be checked efficiently.

$$\mathrm{transI}(o^? : r_1(e) \to r_2(x)) = \mathrm{transF}(o^? : r_1(e) \to r_2(x)) = \{r_1 \to r_2\}$$
$$\mathrm{transI}(x@r = e) = \mathrm{transF}(x@r = e) = \{r \to r\}$$
$$\mathrm{transI}(\mathbf{1}) = \mathrm{transI}(\mathbf{0}) = \mathrm{transF}(\mathbf{1}) = \mathrm{transF}(\mathbf{0}) = \emptyset$$
$$\mathrm{transI}(\mathcal{I} \parallel \mathcal{I}') = \mathrm{transI}(\mathcal{I}) \cup \mathrm{transI}(\mathcal{I}') \qquad \mathrm{transF}(\mathcal{I} \parallel \mathcal{I}') = \mathrm{transF}(\mathcal{I}) \cup \mathrm{transF}(\mathcal{I}')$$
$$\mathrm{transI}(\mathcal{I}; \mathcal{I}') = \begin{cases} \mathrm{transI}(\mathcal{I}') & \text{if } \mathrm{transI}(\mathcal{I}) = \emptyset \\ \mathrm{transI}(\mathcal{I}) & \text{otherwise} \end{cases}$$
$$\mathrm{transF}(\mathcal{I}; \mathcal{I}') = \begin{cases} \mathrm{transF}(\mathcal{I}) & \text{if } \mathrm{transF}(\mathcal{I}') = \emptyset \\ \mathrm{transF}(\mathcal{I}') & \text{otherwise} \end{cases}$$
$$\mathrm{transI}(\textit{if } b@r\ \{\mathcal{I}\}\ \textit{else}\ \{\mathcal{I}'\}) = \mathrm{transI}(\textit{while } b@r\ \{\mathcal{I}\}) = \{r \to r\}$$
$$\mathrm{transF}(\textit{if } b@r\ \{\mathcal{I}\}\ \textit{else}\ \{\mathcal{I}'\}) = \begin{cases} \{r \to r\} & \text{if } \mathrm{transF}(\mathcal{I}) \cup \mathrm{transF}(\mathcal{I}') = \emptyset \\ \mathrm{transF}(\mathcal{I}) \cup \mathrm{transF}(\mathcal{I}') & \text{otherwise} \end{cases}$$
$$\mathrm{transF}(\textit{while } b@r\ \{\mathcal{I}\}) = \begin{cases} \{r \to r\} & \text{if } \mathrm{transF}(\mathcal{I}) = \emptyset \\ \mathrm{transF}(\mathcal{I}) & \text{otherwise} \end{cases}$$
$$\mathrm{transI}(n : \textit{scope } l@r\ \{\mathcal{I}\}\ \textit{prop}\ \{\Delta\}) = \{r \to r\}$$
$$\mathrm{transF}(n : \textit{scope } l@r\ \{\mathcal{I}\}\ \textit{prop}\ \{\Delta\}) = \begin{cases} \{r \to r\} & \text{if } \mathrm{roles}(\mathcal{I}) \subseteq \{r\} \\ \bigcup_{r' \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} \{r' \to r\} & \text{otherwise} \end{cases}$$

Table 5. Auxiliary functions transI, transF.

Theorem 1 (Connectedness-check complexity). *The connectedness of an AIOC process $\mathcal{I}$ can be checked in time $O(n^2 \log(n))$, where n is the number of nodes in the abstract syntax tree of $\mathcal{I}$.*

Proof. For reviewers' convenience the proof is in the Appendix.

Definition 10 (Trace equivalence). *An AIOC system $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ and an APOC system $\langle E, \mathbf{R}, \mathcal{N} \rangle$ are (weak) trace equivalent iff their sets of (weak) traces coincide.*

Theorem 2 (Correctness). *For each initial, connected AIOC process $\mathcal{I}$, each state Σ, each environment E, each set of initial, connected rules $\mathbf{R}$, the AIOC system $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ and the APOC system $\langle E, \mathbf{R}, \mathrm{proj}(\mathcal{I}, \Sigma) \rangle$ are weak trace equivalent if all the rules in the sets of rules provided by the adaptation middleware are initial and connected.*

Proof. For reviewers' convenience the proof is in the Appendix.

Trace-based properties of the AIOC are inherited by the APOC. Examples include deadlock-freedom and termination.

Definition 11 (Deadlock-freedom and termination). *An internal AIOC (resp. APOC) trace is obtained by removing transitions labeled $E, \mathbf{R}$ (for each E and $\mathbf{R}$) from an AIOC (resp. APOC) trace. An AIOC (resp. APOC) system is deadlock-free if all its maximal finite internal traces have $\surd$ as label of the last transition. An AIOC (resp. APOC) system terminates if all its internal traces are finite.*

By construction initial AIOCs are deadlock-free. Hence:

Corollary 1. *For each initial, connected AIOC $\mathcal{I}$, state Σ, set of adaptation rules $\mathbf{R}$ and environment E, the APOC system $\langle E, \mathbf{R}, \mathrm{proj}(\mathcal{I}, \Sigma) \rangle$ is deadlock-free.*

APOCs inherit termination from terminating AIOCs. Note that for ensuring AIOC termination one should restrict the set of applicable adaptation rules.

Corollary 2. *For each AIOC system $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ where $\mathcal{I}$ is connected, if the AIOC system terminates then the APOC system $\langle E, \mathbf{R}, \mathrm{proj}(\mathcal{I}, \Sigma) \rangle$ terminates.*

Proof. It follows from the fact that only a finite number of auxiliary actions are added when moving from AIOCs to APOCs.

5 Implementation

In order to validate our approach we implemented it, by combining and extending the frameworks in [6, 16]. The implementation, called AIOCJ, is composed by:

- AIOCJ-ecl, an Eclipse [11] plug-in, allowing to edit AIOCs and adaptation rules, to check connectedness, and to generate the code for each participant. In AIOCJ-ecl, the AIOC description is extended with deployment information.
- AIOCJ-mid, a service-oriented adaptation middleware supporting the execution and adaptation of applications generated by AIOCJ-ecl. AIOCJ-mid includes distributed adaptation servers, which can be activated and deactivated at run-time. Furthermore, the set of rules of each adaptation server can be changed dynamically. AIOCJ-mid also includes a service acting as environment.

AIOCJ-ecl is a Java application, based on Xtext [24]. The use of Xtext has two main advantages. First, it provides for free features such as syntax highlighting and code completion, helping the developer in writing AIOCs and adaptation rules. Second, it generates an abstract syntax tree for AIOCs and adaptation rules, which we used to check the connectedness and to generate the executable code corresponding to the APOC level. The generated code is provided in Jolie [14, 19], a full-fledged, open source, service-oriented language. We also used Jolie to implement the adaptation middleware. The use of Jolie has several advantages:

- Jolie constructs are similar to those of APOCs, thus allowing a straightforward mapping from the APOC generated by the projection to the Jolie code. In particular, Jolie features message-passing communication and a native parallel operator;
- Jolie supports some mechanisms, such as dynamic embedding, aggregation, and redirection, suitable to implement APOC adaptation mechanisms and managing the related communication patterns.

In principle, one could have implemented the APOC behavior and the adaptation middleware in more widely-used languages such as Java, Haskell, or C#, using reflection (the possibility of examining and modifying at run-time the structure and behavior of a program) to program adaptation mechanisms. However, for the kind of adaptation we consider, using Jolie dynamic embedding, allowing a service to get new code and run it as a sub-service, is simpler. Indeed, the projection generates a Jolie service for each role. This service can be immediately deployed on the network. The execution of scopes is delegated to sub-services. Adaptation is enacted by removing the current sub-service and replacing it with a new one, obtained from the adaptation server. For simplicity, the projection of the body of the rule on the different roles is pre-compiled when the code for the rule is generated.

A main difficulty when implementing the APOC behavior concerns communications: indeed, Jolie provides asynchronous communications, while the APOC semantics is synchronous. We solved this issue using a message handler implementing a synchronous interaction as a request-response service invocation, namely a forward communication followed by an acknowledgment notifying that the message has been received by the target process. In the running application, each role and each scope inside it feature their own message handler. However, a unique message handler for each role is visible from the outside. This is done by using Jolie aggregation mechanism, which allows a service to expose together the functionalities of two or more services. Messages are then sent to the correct target by means of Jolie redirection, allowing to dynamically find the target of a given message.

AIOCJ, including its documentation, the sources of the Buying scenario and the related adaptation rules, and additional examples, is available on the web [1].

6 Related works and discussion

In this paper we presented an approach for rule-based adaptation of distributed applications. The distinctive trait of our approach is that the executable APOC is guaranteed to behave well under all the run-time updates provided by all the possible sets of initial, connected adaptation rules, for any environment condition. In particular, the APOC is compliant with the AIOC specification and is deadlock free.

Our work is related to both works on choreographies/multiparty session types and on adaptation. For this reason, we examined all the papers on session types considered in [3] and all the papers on adaptation in [18], as well as many others. Nevertheless, for space reasons, we discuss here only the works which are closest to ours.

Our approach differs from multiparty session types [5–7, 13] since our AIOC and APOC languages provide executable code, not only types. Moreover, we are not aware of any work on multiparty session types enabling adaptation. In this context, the works closest to ours are [2], on dynamic software updates, and [9], on monitoring for self-adaptive systems. However, in [2] dynamic software updates are simply applied on demand, while here enactment of adaptation depends on the state of the environment and of the running system. Furthermore, [2] targets concurrent systems which are not distributed, thus it can rely on a global predicate on the structure of the system to ensure that an update can be safely applied. In our case all initial, connected updates can be safely applied, and local checks are used to understand whether they are beneficial or not. Finally, the language in [2] is much more constrained than ours, e.g. requiring each pair of participants to interact on a dedicated pair of channels, and assuming that all the roles not involved in a choice behave the same in the two branches. The approach in [9] is different too: it considers self-adaptive systems monitored by different global descriptions. The description specifies also when the used monitor should change, and the new monitor to be used is determined by an adaptation function. As a main difference, they have no notion of external adaptation middleware, and replace the whole type of the system, not only part of it. Finally, their code does not change because processes should be able to implement all the global descriptions since the very beginning.

The authors of [15] define rules for adapting the specification of the initial requirements for a choreography, thus keeping the requirements up-to-date in presence of run-

time changes. Our approach is in the opposite direction: we are not interested in updating the system specification tracking system updates, but in specifying and ensuring correctness of system adaptation itself.

Other formal approaches to adaptation represent choreographies as annotated finite state automata. [21] uses choreographies to propagate protocol changes to the other peers, while [23] presents a test to check whether a set of peers obtained from a choreography can be reconfigured to match a second one. Differently from ours, these works only provide change recommendations for adding and removing message sequences.

In [26], programs are represented by Finite Stare Machines and adaptation steps as transitions among them. Modular model checking is used to verify properties specified in Linear Temporal Logic, extended with an "adapt-operator" Our approach is different, since we consider distributed applications, and we can ensure relevant correctness properties (e.g., deadlock-freedom) by construction, without the need to use computationally heavy verification techniques.

Our work is quite different from most other approaches to adaptation, such as [4, 8, 12, 16, 20, 25], since they do not provide formal guarantees on the behavior of the adapted system. Among them, one of the closest to ours is [16], which proposes a rule-based adaptation in a style similar to ours. Indeed, [16] validates our architectural approach, where code regions to be adapted are syntactically delimited, and the running system interacts with an adaptation middleware using adaptation rules to replace those code regions at run-time. However, differently from ours, adaptable applications in [16] are not distributed. Furthermore [16] does not provide guarantees on the (correct) behavior of the adapted system and has no concept corresponding to our AIOC specification.

Our work is also related to distributed [20] and dynamic [25] Aspect-Oriented Programming (AOP). Indeed, in [1] we show examples taken from them. In general, we can deal with cross-cutting concerns like logging and authentication, typical of AOP, viewing pointcuts as empty scopes and advices as adaptation rules. This allows us to ensure deadlock freedom and similar properties which are not provided by AOP.

The approach presented in this paper can be refined in various directions, and we plan to explore them in future work. A main trade-off in the technical development concerns how to ensure that all the roles are aware of the evolution of the computation. This can be done in three ways: using auxiliary communications generated either by the projection (e.g., for if and while constructs) or by the semantics (e.g., for scopes), or restricting the class of allowed AIOCs (as done for sequential composition using connectedness for sequence). The last solution leaves more burden on the shoulders of the programmer, but allows for optimizations on the number of auxiliary communications, which is currently quite high. In this case, one can extend the techniques in [17] to transform non-connected AIOCs into connected AIOCs to help the programmer, but this may forbid some optimizations. Another aspect deserving more studies is the matching of adaptation rules with scopes, which is now based only on labels. One can easily imagine to use more refined techniques based on preconditions and postconditions specified in a suitable assertion language, and even to exploit information provided by specific ontologies in order to express more sophisticated matching policies.

References

1. http://www.cs.unibo.it/projects/jolie/aiocj.html.
2. G. Anderson and J. Rathke. Dynamic software update for message passing programs. In *APLAS*, volume 7705 of *LNCS*, pages 207–222. Springer, 2012.
3. http://www.operationalsemantics.net/behaviouralwiki/doku.php?id=wg1_2013_bibliography.
4. A. Bucchiarone, A. Marconi, M. Pistore, and H. Raik. Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes. In *ICWS*, pages 33–41. IEEE Press, 2012.
5. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012.
6. M. Carbone and F. Montesi. Deadlock-Freedom-by-Design: Multiparty Asynchronous Global Programming. In *POPL*, pages 263–274. ACM, 2013.
7. G. Castagna, M. Dezani-Ciancaglini, and L. Padovani. On global types and multi-party session. *Logical Methods in Computer Science*, 8(1), 2012.
8. W.-K. Chen, M. A. Hiltunen, and R. D. Schlichting. Constructing Adaptive Software in Distributed Systems. In *ICDCS*, volume 6084 of *LNCS*, pages 635–643. Springer, 2001.
9. M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive monitors for multiparty sessions. Submitted.
10. M. Dalla Preda, I. Lanese, J. Mauro, M. Gabbrielli, and S. Giallorenzo. Safe run-time adaptation of distributed applications. http://www.cs.unibo.it/projects/jolie/aioc.pdf.
11. http://www.eclipse.org/.
12. C. Ghezzi, M. Pradella, and G. Salvaneschi. An Evaluation of the Adaptation Capabilities in Programming Languages. In *SEAMS*, pages 50–59. ACM, 2011.
13. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *POPL*, pages 273–284. ACM Press, 2008.
14. http://www.jolie-lang.org/.
15. I. Jureta, S. Faulkner, and P. Thiran. Dynamic requirements specification for adaptable and open service-oriented systems. In *ICSOC*, volume 4749 of *LNCS*, pages 270–282. Springer, 2007.
16. I. Lanese, A. Bucchiarone, and F. Montesi. A Framework for Rule-Based Dynamic Adaptation. In *TGC*, volume 6084 of *LNCS*, pages 284–300. Springer, 2010.
17. I. Lanese, F. Montesi, and G. Zavattaro. Amending choreographies. In *WWV*, volume 123, pages 34–48. EPTCS, 2013.
18. L. A. F. Leite et al. A systematic literature review of service choreography adaptation. *Service Oriented Computing and Applications*, 2012.
19. F. Montesi, C. Guidi, and G. Zavattaro. Composing services with JOLIE. In *Proc. of ECOWS'07*, pages 13–22. IEEE Press, 2007.
20. R. Pawlak et al. JAC: an aspect-based distributed dynamic framework. *Softw., Pract. Exper.*, 34(12):1119–1148, 2004.
21. S. Rinderle, A. Wombacher, and M. Reichert. Evolution of process choreographies in dychor. In *OTM Conferences (1)*, volume 4275 of *LNCS*, pages 273–290. Springer, 2006.
22. http://www.jboss.org/scribble.
23. A. Wombacher. Alignment of choreography changes in BPEL processes. In *IEEE SCC*, pages 1–8. IEEE Press, 2009.
24. http://www.eclipse.org/Xtext/.
25. Z. Yang, B. H. C. Cheng, R. E. K. Stirewalt, J. Sowell, S. M. Sadjadi, and P. K. McKinley. An aspect-oriented approach to dynamic adaptation. In *WOSS*, pages 85–92. ACM, 2002.
26. J. Zhang, H. Goldsby, and B. H. C. Cheng. Modular Verification of Dynamically Adaptive Systems. In *AOSD*, pages 161–172. ACM, 2009.

A Proof of Theorem 1

In order to prove the bound on the complexity of the connectedness check we use the lemma below, showing that the checks to verify the connectedness for sequence for a single sequence operator can be performed in linear time on the size of the sets generated by transI and transF.

Lemma 1. *Given S, S' sets of multisets of two elements, checking if $\forall s \in S \,.\, \forall s' \in S' \,.\, s \cap s' \neq \emptyset$ can be done in $O(n)$ steps, where n is the maximum of $|S|$ and $|S'|$.*

Proof. W.l.o.g. we can assume that $|S| \leq |S'|$. If $|S| \leq 7$ then the check can be performed in $O(n)$ by comparing all the elements in S with all the elements in S'. If $|S| > 7$ then at least 4 distinct elements appear in the multisets in S since the maximum number of multisets with cardinality 2 obtained by 3 distinct elements is 6. In this case the following cases cover all the possibilities:

- there exist distinct elements a, b, c, d s.t. $\{a, b\}, \{a, c\}$, and $\{a, d\}$ belong to S. In this case for the check to succeed all the multisets in S' must contain a, otherwise the intersection of the multiset not containing a with one among the multisets $\{a, b\}, \{a, c\}$, and $\{a, d\}$ is empty. Similarly, since $|S'| > 7$, for the check to succeed all the multisets in S must contain a. Hence, if $\{a, b\}, \{a, c\}$, and $\{a, d\}$ belong to S then the check succeeds iff a belongs to all the multisets in S and in S'.
- there exist distinct elements a, b, c, d s.t. $\{a, b\}$ and $\{c, d\}$ belong to S. In this case the check succeeds only if S' is a subset of $\{\{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}\}$. Since $|S'| > 7$ the check can never succeed.
- there exist distinct elements a, b, c s.t. $\{a, a\}$ and $\{b, c\}$ belong to S. In this case the check succeeds only if S' is a subset of $\{\{a, b\}, \{a, c\}\}$. Since $|S'| > 7$ the check can never succeed.
- there exist distinct elements a, b s.t. $\{a, a\}$ and $\{b, b\}$ belong to S. In this case the check succeeds only if S' is a subset of $\{\{a, b\}\}$. Since $|S'| > 7$ the check can never succeed.

Summarizing, if $|S| > 7$ the check can succeed iff all the multisets in S and in S' share a common element. The existence of such an element can be verified in time $O(n)$.

Theorem 1 (Connectedness-check complexity). *The connectedness of an AIOC process $\mathcal{I}$ can be checked in time $O(n^2 \log(n))$, where n is the number of nodes in the abstract syntax tree of $\mathcal{I}$.*

Proof. To check the connectedness of $\mathcal{I}$ we first compute the values of the functions transI, transF, and op for each node of the abstract syntax tree (AST). We then check for each sequence operator whether connectedness for sequence holds and for each parallel operator whether connectedness for parallel holds.

The functions transI and transF associate to each node a set of pairs of roles. Assuming an implementation of the data set structure based on balanced trees (with pointers), transI and transF can be computed in constant time for interactions, assignments, $\mathbf{1}$, $\mathbf{0}$, and sequence constructs. For while and scope constructs computing $\mathrm{transF}(\mathcal{I}')$

requires the creation of balanced trees having an element for every role of $\mathcal{I}'$. Since the roles are $O(n)$, $\mathrm{transF}(\mathcal{I}')$ can be computed in $O(n\log(n))$. For parallel and if constructs a union of sets is needed. The union costs $O(n\log(n))$ since each set generated by transI and transF contains at maximum n elements.

The computation of op can be performed in $O(1)$ except for the parallel, sequence, and if operators, where the union of sets cost $O(n\log(n))$. Since the AST contains n nodes, the computation of the sets generated by transI, transF, and op can be performed in $O(n^2\log(n))$.

To check connectedness for sequence we have to verify that for each node $\mathcal{I}';\mathcal{I}''$ of the AST we have that $\forall r_1 \to r_2 \in \mathrm{transF}(\mathcal{I}'), \forall s_1 \to s_2 \in \mathrm{transI}(\mathcal{I}'')\ .\ \{r_1, r_2\} \cap \{s_1, s_2\} \neq \emptyset$. Since $\mathrm{transF}(\mathcal{I}')$ and $\mathrm{transI}(\mathcal{I}'')$ have $O(n)$ elements, thanks to Lemma 1, checking if $\mathcal{I}';\mathcal{I}''$ is connected for sequence costs $O(n)$. Since in the AST there are less then n sequence operators, the check of the connectedness for sequence for the entire AST costs $O(n^2)$.

To check connectedness for parallel we have to verify that for each node $\mathcal{I}' \parallel \mathcal{I}''$ of the AST we have that $\mathrm{op}(\mathcal{I}') \cap \mathrm{op}(\mathcal{I}'') = \emptyset$. Since $\mathrm{op}(\mathcal{I}')$ and $\mathrm{op}(\mathcal{I}'')$ have $O(n)$ elements, checking if their intersection is empty costs $O(n\log(n))$. Since in the AST there are less then n parallel operators the check of the connectedness for parallel for the entire AST costs $O(n^2\log(n))$.

The complexity of checking the connectedness of the entire AST is therefore limited by the cost of computing functions transI, transF, and op, and of checking the connectedness for parallel. All these activities have a complexity of $O(n^2\log(n))$.

B Proof of Theorem 2

This section contains the proof of our main result, Theorem 2, including various auxiliary definitions and lemmas.

The proof strategy consists in defining a notion of bisimilarity (Definition 22) which implies weak trace equivalence (Lemma 9) and then providing a suitable bisimulation relating each well-annotated connected AIOC system with its projection. The proof that this relation is indeed a bisimulation relies on the fact that the events in the AIOC (Definition 14) and in the APOC (Definition 17) are related (Lemma 2).

Observe that annotated AIOCs trivially inherit the semantics of AIOCs, since indexes are just decorations, with no effect on the behavior. Moreover, when applying *rule l where $\mathcal{C}$ specifies $\mathcal{I}$* one has to ensure that $\mathcal{I}$ is annotated with indexes never used before in the process.

Definition 12 (Events). *We use ε to range over events, and we write $[\varepsilon]_r$ to highlight that event ε is performed by role r. An annotated AIOC $\mathcal{I}$ contains the following events:*

Communication events: *a sending event $n : \overline{o^?}@r_2$ in role r_1 and a receiving event $n : o^?@r_1$ in role r_2 for each interaction $n : o^? : r_1(e) \to r_2(x)$; we also denote the sending event as f_n or $[f_n]_{r_1}$ and the receiving event as t_n or $[t_n]_{r_2}$;*

Assignment events: *an assignment event ε_n in role r for each assignment $n : x@r = e$;*

Scope events: *a scope initialization event $\uparrow_{l,n}$ and a scope termination event $\downarrow_{l,n}$ for each scope n : scope $l@r$ $\{\mathcal{I}\}$ prop $\{\Delta\}$. Both these events belong to all the roles in* $\mathrm{roles}(\mathcal{I})$*;*

If events: *a guard if-event ε_n in role r for each construct n : if $b@r$ $\{\mathcal{I}\}$ else $\{\mathcal{I}'\}$;*

While events: *a guard while-event ε_n in role r for each construct n : while $b@r$ $\{\mathcal{I}\}$.*

A sending and a receiving event with the same index n are called matching events. We denote with $\bar{\varepsilon}$ an event matching event ε.

Let $Events(\mathcal{I})$ denote the set of events of the annotated AIOC $\mathcal{I}$.

Definition 13 (Global index). *Given an annotated AIOC process $\mathcal{I}$, or an annotated APOC network $\mathcal{N}$ (defined later on), for each annotated construct with index n we define its global index ξ as follows:*

- *if the construct is not in the body of a while then $\xi = n$;*
- *if the innermost while construct that contains the considered construct has global index ξ' then the considered construct has global index $\xi = \xi' : n$;*

Definition 14 (AIOC dynamic events). *Let $DEvents(\mathcal{I})$ denote the set of dynamic events of $\mathcal{I}$. Given an annotated AIOC $\mathcal{I}$, the set $DEvents(\mathcal{I})$ is computed as in Definition 12 where global indexes replace indexes.*

In the following, we will refer to elements in $DEvents(\mathcal{I})$ simply as events. No misunderstanding may occur since from now on when considering AIOC we always speak about dynamic events.

The relation below roughly considers as events the actions in the AIOC, and defines a causality relation accordingly.

Definition 15 (AIOC Causality relation). *Let us consider an annotated AIOC $\mathcal{I}$. A causality relation $\leq_{AIOC} \subseteq DEvents(\mathcal{I}) \times DEvents(\mathcal{I})$ is a partial order among events in $\mathcal{I}$. We define $\leq_{AIOC}$ as the minimum partial order satisfying:*

Sequentiality: *Let $\mathcal{I}'; \mathcal{I}''$ be a sub term of AIOC $\mathcal{I}$. If ε' is an event in $\mathcal{I}'$ and ε'' is an event in $\mathcal{I}''$, then $\varepsilon' \leq_{AIOC} \varepsilon''$.*

Scope: *Let n : scope $l@r$ $\{\mathcal{I}'\}$ prop $\{\Delta\}$ be a sub term of AIOC $\mathcal{I}$. If ε' is an event in $\mathcal{I}'$ then $\uparrow_{l,\xi} \leq_{AIOC} \varepsilon' \leq_{AIOC} \downarrow_{l,\xi}$.*

Synchronization: *For each interaction the sending event precedes the receiving event.*

If: *Let n : if $b@r$ $\{\mathcal{I}\}$ else $\{\mathcal{I}'\}$ be a sub term of AIOC $\mathcal{I}$, let ε_n be the guard if-event in role r, then for every event ε in $\mathcal{I}$ and for every event ε' in $\mathcal{I}'$ we have $\varepsilon_n \leq_{AIOC} \varepsilon$ and $\varepsilon_n \leq_{AIOC} \varepsilon'$.*

While: *Let n : while $b@r$ $\{\mathcal{I}\}$ be a sub term of AIOC $\mathcal{I}$, let ε_n be the guard while-event in role r, then for every event ε in $\mathcal{I}'$ we have $\varepsilon_n \leq_{AIOC} \varepsilon$.*

We now introduce some auxiliary definitions to reason on APOC networks.

Definition 16 (Annotated APOC). *Annotated APOC networks are derived by a grammar obtained by adding unique indexes $n \in \mathbb{N}$ also to communications, assignments,*

while and if constructs; thus obtaining the following grammar:

$$
\begin{aligned}
P ::= {} & n : o^? : x \text{ from } r \mid n : o^? : e \text{ to } r \mid n : o^* : X \text{ to } r \mid \\
& P; P' \mid \ P \mid P' \mid n : x = e \mid \mathbf{1} \mid \mathbf{0} \mid \\
& n : \text{if } b \ \{P\} \text{ else } \{P'\} \mid n : \text{while } b \ \{P\} \mid \\
& n : \text{scope } l@r \ \{P\} \text{ prop } \{\Delta\} \text{ roles } \{S\} \mid \\
& n : \text{scope } l@r \ \{P\} \\
X ::= {} & \texttt{no} \mid P \\
\mathcal{N} ::= {} & (P, \Gamma)_r \mid \mathcal{N} \parallel \mathcal{N}'
\end{aligned}
$$

Annotated APOCs trivially inherit the semantics of APOCs, since indexes are just decorations, with no effect on the behavior. However, one has to clarify how indexes are generated for newly introduced processes. When applying *rule l where $\mathcal{C}$ specifies $\mathcal{I}$*, one has to generate indexes for constructs in the projection of $\mathcal{I}$ which are distinct, and not used in the target system. Similarly, one should assign to the auxiliary communications introduced by rules LEAD-ADAPT and LEAD-NOADAPT indexes never used elsewhere. Rule ADAPT instead should use the index of the corresponding communication introduced by rule LEAD-ADAPT. Note that the unfolding of the while leaves the indexes unchanged. This explains the use of global indexes: after the unfolding indexes are not unique any more, while global indexes are. Also, one can easily extend the projection to a function from annotated AIOC processes to annotated APOC networks, requiring that the input and output actions obtained by projecting interaction with index n have both index n.

Definition 17 (APOC events). *An annotated APOC network $\mathcal{N}$ contains the following events:*

Communication events: *a sending event $\xi : \overline{o^?}@r_2$ in role r_1 for each output $n : o^? : e$ to r_2 with global index ξ in role r_1; and a receiving event $\xi : o^?@r_1$ in role r_2 for each input $n : o^? : e$ from r_1 with global index ξ in role r_2; we will also denote the sending event as f_ξ or $[f_\xi]_{r_1}$; and the receiving event as t_ξ or $[t_\xi]_{r_2}$.*

Assignment events: *an assignment event ε_ξ in role r for each assignment $n : x@r = e$ with global index ξ;*

Scope events: *a scope initialization event $\uparrow_{l,\xi}$ and a scope termination event $\downarrow_{l,\xi}$ for each n : scope $l@r \ \{P\}$ prop $\{\Delta\}$ roles $\{S\}$ or n : scope $l@r \ \{P\}$ with global index ξ. Scope events with the same name coincide, and thus may belong to different roles;*

If events: *a guard if-event ε_ξ in role r for each construct n : if $b@r \ \{P\}$ else $\{P'\}$ with global index ξ;*

While events: *a guard while-event ε_ξ in role r for each construct n : while $b@r \ \{P\}$ with global index ξ.*

We denote with $Events(\mathcal{N})$ the set of events of the network $\mathcal{N}$. A sending and a receiving event with the same global index ξ are called matching events. We denote with $\bar{\varepsilon}$ an event matching event ε. A communication event is either a sending event or a receiving event. A communication event is unmatched if there is no event matching it.

Definition 18 (APOC causality relation). *Let us consider an annotated APOC network $\mathcal{N}$. A causality relation $\leq_{APOC} \subseteq Events(\mathcal{N}) \times Events(\mathcal{N})$ is a partial order among events in $\mathcal{N}$. We define $\leq$ as the minimum partial order satisfying:*

Sequentiality: *Let $P'; P''$ be a sub term of APOC network $\mathcal{N}$. If ε' is an event in P' and ε'' is an event in P'', both in the same role r, then $\varepsilon' \leq_{APOC} \varepsilon''$.*

Scope-coordinator: *Let n : scope $l@r$ $\{P\}$ prop $\{\Delta\}$ roles $\{S\}$ be a sub term of APOC $\mathcal{N}$ in role r with global index ξ. If ε' is an event in P then $\uparrow_{l,\xi} \leq_{APOC} \varepsilon' \leq_{APOC} \downarrow_{l,\xi}$.*

Scope-simple: *Let n : scope $l@r$ $\{P\}$ be a sub term of APOC $\mathcal{N}$ in role r' with global index ξ. If ε' is an event in P then $\uparrow_{l,\xi} \leq_{APOC} \varepsilon' \leq_{APOC} \downarrow_{l,\xi}$.*

Synchronization: *For each pair of events ε and ε', $\varepsilon \leq \varepsilon'$ implies $\bar{\varepsilon} \leq_{APOC} \varepsilon'$.*

If: *Let n : if b $\{P\}$ else $\{P'\}$ be a sub term of APOC network $\mathcal{N}$ with global index ξ, let ε_ξ be the guard if-event in role r, then for every event ε in P and for every event ε' in P' we have $\varepsilon_\xi \leq_{APOC} \varepsilon$ and $\varepsilon_\xi \leq_{APOC} \varepsilon'$.*

While: *Let n : while b $\{P\}$ be a sub term of APOC network $\mathcal{N}$ with global index ξ, let ε_ξ be the guard while-event in role r, then for every event ε in P we have $\varepsilon_\xi \leq_{APOC} \varepsilon$.*

Lemma 2. *Given an AIOC process $\mathcal{I}$, for each state Σ the APOC network* $\mathrm{proj}(\mathcal{I}, \Sigma)$ *obtained by projecting it is such that:*

1. $DEvents(\mathcal{I}) \subseteq Events(\mathrm{proj}(\mathcal{I}, \Sigma))$;
2. $\varepsilon_1 \leq_{AIOC} \varepsilon_2 \Rightarrow \varepsilon_1 \leq_{APOC} \varepsilon_2 \vee \varepsilon_1 \leq_{APOC} \bar{\varepsilon_2}$

Proof. 1. By definition of projection.

2. Let $\varepsilon_1 \leq_{AIOC} \varepsilon_2$ then:

Sequentiality: Consider $\mathcal{I} = \mathcal{I}'; \mathcal{I}''$. If events are in the same role the implication follows from the sequentiality of the $\leq_{APOC}$.

Let us show that there exist an event ε'' in an initial interaction of $\mathcal{I}''$ such that either $\varepsilon'' \leq_{APOC} \varepsilon_2$ or $\varepsilon'' \leq_{APOC} \bar{\varepsilon_2}$. The proof is by induction on the structure of $\mathcal{I}''$. The only difficult case is sequential composition. Suppose $\mathcal{I}'' = \mathcal{I}_1; \mathcal{I}_2$. If $\varepsilon_2 \in DEvents(\mathcal{I}_1)$ the thesis follows from inductive hypothesis. If $\varepsilon_2 \in DEvents(\mathcal{I}_2)$ then by induction there exists an event ε_3 in an initial interaction of $\mathcal{I}_2$ s.t. $\varepsilon_3 \leq_{APOC} \varepsilon_2$ or $\varepsilon_3 \leq_{APOC} \bar{\varepsilon_2}$. By synchronization (Definition 18) we have that $\bar{\varepsilon_3} \leq_{APOC} \varepsilon_2$ or $\bar{\varepsilon_3} \leq_{APOC} \bar{\varepsilon_2}$. By connectedness for sequence we have that ε_3 or $\bar{\varepsilon_3}$ are in the same role of en event ε_4 in $\mathcal{I}'$. By sequentiality (Definition 18) we have that $\varepsilon_4 \leq_{APOC} \varepsilon_3$ or $\varepsilon_4 \leq_{APOC} \bar{\varepsilon_3}$. By synchronization we have that $\bar{\varepsilon_4} \leq_{APOC} \varepsilon_3$ or $\bar{\varepsilon_4} \leq_{APOC} \bar{\varepsilon_3}$. The thesis follows from the inductive hypothesis on ε_4 and by transitivity of $\leq_{APOC}$.

Let us also show that there exists a final event $\varepsilon''' \in DEvents(\mathcal{I}')$ such that $\varepsilon_1 \leq_{APOC} \varepsilon'''$ or $\varepsilon_1 \leq_{APOC} \bar{\varepsilon'''}$. The proof is by induction on the structure of $\mathcal{I}'$. The only difficult case is sequential composition. Suppose $\mathcal{I}' = \mathcal{I}_1; \mathcal{I}_2$ If $\varepsilon_1 \in DEvents(\mathcal{I}_2)$ the thesis follows from inductive hypothesis. If $\varepsilon_1 \in DEvents(\mathcal{I}_1)$ then the proof is similar to the one above, finding a final event in $\mathcal{I}_1$ and applying sequentiality, synchronization, and transitivity.

The thesis follows from the two results above again by sequentiality, synchronization, and transitivity.

Scope: it means that either (1) $\varepsilon_1 = \uparrow_{l,n}$ and ε_2 is an event in the scope or (2) $\varepsilon_1 = \uparrow_{l,n}$ and $\varepsilon_2 = \downarrow_{l,n}$, or (3) ε_1 is an event in the scope and $\varepsilon_2 = \downarrow_{l,n}$. We consider the first case since the third one is analogous and the second one follows by transitivity. If ε_2 is in the coordinator then the thesis follows easily. Otherwise it follows thanks to the auxiliary synchronizations with a reasoning similar to the one for sequentiality.

Synchronization: it means that ε_1 is a sending event and ε_2 is the corresponding receiving event, namely $\varepsilon_1 = \overline{\varepsilon_2}$. Thus, since $\varepsilon_2 \leq_{APOC} \varepsilon_2$ then $\overline{\varepsilon_2} \leq_{APOC} \varepsilon_2$.

If: it means that ε_1 is the evaluation of the guard and ε_2 is in one of the two branches. Thus, if ε_2 is in the coordinator then the thesis follows easily. Otherwise it follows thanks to the auxiliary synchronizations with a reasoning similar to the one for sequentiality.

While: it means that ε_1 is the evaluation of the guard and ε_2 is in the body of the while. Thus, if ε_2 is in the coordinator then the thesis follows easily. Otherwise it follows thanks to the auxiliary synchronizations with a reasoning similar to the one for sequentiality.

Definition 19 (Conflicting events). *Given an AIOC process $\mathcal{I}$ we say that two events $e, e' \in DEvents(\mathcal{I})$ are conflicting if they belong to different branches of the same if construct, i.e. there exists a subprocess if b $\{\mathcal{I}'\}$ else $\{\mathcal{I}''\}$ of $\mathcal{I}$ such that $e \in DEvents(\mathcal{I}') \wedge e' \in DEvents(\mathcal{I}'')$ or $e' \in DEvents(\mathcal{I}') \wedge e \in DEvents(\mathcal{I}'')$.*

Similarly, given an APOC network $\mathcal{N}$, we say that two events $e, e' \in Events(\mathcal{N})$ are conflicting if they belong to different branches of the same if construct, i.e. there exists a subprocess if b $\{P\}$ else $\{P'\}$ of $\mathcal{N}$ such that $e \in Events(P) \wedge e' \in Events(P')$ or $e' \in Events(P) \wedge e \in Events(P')$.

Definition 20 (Well-annotated APOC). *An annotated APOC network $\mathcal{N}$ is well-annotated for its causality relation $\leq_{APOC}$ if the following conditions hold:*

C1 *for each global index ξ there are at most two communication events with global index ξ and, in this case, they are matching events;*

C2 *if $\varepsilon_1 \leq_{APOC} \varepsilon_2$ then ε_2 can become enabled only after ε_1 has been executed or discarded;*

C3 *for each pair of non-conflicting sending events $[f_\xi]_r$ and $[f_{\xi'}]_r$ on the same operation $o^?$ with the same target s such that $\xi \neq \xi'$ we have $[f_\xi]_r \leq_{APOC} [f_{\xi'}]_r$ or $[f_{\xi'}]_r \leq_{APOC} [f_\xi]_r$;*

C4 *for each pair of non-conflicting receiving events $[t_\xi]_s$ and $[t_{\xi'}]_s$ on the same operation $o^?$ with the same sender r such that $\xi \neq \xi'$ we have $[t_\xi]_s \leq [t_{\xi'}]_s$ or $[t_{\xi'}]_s \leq [t_\xi]_s$;*

C5 *if ε is an event inside a scope with name l and global index ξ then its matching event $\overline{\varepsilon}$ (if it exists) is inside a scope with the same name and global index.*

C6 *if two events have the same index but different global indexes then one of them is inside a while with global index ξ_1, let us call it ε_1, and the other, ε_2, is not. Furthermore, $\varepsilon_2 \leq_{APOC} \varepsilon_{\xi_1}$ where ε_{ξ_1} is the guarding while-event of the while with global index ξ_1.*

Adaptation, conditional choice and iteration at the AIOC level happen in one step, while they correspond to many steps of the projected APOC. Thus, we define the function upd that bridges this gap. Function upd applies the missing APOC steps to complete already started AIOC actions.

Definition 21 (upd **function).** *Let $\mathcal{N}$ be an annotated APOC. The* upd *function is defined as the composition of a function* prop *that propagates decisions from the coordinator to other roles, and of a function* sim *that eliminates all the auxiliary closing communications. Thus,* $\mathrm{upd}(\mathcal{N}) = \mathrm{sim}(\mathrm{prop}(\mathcal{N}))$. *More specifically,* $\mathrm{prop}(\mathcal{N})$ *is obtained from $\mathcal{N}$ by repeating the following operation while possible:*

1. *for each* $n : o^*_m : \mathit{true}$ *to* r' *enabled, replace every* $o^*_n : x_n$ *from* r; *while* $x_n\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^*_n : x_n$ *from* $r\}$ *not inside another while construct, with* $P; o^*_n : \mathtt{ok}$ *to* $r; o^*_n : x_n$ *from* r; *while* $x_n\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^*_n : x_n$ *from* $r\}$*; and replace* $n : o^*_m : \mathit{true}$ *to* r' *with* $\mathbf{1}$.
2. *for each* $n : o^*_m : \mathit{false}$ *to* r' *enabled, replace every* $o^* : x_m$ *from* r; *while* $x_m\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^* : x_m$ *from* $r\}$ *not inside another while construct, with* $\mathbf{1}$*; and replace* $n : o^*_m : \mathit{false}$ *to* r' *with* $\mathbf{1}$.
3. *for each while* $x_m\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^* : x_m$ *from* $r\}$ *enabled not inside another while construct, such that* x_m *evaluates to* `true` *in the local state, replace it with* $P; o^*_n : \mathtt{ok}$ *to* $r; o^*_n : x_n$ *from* r; *while* $x_n\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^*_n : x_n$ *from* $r\}$.
4. *for each while* $x_m\ \{P; o^*_n : \mathtt{ok}$ *to* $r; o^* : x_m$ *from* $r\}$ *enabled not inside another while construct, such that* x_m *evaluates to* `false` *in the local state, replace it with* $\mathbf{1}$.
5. *for each* $n : o^*_m : \mathit{true}$ *to* r' *enabled, replace every* $o^*_m : x_m$ *from* $r; m :$ *if* $x_m\ \{P'\}$ *else* $\{P''\}$ *not inside a while construct, with* P'*; and replace* $n : o^*_m : \mathit{true}$ *to* r' *with* $\mathbf{1}$.
6. *for each* $n : o^*_m : \mathit{false}$ *to* r' *enabled, replace every* $o^*_m : x_m$ *from* $r; m :$ *if* $x_m\ \{P'\}$ *else* $\{P''\}$ *not inside a while construct, with* P''*; and replace* $n : o^*_m : \mathit{false}$ *to* r' *with* $\mathbf{1}$.
7. *for each* $m :$ *if* $x_m\ \{P'\}$ *else* $\{P''\}$ *enabled such that* x_m *evaluates to* `true` *in the local state, replace it with* P'.
8. *for each* $m :$ *if* $x_m\ \{P'\}$ *else* $\{P''\}$ *enabled such that* x_m *evaluates to* `false` *in the local state, replace it with* P''.
9. *for each* $n : o^*_{l,m} : P$ *to* s *enabled, replace every* $m :$ *scope* $l@r\ \{P'\}$ *in role* s *not inside a while construct, with* P*, and replace* $n : o^*_{l,m} : P$ *to* s *with* $\mathbf{1}$.
10. *for each* $n : o^*_{l,m} : \mathtt{no}$ *to* s *enabled, replace every* $m :$ *scope* $l@r\ \{P'\}$ *in the role* s *not inside a while construct, with* P' *and replace* $n : o^*_{l,m} : P$ *to* s *with* $\mathbf{1}$.

$\mathrm{sim}(\mathcal{N})$ *is obtained from $\mathcal{N}$ by repeating the following operation while possible:*

- *replace each* $n : o^*_n : \mathtt{ok}$ *to* r, $n : o^*_{l,m} : \mathtt{ok}$ *to* r, $n : o^*_n : \_$ *from* r *or* $n : o^*_{l,m} : \_$ *from* r *not inside a while construct with* $\mathbf{1}$.
- *replace each operation occurrence of the form* $n \cdot o^?$ *with* $o^?$.

Furthermore sim *may apply 0 or more times the following operation:*

- *replace a sub term* $\mathbf{1}; P$ *by* P *or a sub term* $\mathbf{1} \mid P$ *by* P.

We denote with $\mathrm{upd}(\mathcal{N})$ *the annotated APOC obtained from* $\mathcal{N}$.

The result below proves that in a well-annotated APOC only actions corresponding to events minimal w.r.t. the causality relation $\leq_{APOC}$ may be enabled.

Lemma 3. *If* $\mathcal{N}$ *is an APOC,* $\leq_{APOC}$ *its causality relation and* ε *is an event corresponding to an action enabled in* $\mathcal{N}$ *then* ε *is minimal w.r.t.* $\leq_{APOC}$.

Proof. The proof is by contradiction. Suppose ε is enabled but not minimal, i.e. there exists ε' such that $\varepsilon' \leq_{APOC} \varepsilon$. If there is more than one such ε' consider the one such that the length of the derivation of $\varepsilon' \leq_{APOC} \varepsilon$ is minimal. This derivation should have length one, and following Definition 18 it may result from one of the following cases:

- Sequentiality: $\varepsilon' \leq_{APOC} \varepsilon$ means that $\varepsilon' \in \mathit{Events}(P')$, $\varepsilon \in \mathit{Events}(P'')$, and $P'; P''$ is a sub term of $\mathcal{N}$. Because of the semantics of sequential composition ε cannot be enabled.
- Scope: let $n : \mathit{scope}\ l@r\ \{P\}\ \mathit{prop}\ \{\Delta\}\ \mathit{roles}\ \{S\}$ or $n : \mathit{scope}\ l@r\ \{P\}$ be a subprocess of $\mathcal{N}$ with global index ξ. We have the following cases:
  - $\varepsilon' = \uparrow_{l,\xi}$ and $\varepsilon \in \mathit{Events}(P)$, and this implies that ε cannot be enabled since if ε' is enabled then the rules ADAPT or NO-ADAPT for the evolution of the scope have not been applied yet;
  - $\varepsilon' = \uparrow_{l,\xi}$ and $\varepsilon = \downarrow_{l,\xi}$, and this implies that ε cannot be enabled since the scope cannot be finalized before it has started;
  - $\varepsilon' \in \mathit{Events}(P)$ and $\varepsilon = \downarrow_{l,\xi}$, this is impossible since if ε' is enabled there is no event ε because the events $\uparrow_{l,\xi}$ and $\downarrow_{l,\xi}$ disappear as soon as the rule LEAD-ADAPT or LEAD-NOADAPT are performed.
- If: $\varepsilon \leq_{APOC} \varepsilon'$ means that ε is the evaluation of the condition of the sub term $n : \mathit{if}\ x_n\ \{P'\}\ \mathit{else}\ \{P''\}$ and $\varepsilon' \in \mathit{Events}(P') \cup \mathit{Events}(P'')$. Event ε' cannot be enabled because of the semantics of if.
- While: $\varepsilon \leq_{APOC} \varepsilon'$ means that ε is the evaluation of the condition of the sub term $n : \mathit{while}\ x_n\ \{P\}$ and $\varepsilon' \in \mathit{Events}(P)$. Event ε' cannot be enabled because of the semantics of while.

The following result shows that if an interaction is performed the two executed events are matching events.

Lemma 4. *If* $\mathcal{N}$ *is a well-annotated APOC and* $\mathcal{N} \xrightarrow{o^?:r_1(v)\to r_2(x)} \mathcal{N}'$ *then the two executed events are matching events.*

Proof. By definition of APOC semantics we have that the transition $\mathcal{N} \xrightarrow{o^?:r_1(v)\to r_2(x)} \mathcal{N}'$ can be generated either by the SYNCH rule or by the SYNCH-ADAPT rule. In both the cases we have that the two events are on the same operation and that r_2 is the target of the first event. Assume that they are not matching events. Then for the definition of well-annotated APOC they are either conflicting or in the causality relation. In the first case none of them can be enabled by Definition 18 since they are inside an *if* construct. In the second case thanks to Lemma 3, at least one of them cannot be enabled since it is not minimal. This is absurd, thus they have to be matching events.

We now prove that all the projections of connected well-annotated AIOCs are well-annotated APOCs.

Lemma 5. *Let $\mathcal{I}$ be a well-annotated connected AIOC process, and Σ a state. Then its projection $\mathcal{N} = \mathrm{proj}(\mathcal{I}, \Sigma)$ is a well-annotated APOC network w.r.t. $\leq_{APOC}$.*

Proof. We have to prove that $\mathrm{proj}(\mathcal{I}, \Sigma)$ satisfies the conditions of Definition 20 of well-annotated APOC:

C1 For each global index ξ there are at most two communication events with global index ξ and, in this case, they are matching events. The condition follows by the definition of the projection function and the fact that, on projections of well-annotated AIOCs, global indexes are in a bijection with indexes.

C2 If $\varepsilon_1 \leq_{APOC} \varepsilon_2$ then ε_2 can become enabled only after ε_1 has been executed or discarded. This condition follows from Lemma 3.

C3 For each pair of non-conflicting sending events $[f_\xi]_r$ and $[f_{\xi'}]_r$ on the same operation $o^?$ and with the same target such that $\xi \neq \xi'$ we have $[f_\xi]_r \leq_{APOC} [f_{\xi'}]_r$ or $[f_{\xi'}]_r \leq_{APOC} [f_\xi]_r$. Note that the two events are in the same role, thus w.l.o.g. we can assume that they there exist two processes P, P' such that $[f_\xi]_r \in \mathit{Events}(P)$ and $[f_{\xi'}]_r \in \mathit{Events}(P')$ and that one among $P; P'$, $P|P'$, and *if* b $\{P\}$ *else* $\{P'\}$ is a subprocess of $\mathcal{N}$.
Since $\mathcal{I}$ is connected for parallel, by Definition 9 and by definition of the projection function the second case can never happen. Similarly, since the events are non-conflicting by Definition 19 the third case can never happen. If $P; P'$ is a subprocess of $\mathcal{N}$ then by sequentiality (Definition 18) we have the thesis.

C4 Similar to the previous case.

C5 By definition of the projection function.

C6 Trivial, since by definition of well-annotated AIOC it never happens that there are two events with the same index and different global indexes.

The next lemma shows that for every environment E and for every set of rules $\mathbf{R}$ the APOC $\mathcal{N}$ and $\mathrm{upd}(\mathcal{N})$ have the same set of weak traces.

Lemma 6. *Let $\mathcal{N}$ be an APOC. The following properties hold:*

1. *if* $\mathrm{upd}(\mathcal{N}) \xrightarrow{\alpha} \mathcal{N}'$ *then* $\mathcal{N} \xrightarrow{\chi_1} \ldots \xrightarrow{\chi_k}\xrightarrow{\alpha} \mathcal{N}''$ *where* $\chi_i \in \{o^* : r_1(v) \to r_2(x), \tau\}$ *and* $\mathrm{upd}(\mathcal{N}'') = \mathrm{upd}(\mathcal{N}')$.
2. *if* $\mathcal{N} \xrightarrow{\chi} \mathcal{N}'$ *for* $\chi \in \{o^? : r_1(v) \to r_2(x); \surd; E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}; E, \mathbf{R}, [\texttt{no-adapt}], \tau\}$*, then we have that one of the following holds: (A)* $\mathrm{upd}(\mathcal{N}) \xrightarrow{\chi} \mathcal{N}''$ *such that* $\mathrm{upd}(\mathcal{N}') = \mathrm{upd}(\mathcal{N}'')$*, or (B)* $\mathrm{upd}(\mathcal{N}) = \mathrm{upd}(\mathcal{N}')$ *and* $\chi \in \{o^* : r_1(v) \to r_2(x), \tau\}$*;*

Proof. 1. The upd function corresponds to perform weak actions, namely actions with labels in $\{o^* : r_1(v) \to r_2(x), \tau\}$. $\mathcal{N}$ may perform the enabled weak actions that correspond to the application of upd reducing to $\mathcal{N}'''$. Then, α is enabled also in $\mathcal{N}'''$ and we have $\mathcal{N}''' \xrightarrow{\alpha} \mathcal{N}''$. At this point we have that $\mathcal{N}''$ and $\mathcal{N}'$ may differ only for weak actions removed by upd.

2. Either the transition with label χ corresponds to one of the transitions executed by function upd or not. In the first case statement (B) holds trivially. Otherwise transition labeled by χ is still enabled in $\mathrm{upd}(\mathcal{N})$ and the thesis follows.

We now prove a few properties of transitions with label $\surd$.

Lemma 7. *If $\mathcal{I} \xrightarrow{\surd}$ then, for each role $s \in \mathrm{roles}(\mathcal{I})$, $\pi(\mathcal{I}, s) \xrightarrow{\surd}$ and vice versa.*

Proof. By structural induction on $\mathcal{I}$.

The next lemma shows that if two matching events are enabled in the projection of an AIOC, then the corresponding interaction is enabled in the AIOC.

Lemma 8. *Let $\mathcal{I}$ be an AIOC derived from a well-annotated connected AIOC and $n : o^? : r_1(e) \to r_2(x)$ be an interaction in $\mathcal{I}$. If $n : o^? : e$ to r_1 and $n : o^? : x$ from r_2 are matching events and are both enabled in $\mathrm{proj}(\mathcal{I}, \Sigma)$ then $n : o^? : r_1(e) \to r_2(x)$ is enabled.*

Proof. If $\mathcal{I}$ is well-annotated and connected then the proof is by structural induction on $\mathcal{I}$. The cases for $\mathbf{1}$, $\mathbf{0}$, scopes if and while constructs are trivial. For parallel composition just consider that since the two events have the same global index then they are from the same component, and the thesis follows by inductive hypothesis. Let us consider sequential composition. Suppose $\mathcal{I} = \mathcal{I}'; \mathcal{I}''$. If $n : o^? : r_1(e) \to r_2(x) \in \mathcal{I}'$ then the thesis follows by inductive hypothesis. Otherwise by inductive hypothesis $n : o^? : r_1(e) \to r_2(x)$ is enabled in $\mathcal{I}''$. Thus, $r_1 \to r_2 \in \mathrm{transI}(\mathcal{I}'')$. From connectedness for sequence $\forall s_1 \to s_2 \in \mathrm{transF}(\mathcal{I}')$ then $\{r_1, r_2\} \cap \{s_1, s_2\} \neq \emptyset$. This is not possible since otherwise at least one of the events $n : o^? : e$ *to* r_1 and $n : o^? : x$ *from* r_2 would not be enabled. Thus, the only possibility is $\mathrm{transF}(\mathcal{I}') = \emptyset$. This implies that $\mathcal{I}' \xrightarrow{\surd}$. Thus, $n : o^? : r_1(e) \to r_2(x)$ is enabled in $\mathcal{I}$.

If $\mathcal{I}$ is not well-annotated and connected, then note that the only reason why it may be not well-annotated is that some auxiliary interactions are missing. However, this may at most enable more interactions, thus the thesis follows.

Definition 22 (Weak System Bisimilarity). *A weak system bisimulation is a relation R between AIOC systems and APOC systems such that if $(\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle, \langle E', \mathbf{R}', \mathcal{N} \rangle) \in R$ then:*

- *if $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}'' \rangle$ then $\langle E', \mathbf{R}', \mathcal{N} \rangle \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_k} \xrightarrow{\alpha'} \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle$ with $\forall i \in [1..k], \alpha_i \in \{o^* : r_1(v) \to r_2(x), \tau\}$ and $(\langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}'' \rangle, \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle) \in R$ and $\alpha' = \alpha$ or $\alpha' = n \cdot o^? : r_1(v) \to r_2(x)$ and $\alpha = o^? : r_1(v) \to r_2(x)$;*

- *if $\langle E', \mathbf{R}', \mathcal{N} \rangle \xrightarrow{\alpha'} \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle$ then one of the following two holds:*

  - *$\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}, \mathcal{I}'' \rangle$, with $\alpha' = \alpha$ or $\alpha' = n \cdot o^? : r_1(v) \to r_2(x)$ and $\alpha = o^? : r_1(v) \to r_2(x)$ and it holds that $(\langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}'' \rangle, \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle) \in R$;*

- $\alpha' \in \{o^* : r_1(v) \to r_2(x), o^* : r_1(X) \to r_2(\_), \tau\}$ *and it holds that* $(\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle, \langle E''', \mathbf{R}''', \mathcal{N}'' \rangle) \in R$

Weak system bisimilarity $\sim$ *is the largest weak system bisimulation.*

The following result states that weak system bisimilarity implies weak trace equivalence.

Lemma 9. *Let* $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ *be an AIOC system and* $\langle E', \mathbf{R}', \mathcal{N} \rangle$ *be an APOC system. If* $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \sim \langle E', \mathbf{R}', \mathcal{N} \rangle$ *then the AIOC system* $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ *and the APOC system* $\langle E', \mathbf{R}', \mathcal{N} \rangle$ *are weak trace equivalent.*

Proof. Easy, by coinduction.

We can now prove our main theorem that states that given a connected well-annotated AIOC process and a state Σ the APOC network obtained by its projection behaves as expected.

Theorem 2. *For each initial, connected AIOC process* $\mathcal{I}$*, each state* Σ*, each environment* E*, each set of initial, connected rules* $\mathbf{R}$ *the AIOC system* $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle$ *and the APOC system* $\langle E, \mathbf{R}, \mathrm{proj}(\mathcal{I}, \Sigma) \rangle$ *are weak trace equivalent, provided that all the rules in the sets of rules generated by the external updates are initial and connected.*

Proof. We have to prove that the relation R below is a weak system bisimulation.

$$R = \left\{ (\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle, \langle E, \mathbf{R}, \mathcal{N} \rangle) \;\middle|\; \begin{array}{l} \mathit{DEvents}(\mathcal{I}) \subseteq \mathit{Events}(\mathrm{prop}(\mathcal{N})), \mathrm{upd}(\mathcal{N}) = \mathrm{proj}(\mathcal{I}, \Sigma) \\ \forall \varepsilon_1, \varepsilon_2 \in \mathit{DEvents}(\mathcal{I}). \\ \varepsilon_1 \leq_{AIOC} \varepsilon_2 \Rightarrow \varepsilon_1 \leq_{APOC} \varepsilon_2 \vee \varepsilon_1 \leq_{APOC} \bar{\varepsilon_2}, \end{array} \right\}$$

where $\mathcal{I}$ is derived from a well-annotated connected AIOC and $\mathrm{upd}(\mathcal{N})$ is a well-annotated APOC.

Let us show that $(\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle, \langle E, \mathbf{R}, \mathrm{proj}(\mathcal{I}, \Sigma) \rangle)) \in R$. Note that for each $\mathcal{I}$ we have $\mathrm{upd}(\mathrm{proj}(\mathcal{I}, \Sigma)) = \mathrm{proj}(\mathcal{I}, \Sigma)$. From Lemma 5 $\mathrm{proj}(\mathcal{I}, \Sigma)$ is well annotated, thus $\mathrm{upd}(\mathrm{proj}(\mathcal{I}, \Sigma))$ is well annotated.

Observe that prop is the identity on $\mathrm{proj}(\mathcal{I}, \Sigma)$, thus from Lemma 2 we have that the conditions on the events are satisfied.

We proceed by proving that R is a weak system bisimulation that, from Lemma 9, implies weak trace equivalence.

The proof is by structural induction on the AIOC $\mathcal{I}$. All the sub terms of a connected AIOC are connected, thus the induction can be performed. We consider both challenges from the AIOC ($\to$) and from the APOC ($\leftarrow$).

To prove that R is a weak system bisimulation it is enough to prove that for each $(\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle, \langle E, \mathbf{R}, \mathcal{N} \rangle)$ where $\mathcal{N} = \mathrm{proj}(\mathcal{I}, \Sigma)$ then:

– if $\langle \Sigma, E, \mathbf{R}, \mathcal{I} \rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}'' \rangle$ then $\langle E, \mathbf{R}, \mathcal{N} \rangle \xrightarrow{\alpha'} \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle$ with $(\langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}'' \rangle, \langle E''', \mathbf{R}''', \mathcal{N}''' \rangle) \in R$ and $\alpha' = \alpha$ or $\alpha' = n \cdot o^? : r_1(v) \to r_2(x)$ and $\alpha = o^? : r_1(v) \to r_2(x)$;

– if $\langle E, \mathbf{R}, \mathcal{N}\rangle \xrightarrow{\alpha'} \langle E''', \mathbf{R}''', \mathcal{N}'''\rangle$ then $\langle \Sigma, E, \mathbf{R}, \mathcal{I}\rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}, \mathcal{I}''\rangle$ and $(\langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}''\rangle, \langle E''', \mathbf{R}''', \mathcal{N}'''\rangle) \in R$ and $\alpha' = \alpha$ or $\alpha' = n \cdot o^? : r_1(v) \to r_2(x)$ and $\alpha = o^? : r_1(v) \to r_2(x)$.

In fact, consider $\mathcal{N}$ with $\mathrm{upd}(\mathcal{N}) = \mathrm{proj}(\mathcal{I}, \Sigma)$. If $\langle \Sigma, E, \mathbf{R}, \mathcal{I}\rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}''\rangle$, then by hypothesis $\mathrm{upd}(\mathcal{N}) \xrightarrow{\alpha} \mathcal{N}''$. The thesis follows from Lemma 6 (case one). If instead $\langle E, \mathbf{R}, \mathcal{N}\rangle \xrightarrow{\alpha} \langle E''', \mathbf{R}''', \mathcal{N}'''\rangle$ then thanks to Lemma 6 we have one of the following: (A) $\mathrm{upd}(\mathcal{N}) \xrightarrow{\chi} \mathcal{N}''$ such that $\mathrm{upd}(\mathcal{N}''') = \mathrm{upd}(\mathcal{N}'')$, or (B) $\mathrm{upd}(\mathcal{N}) = \mathrm{upd}(\mathcal{N}')$ and $\chi \in \{o^* : r_1(v) \to r_2(x), o^* : r_1(X) \to r_2(\_), \tau\}$. In case (A) $\langle E, \mathbf{R}, \mathrm{upd}(\mathcal{N})\rangle \xrightarrow{\alpha} \langle E'', \mathbf{R}'', \mathcal{N}''\rangle$. Then we have $\langle \Sigma, E, \mathbf{R}, \mathcal{I}\rangle \xrightarrow{\alpha} \langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}''\rangle$ and $(\langle \Sigma'', E'', \mathbf{R}'', \mathcal{I}''\rangle, \langle E'', \mathbf{R}'', \mathcal{N}''\rangle) \in R$. The thesis follows since $\mathrm{upd}(\mathcal{N}''') = \mathrm{upd}(\mathcal{N}'')$. In case (B) the step is matched by the AIOC by staying idle, following the second option in the definition of weak system bisimilarity.

The case for label $\surd$ follows from Lemma 7. The case for labels $\Sigma, \mathbf{R}$ is trivial. Let us consider the other labels. For shortening the notation we consider only AIOC processes and APOC networks, since they are responsible of most of the system actions. Note, however, that the label for assignments in AIOC processes is $[v/x, r]$, while the label is τ in APOC networks. The mismatch is solved at the system level, where both labels become τ.

Case 1, 0 and $n : x@r = e$**:** trivial;

Case $n : o^? : r_1(e) \to r_2(x)$**:** trivial, by noting that the possible mismatch on the name of operations, namely between $n \cdot o^?$ and $o^?$, is solved by the Definition of weak system bisimilarity.

Case $\mathcal{I}; \mathcal{I}'$**:** from the definition of the projection function we have that $\mathcal{N} = \|_{r \in \mathrm{roles}(\mathcal{I}; \mathcal{I}')} (\pi(\mathcal{I}, r); \pi(\mathcal{I}', r), \Sigma_r)_r$.

$\to$ Suppose that $\mathcal{I}; \mathcal{I}' \xrightarrow{\alpha} \mathcal{I}''$ with $\alpha \in \{o^? : r_1(v) \to r_2(x), E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}, E, \mathbf{R}, [\texttt{no-adapt}], [v/x, r], \tau\}$. There are two possibilities: either $\mathcal{I} \xrightarrow{\alpha} \mathcal{I}'''$ and $\mathcal{I}'' = \mathcal{I}'''; \mathcal{I}'$ or $\mathcal{I} \xrightarrow{\surd}$ and $\mathcal{I}' \xrightarrow{\alpha} \mathcal{I}''$. In the first case by inductive hypothesis $\|_{r \in \mathrm{roles}(\mathcal{I})} (\pi(\mathcal{I}, r), \Sigma_r)_r \xrightarrow{\alpha} \mathcal{N}'''$ and $\mathrm{upd}(\mathcal{N}''') = \|_{r \in \mathrm{roles}(\mathcal{I})} (\pi(\mathcal{I}''', r), \Sigma'_r)_r$. Thus $\|_{r \in \mathrm{roles}(\mathcal{I})} (\pi(\mathcal{I}, r); \pi(\mathcal{I}', r), \Sigma_r)_r \xrightarrow{\alpha} \mathcal{N}$ and we have $\mathrm{upd}(\mathcal{N}) = \|_{r \in \mathrm{roles}(\mathcal{I})} (\pi(\mathcal{I}''', r); \pi(\mathcal{I}', r), \Sigma'_r)_r$. If $\mathrm{roles}(\mathcal{I}') \subseteq \mathrm{roles}(\mathcal{I})$ then the thesis follows. Otherwise roles in $\mathrm{roles}(\mathcal{I}') \setminus \mathrm{roles}(\mathcal{I})$ are unchanged. Note however that the projection of $\mathcal{I}$ on these roles is a term composed only by $\mathbf{1}$s, which can be removed by upd.

If $\mathcal{I} \xrightarrow{\surd}$ and $\mathcal{I}' \xrightarrow{\alpha} \mathcal{I}''$ then by inductive hypothesis $\mathrm{proj}(\mathcal{I}', \Sigma) \xrightarrow{\alpha} \mathcal{N}''$ with $\mathrm{upd}(\mathcal{N}'') = \mathrm{proj}(\mathcal{I}'', \Sigma')$. The thesis follows since thanks to Lemma 7 $\mathrm{proj}(\mathcal{I}; \mathcal{I}', \Sigma) \xrightarrow{\alpha} \mathcal{N}$ and $\mathrm{upd}(\mathcal{N}) = \mathrm{proj}(\mathcal{I}'', \Sigma')$.

Note that in both the cases conditions on events follow by inductive hypothesis.

$\leftarrow$ Suppose that

$$\mathcal{N} = \|_{r \in \mathrm{roles}(\mathcal{I}; \mathcal{I}')} (\pi(\mathcal{I}, r); \pi(\mathcal{I}', r), \Sigma_r)_r \xrightarrow{\alpha} \|_{r \in \mathrm{roles}(\mathcal{I}; \mathcal{I}')} (P_r, \Sigma'_r)_r$$

with $\alpha \in \{o^? : r_1(v) \to r_2(x), E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}, E, \mathbf{R}, [\texttt{no-adapt}], \tau\}$. We have a case analysis on α.

If $\alpha = o^? : r_1(v) \to r_2(x)$ then $\pi(\mathcal{I};\mathcal{I}', r_1) \xrightarrow{\Sigma_{r_1}, \overline{o^?}\langle v\rangle @ r_2} P_{r_1}$ and also $\pi(\mathcal{I};\mathcal{I}', r_2) \xrightarrow{o^?(x \leftarrow v)@r_1} P_{r_2}$. The two events should have the same global index thanks to Lemma 4. Thus, they are either both from $\mathcal{I}$ or both from $\mathcal{I}'$.
In the first case we have also

$$\|_{r\in \mathrm{roles}(\mathcal{I};\mathcal{I}')}\ (\pi(\mathcal{I}, r), \Sigma_r)_r \xrightarrow{o^?:r_1(v)\to r_2(x)} \|_{r\in \mathrm{roles}(\mathcal{I};\mathcal{I}')}\ (P''_r, \Sigma'_r)_r$$

with $P_r = P''_r; \pi(\mathcal{I}', r)$. Thus, by inductive hypothesis, $\mathcal{I} \xrightarrow{\Sigma, o^?:r_1(v)\to r_2(x)} \mathcal{I}''$ and $\mathrm{upd}(\|_{r\in \mathrm{roles}\,\mathcal{I};\mathcal{I}'}\ (P''_r, \Sigma_r)_r)$ is the projection of $\mathcal{I}''$ with state Σ. Hence, we have that $\mathcal{I};\mathcal{I}' \xrightarrow{\Sigma, o^?:r_1(v)\to r_2(x)} \mathcal{I}'';\mathcal{I}'$.
In the second case, thanks to Lemma 8, we have that the interaction is enabled. Thus, $\mathcal{I} \xrightarrow{\surd}$ and $\mathcal{I}' \xrightarrow{\Sigma, o^?:r_1(v)\to r_2(x)} \mathcal{I}''$. Thanks to Lemma 7 then we have $\pi(\mathcal{I}, r_1) \xrightarrow{\surd}$ and $\pi(\mathcal{I}, r_2) \xrightarrow{\surd}$. Thus, we have $\pi(\mathcal{I}', r_1) \xrightarrow{\Sigma_{r_1}, \overline{o^?}\langle v\rangle @ r_2} P_{r_1}$, $\pi(\mathcal{I}', r_2) \xrightarrow{o^?(x\leftarrow v)@r_1} P_{r_2}$ and $\mathrm{proj}(\mathcal{I}', \Sigma) \xrightarrow{o^?:r_1(v)\to r_2(x)} \|_{r\in \mathrm{roles}(\mathcal{I}')}$ $(P_r, \Sigma_r)_r$. The thesis follows by inductive hypothesis.
For the other possibilities of α, only the process of one role changes. Thus, the thesis follows by induction.
Note that in all the above cases, conditions on events follow by inductive hypothesis.

Case $\mathcal{I} \parallel \mathcal{I}'$**:** from the definition of the projection function we have $\mathcal{N} = \|_{r\in \mathrm{roles}(\mathcal{I};\mathcal{I}')}$ $(\pi(\mathcal{I}, r) \mid \pi(\mathcal{I}', r), \Sigma_r)_r$.

- $\to$ If $\mathcal{I} \parallel \mathcal{I}'$ can perform an interaction then one of its two components can perform the same interaction and the thesis follows by inductive hypothesis. Additional roles not occurring in the term performing the transition are dealt with by function upd.
- $\leftarrow$ We have a case analysis on α. If $\alpha = o^? : r_1(v) \to r_2(x)$ then an input and an output on the same operation are enabled. Thanks to Lemma 4 they have the same global index. Thus they are from the same component and the thesis follows by inductive hypothesis. For the other possibilities of α, only the process of one role changes. The thesis follows by induction. In all the cases, roles not occurring in the term performing the transition are dealt with by function upd.

Case n : ***if*** $b@r$ $\{\mathcal{I}\}$ ***else*** $\{\mathcal{I}'\}$**:** from the definition of projection

$$\begin{aligned}\mathcal{N} = \|_{s\in \mathrm{roles}(\mathcal{I})\cup \mathrm{roles}(\mathcal{I}')\setminus\{r\}}\ & (o^*_n : x_n\ \textit{from}\ r;\\ & \quad \textit{if}\ x_n\ \{\pi(\mathcal{I}, s)\}\ \textit{else}\ \{\pi(\mathcal{I}', s)\}, \Sigma_s)_s\ \|\\ & (\textit{if}\ b\ \{\Pi_{r'\in \mathrm{roles}(\mathcal{I})\cup \mathrm{roles}(\mathcal{I}')\setminus\{r\}} o^*_n : \textit{true}\ \textit{to}\ r'; \pi(\mathcal{I}, r)\}, \Sigma_r)_r\end{aligned}$$

Let us consider the case when the condition is true (the other one is analogous).

$\rightarrow$ The only possible transition from the AIOC is n : *if* $b@r$ $\{\mathcal{I}\}$ *else* $\{\mathcal{I}'\} \xrightarrow{\Sigma,\tau} I$. The APOC can match this transition by reducing to

$$\begin{aligned}\mathcal{N}' = \|_{s\in \mathrm{roles}(\mathcal{I})\cup \mathrm{roles}(\mathcal{I}')\setminus\{r\}} \ & (o_n^* : x_n \textit{ from } r; \\ & \textit{if } x_n \ \{\pi(\mathcal{I}, s)\} \textit{ else } \{\pi(\mathcal{I}', s)\}, \Sigma_s)_s \ \| \\ & (\Pi_{r'\in \mathrm{roles}(\mathcal{I})\cup \mathrm{roles}(\mathcal{I}')\setminus\{r\}} o_n^* : \mathit{true} \textit{ to } r'; \pi(\mathcal{I}, r), \Sigma_r)_r\end{aligned}$$

By applying function upd we get

$$\mathrm{upd}(\mathcal{N}') = \|_{s\in \mathrm{roles}(\mathcal{I})\cup \mathrm{roles}(\mathcal{I}')\setminus\{r\}} \ (\pi(\mathcal{I}, s), \Sigma_s)_s \ \| \ (\pi(\mathcal{I}, r), \Sigma_r)_r$$

Concerning events, at the AIOC level events corresponding to the guard and to the non-chosen branch are removed. The same holds at the APOC level, thus conditions on the remaining events are inherited. This concludes the proof.

$\leftarrow$ The only possible transition from the APOC is the evaluation of the guard from the coordinator. This reduces $\mathcal{N}$ to $\mathcal{N}'$ above and the thesis follows from the same reasoning.

Case n : ***while*** $b@r$ $\{\mathcal{I}\}$**:** from the definition of projection

$$\begin{aligned}\mathcal{N} = \|_{s\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} \ & (o^* : x_n \textit{ from } r; \\ & \textit{while } x_n \ \{\pi(\mathcal{I}, s); o_n^* : \mathtt{ok} \textit{ to } r; o_n^* : x_n \textit{ from } r\}, \Sigma_s)_s \ \| \\ & (\textit{while } b \ \{\Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{true} \textit{ to } r'; \pi(\mathcal{I}, r); \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \_ \textit{ from } r'\}; \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{false} \textit{ to } r', \Sigma_r)_r\end{aligned}$$

$\rightarrow$ Let us consider the case when the condition is true. The only possible transition from the AIOC is n : *while* $b@r$ $\{\mathcal{I}\} \xrightarrow{\Sigma,\tau} \mathcal{I}; n$: *while* $b@r$ $\{\mathcal{I}\}$. The APOC can match this transition by reducing to

$$\begin{aligned}\mathcal{N}' = \|_{s\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} \ & (o^* : x_n \textit{ from } r; \\ & \textit{while } x_n \ \{\pi(\mathcal{I}, s); o_n^* : \mathtt{ok} \textit{ to } r; o_n^* : x_n \textit{ from } r\}, \Sigma_s)_s \ \| \\ & (\Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{true} \textit{ to } r'; \pi(\mathcal{I}, r); \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \_ \textit{ from } r'; \\ & \textit{while } b \ \{\Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{true} \textit{ to } r'; \pi(\mathcal{I}, r); \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \_ \textit{ from } r'\}; \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{false} \textit{ to } r', \Sigma_r)_r\end{aligned}$$

By applying function upd we get

$$\begin{aligned}\mathrm{upd}(\mathcal{N}') = \|_{s\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} \ & (\pi(\mathcal{I}, s); o_n^* : x_n \textit{ from } r; \\ & \textit{while } x_n \ \{\pi(\mathcal{I}, s); o_n^* : \mathtt{ok} \textit{ to } r; o_n^* : x_n \textit{ from } r\}, \Sigma_s)_s \ \| \\ & (\pi(\mathcal{I}, r); \textit{while } b \ \{\Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{true} \textit{ to } r'; \pi(\mathcal{I}, r); \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \_ \textit{ from } r'\}; \\ & \Pi_{r'\in \mathrm{roles}(\mathcal{I})\setminus\{r\}} o_n^* : \mathit{false} \textit{ to } r', \Sigma_r)_r\end{aligned}$$

which is exactly the projection of $\mathcal{I}; n : \textit{while}\ b@r\ \{\mathcal{I}\}$.

As far as events are concerned, in $\mathrm{prop}(\mathcal{N}')$ we have all the needed events since, in particular, we have already done the unfolding of the while in all the roles. Concerning the ordering, at the AIOC level, we have two kinds of relations: (1) events in the unfolded process precede the guard event; (2) the guard event precedes the events in the body. The first kind of relations is matched at the APOC level thanks to the auxiliary synchronizations that close the unfolded body (which are not removed by prop) using synchronization and sequentiality. The second kind of relations is matched thanks to the auxiliary synchronizations that start the following iteration using synchronization, sequentiality and while.

The case when the condition evaluates to `false` is simpler.

$\leftarrow$ The only possible transition from the APOC is the evaluation of the guard from the coordinator. This reduces $\mathcal{N}$ to $\mathcal{N}'$ above and the thesis follows from the same reasoning.

Case $n : \textit{scope}\ l@r\ \{\mathcal{I}\}\ \textit{prop}\ \{\Delta\}$**:** from the definition of the projection

$\mathcal{N} = \|_{s \in \mathrm{roles}(\mathcal{I}) \cup \{r\}}\ (\pi(n : \textit{scope}\ l@r\ \{\mathcal{I}\}\ \textit{prop}\ \{\Delta\}, s), \Sigma_s)_s$.

$\rightarrow$ The only possible transitions are obtained by applying rules LEAD-ADAPT or LEAD-NOADAPT to the coordinator scope. Let us consider the first case.

$$\|_{s \in \mathrm{roles}(\mathcal{I}) \cup \{r\}}\ (\pi(n : \textit{scope}\ l@r\ \{\mathcal{I}\}\ \textit{prop}\ \{\Delta\}, s), \Sigma_s)_s \xrightarrow{E, \mathbf{R}, [l, \mathcal{C}] \mapsto \mathcal{I}'} \|_{s \in \mathrm{roles}(\mathcal{I}) \cup \{r\}}\ (P_s, \Sigma_s)_s = \mathcal{N}'$$

For the coordinator we have that P_r is:

$$\begin{aligned} &\mathit{\Pi}_{r_i \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} \\ &\qquad o^*_{l,n} : \pi(\mathrm{freshIndex}(\mathcal{I}', n), r_i)\ \textit{to}\ r_i; \pi(\mathrm{freshIndex}(\mathcal{I}', n), r); \\ &\qquad\qquad \mathit{\Pi}_{r_i \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}} o^*_{l,n} : \_\ \textit{from}\ r_i \end{aligned}$$

For other roles $P_{r_i} = n : \textit{scope}\ l@r\ \{P\}$. By applying the upd function we get:

$$\begin{aligned} \mathrm{upd}(\mathcal{N}') = \pi(\mathrm{freshIndex}(\mathcal{I}', n), r)\ \| \\ \|_{r_i \in \mathrm{roles}(\mathcal{I}) \setminus \{r\}}\ \pi(\mathrm{freshIndex}(\mathcal{I}', n), r_i) \end{aligned}$$

This is exactly the projection of the AIOC obtained after applying the rule ADAPT. The conditions on events are inherited. Observe that the closing event of the scope is replaced by events corresponding to the auxiliary interactions closing the scope. This allows us to preserve the causality dependencies also when the scope is inserted in a bigger context.

The case of rule LEAD-NOADAPT is simpler.

$\leftarrow$ The only possible transition from the APOC is the one of the coordinator of the scope checking whether adaptation is needed. This reduces $\mathcal{N}$ to $\mathcal{N}'$ above and the thesis follows from the same reasoning.