Towards Global and Local Types for Adaptation

Mario Bravetti, Marco Carbone, Thomas Hildebrandt, Ivan Lanese, Jacopo Mauro, Jorge A. Perez, Gianluigi Zavattaro

Towards Global and Local Types for Adaptation

Mario Bravetti[1], Marco Carbone[2], Thomas Hildebrandt[2], Ivan Lanese[1], Jacopo Mauro[1], Jorge A. Pérez[3], and Gianluigi Zavattaro[1]

[1] Lab. Focus, University of Bologna/INRIA
[2] IT University of Copenhagen
[3] CITI and Departamento de Informática, FCT - Universidade Nova de Lisboa

Abstract. Choreographies allow designers to specify the protocols followed by participants of a distributed interaction. In this context, adaptation may be necessary to respond to external requests or to better suit a changing environment (a self-update). Adapting the behavior of a participant requires to update in a coordinated way possibly all the participants interacting with him. We propose a language able to describe a choreography together with its adaptation strategies, and we discuss the main issues that have to be solved to enable adaptation on a participant code dealing with many interleaved protocols.

1 Introduction

Modern complex distributed software systems face the great challenge of adapting to varying contextual conditions, user requirements or execution environments. Service-oriented Computing (SOC), and service-oriented architectures in general, have been designed to support a specific form of adaptation: services can be dynamically discovered and properly combined in order to achieve an overall service composition that satisfies some specific desiderata that could be known only at service composition time. Rather sophisticated theories have been defined for checking and guaranteeing the correctness of these service assemblies (see, e.g., the rich literature on choreography/orchestration languages [2, 14], behavioral contracts [7, 6], and session types [4, 12, 5]). In this paper, we consider a more fine-grained form of adaptation that can occur when the services have been already combined but have not yet completed their task. This form of adaptation may arise, for instance, when the desiderata dynamically change or when some unexpected external event occurs. In particular in the context of computer-supported case management, e.g. for health-care or financial services, changes are the norm rather than the exception. This has lead to an increasing interest both from academia and industry in the development of technologies supporting dynamic changes in choreographies and processes, collectively referred to as *adaptive case management* (ACM) [19, 17] and being addressed in the recent proposal for a Case Management Model and Notation (CMMN) from OMG [18]. For such technologies, it is crucial that modifications occur in a consistent and coordinated manner in order to avoid breaking the correctness of the overall service composition.

In this paper, we initiate the investigation of new models and theories for service composition that properly take into account this form of adaptation. First of all, we extend a previous language for the description of service choreographies [2] with two

operators: the first one allows for the specification of *adaptable scopes* that can be dynamically modified, while the second may dynamically update code in one of such scopes. This language is designed for the *global description* of dynamically adaptable multi-party interaction protocols. As a second step in the development of our theory, we define a service behavioral contract language for the *local description* of the input-output communications. In order to support adaptation, also in this case we enhance an existing service contract language [2] with two new operators for adaptable scope declaration and code update, respectively. The most challenging aspect to be taken into account is the fact that, at the local level, peers should synchronize their local adaptations in order to guarantee a consistent adaptation of their behavior. As mentioned above, these two languages are expected to be used to describe multi-party protocols from global and local perspectives, respectively. The relationship between the two languages is formalized in terms of a *projection* function which allows us to obtain endpoint specifications from a given choreography.

The complete theory that we plan to develop will also consider a concrete language for programming services; such a language will include update mechanisms like those provided by, for instance, the Jorba service orchestration language [13]. The ultimate aim of our research is to define appropriate behavioral typing techniques able to check whether the concretely programmed services correctly implement the specified multi-party adaptable protocols. This will be achieved by considering the global specification of the protocol, by projecting such specification on the considered peer, and then by checking whether the actual service correctly implements the projected behavior. In order to clarify our objective, we discuss an example inspired by a health-care scenario [16]. Two adaptable protocols are described by using the proposed choreography languages: the first protocol describes the interaction between the *doctor* and the *laboratory* agents, while the second involves a *doctor*, a *nurse*, and a *patient*. In case of emergency, the doctor may speed up the used protocols by interrupting running tests and avoiding the possibility that the nurse refuses to use a medicine she does not trust —this possibility is normally allowed by the protocol. Then, using a π-calculus-like language, we present the actual behavior of the doctor and discuss the kinds of problems that we will have to address in order to define appropriate behavioral type checking techniques.

Structure of the paper. The next section introduces choreography and endpoint languages with adaptation constructs, and the projection function that relates global and local specifications. Then, in §3 we outline a concrete specification language and discuss the health-care scenario. In §4 we present some concluding remarks and briefly review related works.

Disclaimer. This paper discusses ongoing work supported by the "Behavioural Types for Reliable Large-Scale Software Systems" (BETTY) Cost Action. Our main aim is to report about the current state of this research activity.

2 Choreography and Endpoint Languages for Adaptation

In the paper, we use the following sets: *channels*, ranged over by $a, a', \ldots$; *scope names*, ranged over by $X, X', \ldots$; and *roles/participants*, ranged over by $r, r_1, r_2, \ldots$. Also, we use $T, T', \ldots$ to denote sets of roles.

2.1 Choreography Language

Syntax. We describe here the syntax of our choreography language. To this end, we first define a set of so-called *choreography terms*. Then, by requiring some well-formedness conditions on such terms, we obtain actual *choreographies*.

The syntax of choreography terms is as follows:

$$
\begin{array}{lllll}
C ::= a_{r_1 \to r_2} & \text{(interaction)} & \mid & C \ ; \ C & \text{(sequence)} \\
\mid \ C \mid C & \text{(parallel)} & \mid & C + C & \text{(choice)} \\
\mid \ C^* & \text{(star)} & \mid & \mathbf{1} & \text{(one)} \\
\mid \ \mathbf{0} & \text{(nil)} & & & \\
\mid \ X : T[C] & \text{(scope)} & \mid & X_r\{C\} & \text{(update)}
\end{array}
$$

The basic element of a choreography term C is an interaction $a_{r_1 \to r_2}$, with the intended meaning that participant r_1 sends a message to participant r_2 over channel a. Two terms C_1 and C_2 can be composed in sequence ($C_1 \ ; \ C_2$), in parallel ($C_1 \mid C_2$), and using nondeterministic choice ($C_1 + C_2$). Also, a choreography term may be iterated zero or more times using the Kleene star *. The empty choreography term, which just successfully terminates, is denoted by $\mathbf{1}$. The deadlocked choreography term $\mathbf{0}$ is needed for the definition of the semantics: we will assume that it is never used when writing a choreography (see Definition 1).

The two last operators deal with adaptation. Adaptation is specified by defining a *scope* that delimits a choreography term that, at runtime, may be replaced by a new choreography term, coming from either inside or outside the system. Adaptations coming from outside may be decided by the user through some adaptation interface, by some manager module, or by the environment. In contrast, adaptations coming from inside represent self-updates, decided by a part of the system towards itself or towards another part of the system, usually as a result of some interaction producing unexpected values. Adaptations from outside and from inside are indeed quite similar, e.g., an update decided by a manager module may be from inside if the manager behavior is part of the choreography term, from outside if it is not. Construct $X : T[C]$ defines a scope named X currently executing choreography term C — the name is needed to designate it as a target for a particular adaptation. Type T is the set of roles (possibly) occurring in the scope. This is needed since a given update can be applied to a scope only if it specifies how all the involved roles are adapted. Operator $X_r\{C\}$ defines *internal updates*, i.e., updates offered by a participant of the choreography term. Here r denotes the participant offering the update, X is the name of the target scope, and C is the new choreography term.

Not all choreography terms generated by the syntax above are useful choreographies. To formally define the choreography terms which actually represent choreographies, we rely on some auxiliary definitions. The set of roles inside a choreography term

C, denoted $roles(C)$, is defined inductively as follows:

$$\begin{aligned} roles(a_{r_1 \to r_2}) &= \{r_1, r_2\} & roles(X_r\{C\}) &= \{r\} \\ roles(X : T[C]) &= T \cup roles(C) & roles(C^*) &= roles(C) \\ roles(C_1 \ ; \ C_2) &= roles(C_1 \mid C_2) = roles(C_1 + C_2) = roles(C_1) \cup roles(C_2) \\ roles(\mathbf{1}) &= roles(\mathbf{0}) = \emptyset \end{aligned}$$

Notice that for $X_r\{C\}$ we consider role r but not the roles in C. This is because $X_r\{C\}$ may correspond to an external update on some different choreography term. We are now ready to define choreographies.

Definition 1 (Choreography). *A choreography term C is a choreography if:*

1. *C does not contain occurrences of $\mathbf{0}$;*
2. *all names of scopes in C are pairwise distinct;*
3. *C is well-typed, i.e. for every scope $X : T[C']$ occurring in C:*
   - *$roles(C') \subseteq T$ and*
   - *every update prefix $X_r\{C''\}$ occurring in C is such that $roles(C'') \subseteq T$.*

We use $type(X)$ to denote the type T associated to the unique scope $X : T[C']$.

Semantics. We now define the semantics of choreography terms via a labeled transition system. As in the syntax, the most interesting part of the semantics concerns update constructs. Recall that T is a set of roles. In the definition below, we use $C[C'/X]$ to denote the substitution that replaces all scopes $X : T[C'']$ with name X occurring in C (not inside update prefixes) with $X : T[C']$. As usual, transition $C \xrightarrow{\alpha} C'$ intuitively says that choreography term C may evolve to C' by performing an action represented by a label α. Our set of labels includes $\surd$ (termination), $a_{r_1 \to r_2}$ (interaction), and $X_r\{C\}$ (update).

Definition 2. *The semantics of choreography terms is the smallest labeled transition system closed under the rules in Table 1.*

We briefly comment on the rules in Table 1. Rules in the first four rows of the table are standard (cf. [2]). Rule (ONE) defines termination for the empty choreography term. Rule (COMM) executes an interaction, making it visible in the label. While rule (SEQ) allows the first component of a sequential composition to compute, rule (SEQTICK) allows it to terminate, starting the execution of the second component. Rule (PAR) allows parallel components to interleave their executions. Rule (PARTICK) allows parallel components to synchronize their termination. Rule (CHO) selects a branch in a nondeterministic choice. Rule (STAR) unfolds the Kleene star. Note that the unfolding may break uniqueness of scopes with a given name—we will come back to this point later on. Rule (STARTICK) defines termination of a Kleene star.

The remaining rules in Table 1 deal with adaptation. Rule (COMMUPD) makes an internal adaptation available, moving the information to the label. Adaptations propagate through sequence, parallel composition, and Kleene star using rules (SEQUPD),

$$(\textsc{One})\ \frac{}{\mathbf{1} \xrightarrow{\checkmark} \mathbf{0}} \qquad (\textsc{Comm})\ \frac{}{a_{r_1 \to r_2} \xrightarrow{a_{r_1 \to r_2}} \mathbf{1}} \qquad (\textsc{Seq})\ \frac{C_1 \xrightarrow{a_{r_1 \to r_2}} C_1'}{C_1;\ C_2 \xrightarrow{a_{r_1 \to r_2}} C_1'; C_2}$$

$$(\textsc{SeqTick})\ \frac{C_1 \xrightarrow{\checkmark} C_1' \qquad C_2 \xrightarrow{\alpha} C_2'}{C_1;\ C_2 \xrightarrow{\alpha} C_2'} \qquad (\textsc{Par})\ \frac{C_1 \xrightarrow{a_{r_1 \to r_2}} C_1'}{C_1 \mid C_2 \xrightarrow{a_{r_1 \to r_2}} C_1' \mid C_2}$$

$$(\textsc{ParTick})\ \frac{C_1 \xrightarrow{\checkmark} C_1' \qquad C_2 \xrightarrow{\checkmark} C_2'}{C_1 \mid C_2 \xrightarrow{\checkmark} C_1' \mid C_2'} \qquad (\textsc{Cho})\ \frac{C_1 \xrightarrow{\alpha} C_1'}{C_1 + C_2 \xrightarrow{\alpha} C_1'}$$

$$(\textsc{Star})\ \frac{C \xrightarrow{a_{r_1 \to r_2}} C'}{C^* \xrightarrow{a_{r_1 \to r_2}} C';\ C^*} \qquad (\textsc{StarTick})\ \frac{}{C^* \xrightarrow{\checkmark} \mathbf{0}}$$

$$(\textsc{CommUpd})\ \frac{}{X_r\{C\} \xrightarrow{X_r\{C\}} \mathbf{1}} \qquad (\textsc{SeqUpd})\ \frac{C_1 \xrightarrow{X_r\{C\}} C_1'}{C_1;\ C_2 \xrightarrow{X_r\{C\}} C_1'; (C_2[C/X])}$$

$$(\textsc{ParUpd})\ \frac{C_1 \xrightarrow{X_r\{C\}} C_1'}{C_1 \mid C_2 \xrightarrow{X_r\{C\}} C_1' \mid (C_2[C/X])} \qquad (\textsc{StarUpd})\ \frac{C_1 \xrightarrow{X_r\{C\}} C_1'}{C_1^* \xrightarrow{X_r\{C\}} C_1';\ (C_1[C/X])^*}$$

$$(\textsc{ScopeUpd})\ \frac{C_1 \xrightarrow{X_r\{C\}} C_1'}{X : T[C_1] \xrightarrow{X_r\{C\}} X : T[C]}$$

$$(\textsc{Scope})\ \frac{C_1 \xrightarrow{\alpha} C_1' \qquad \alpha \neq X_r\{C\} \text{ for any } r, C}{X : T[C_1] \xrightarrow{\alpha} X : T[C_1']}$$

Table 1. Semantics of Choreography Terms

(PARUPD), and (STARUPD), respectively. Note that, while propagating, the update is applied to the continuation of the sequential composition, to parallel terms, and to the body of Kleene star. Notably, the update is applied to both enabled and non enabled occurrences of the desired scope. Rule (SCOPEUPD) allows a scope to update itself (provided that the names coincide), while propagating the update to the rest of the choreography term. Rule (SCOPE) allows a scope to compute.

We can now define the notion of *closed* traces that correspond to computations of stand-alone choreography terms.

Definition 3 (Traces). *Given a choreography term C_0 a trace is a (possibly infinite) sequence $C_0 \xrightarrow{\alpha_1} C_1 \xrightarrow{\alpha_2} C_2 \xrightarrow{\alpha_3} \cdots$.*

In order to model choreography terms that can be externally updated we need to introduce the notion of *open* transitions.

Definition 4 (Open transitions). *The choreography term C has an open transition of the form $C \xrightarrow{X\{C''\}} C[C''/X]$ if:*

- *there is a choreography C_0 with a trace $C_0 \xrightarrow{\alpha} \cdots \xrightarrow{\alpha'} C_0'|C$;*

– $C_0' \xrightarrow{X_r\{C''\}} C_0''$ *where* $r \notin roles(C)$ *and* X *is the name of a scope in* C.

We can now define the notion of *open traces* corresponding to computations including also open transitions.

Definition 5 (Open Traces). *Given a choreography term* C_0 *an open trace is a (possibly infinite) sequence* $C_0 \xrightarrow{\alpha_1} C_1 \xrightarrow{\alpha_2} C_2 \xrightarrow{\alpha_3} \cdots$ *where every* $C_i \xrightarrow{\alpha_{i+1}} C_{i+1}$ *is either a transition of the semantics in Table 1 or an open transition.*

As we have said, in a choreography we assume scope names to be unique. However, uniqueness is not preserved by transitions. Nevertheless a slightly weaker property (arising from the fact that we consider Kleene star as the only form of recursion) is indeed preserved, and it simplifies the implementation of the adaptation mechanisms at the level of endpoints.

Proposition 1. *Let* C *be a choreography and let* C' *be a choreography term reachable from* C *via zero or more transitions (possibly open). For every* X*,* C' *contains at most one occurrence of a scope named* X *which is enabled (i.e., which can compute).*

An Example. Below we give an example of an adaptable choreography to illustrate the features introduced above. The example is based on a health-care workflow inspired by field study [16] carried out in previous work. The field study was also considered as inspiration for recent work on session types for health-care processes [11] and adaptable declarative case management processes [17], but the combination of session types and adaptability has not been treated previously.

In the considered scenario, doctors, nurses and patients employ a distributed, electronic health-care record system where each actor (including the patient) uses a tablet pc/smartphone to coordinate the treatment. Below, iteration C^+ stands for $C; C^*$.

$$X : \{D, N\}[((prescribe_{D\to N})^+;$$
$$(sign_{D\to N} + X_D\{sign_{D\to N}\}; up_{D\to N}); trust_{N\to D})^+];$$
$$medicine_{N\to P}$$

where D, N, P denote participant doctors, nurses, and patients, respectively.

The doctor first records one or more prescriptions, which are sent to the nurse's tablet $(prescribe_{D\to N})^+$. When receiving a signature, $sign_{D\to N}$, the nurse informs the doctor if the prescription is trusted. If not trusted then the doctor must prescribe a new medicine. If trusted, the nurse proceeds and gives the medicine to the patient, which is recorded at the patient's smartphone, $medicine_{N\to P}$. However, instead of signing and waiting for the nurse to trust the medicine, in emergency cases the doctor may update the protocol so that the possibility of not trusting the prescription is removed: the nurse would have to give the medicine to the patient right after receiving the signature. In the example, this is done by a self-update ($X_D\{sign_{D\to N}\}$) of the running scope. In other scenarios, this could have been done by an entity not represented in the choreography, such as the hospital director, thus resulting in an external update. The doctor notifies the protocol update to the nurse using the $up_{D\to N}$ interaction.

Now consider the further complication that the doctor may run a test protocol with a laboratory, after prescribing a medicine and before signing:

$$X'\{D, L\} : [orderTest_{D\to L}\ ;\ (results_{L\to D} + X'_D\{\mathbf{1}\})]$$

We allow the test protocol also to be adaptable, since the doctor may decide that there is an emergency while waiting for the results, and thus also having to interrupt the test protocol. If the two protocols are performed in interleaving by the same code, then the updates of the two protocols should be coordinated. We illustrate this in § 3 below.

2.2 Endpoint Language

Since choreographies are at the very high level of abstraction, defining a description of the same system nearer to an actual implementation is of interest. In particular, for each participant in a choreography (also called *endpoint*) we would like to describe the actions it has to take in order to follow the choreography. The syntax of endpoint processes is as follows:

$P ::= \overline{a}_r$	(output)	$\mid$	a_r	(input)
$\mid\ P\ ;\ P$	(sequence)	$\mid$	$P \mid P$	(parallel)
$\mid\ P + P$	(choice)	$\mid$	P^*	(star)
$\mid\ \mathbf{1}$	(one)	$\mid$	$\mathbf{0}$	(zero)
$\mid\ X[P]^F$	(scope)	$\mid$	$X_{(r_1,\ldots,r_n)}\{P_1, \ldots, P_n\}$	(update)

where F is either A, denoting an active (running) scope, or ε, denoting a scope still to be started (ε is omitted in the following).

As for choreographies, endpoint processes contain some standard operators and some operators dealing with adaptation. Communication is performed by $\overline{a}_r$, denoting an output on channel a towards participant r. Dually, a_r denotes an input from participant r on channel a. Intuitively, an output $\overline{a}_r$ in role s and an input a_s in role r should synchronize. Two endpoint processes P_1 and P_2 can be composed in sequence ($P_1\ ;\ P_2$), in parallel ($P_1 \mid P_2$), and using nondeterministic choice ($P_1 + P_2$). Endpoint processes can be iterated using a Kleene star *. The empty endpoint process is denoted by $\mathbf{1}$ and the deadlocked endpoint process is denoted by $\mathbf{0}$.

Adaptation is applied to scopes. $X[P]^F$ denotes a scope named X executing process P. F is a flag distinguishing scopes whose execution has already begun (A) from scopes which have not started yet (ε). The update operator $X_{(r_1,\ldots,r_n)}\{P_1, \ldots, P_n\}$ provides an update for scope named X, involving roles $r_1, \ldots, r_n$. The new process for role r_i is P_i.

Endpoints are of the form $[\![P]\!]_r$, where r is the name of the endpoint and P its process. Systems, denoted S, are obtained by composition of parallel endpoints:

$$S ::= [\![P]\!]_r \quad \text{(endpoint)} \qquad \mid \qquad S \| S \quad \text{(parallel system)}$$

As for choreographies, not all systems are endpoint specifications. By a slight abuse of notation we extend $type(X)$ to endpoints associating a set of roles to each scope name X. Endpoint specifications are defined as follows.

Definition 6. *A system S is an endpoint specification if the following conditions hold:*

(i) no active scopes are present
(ii) endpoint names are unique
(iii) all roles r occurring in terms of the form $\overline{a}_r$, a_r, or such that $r \in type(X)$ for some scope X are endpoints of S
(iv) a scope with name X con occur (outside updates) only in endpoints $r \in type(X)$
(v) every update has the form $X_{type(X)}\{P_1, \ldots, P_n\}$
(vi) outputs $\overline{a}_r$ and inputs a_r included in $X_{type(X)}\{P_1, \ldots, P_n\}$ are such that $r \in type(X)$.

In this presentation, we do not formally define a semantics for endpoints: we just point out that it should include labels corresponding to all the labels of the semantics of choreography terms, plus some additional labels corresponding to partial activities, such as an input. We also highlight the fact that all scopes which correspond to the same choreography scope evolve together: their scope start transitions (transforming a scope from inactive to active) are synchronized, as well as their scope end transitions (removing it). The fact that choreographies feature at most one scope with a given name is instrumental in ensuring this property.

2.3 Projection

Since choreographies provide system descriptions at the high level of abstraction and endpoint specifications provide more low level descriptions, a main issue is to derive from a given choreography an endpoint specification executing it. This is done using the notion of *projection*.

Definition 7 (Projection). *The projection of a choreography C on a role r, denoted by $C \upharpoonright_r$, is defined by the clauses below*

$$a_{r_1 \to r_2} \upharpoonright_r = \begin{cases} \overline{a}_{r_2} & \text{if } r = r_1 \\ a_{r_1} & \text{if } r = r_2 \\ \mathbf{1} & \text{otherwise} \end{cases}$$

$$X_{r'}\{C\} \upharpoonright_r = \begin{cases} X_{(r_1,\ldots,r_n)}\{C \upharpoonright_{r_1}, \ldots, C \upharpoonright_{r_n}\} \text{with } \{r_1, \ldots, r_n\} = type(X) \text{ if } r = r' \\ \mathbf{1} \qquad \text{otherwise} \end{cases}$$

$$X : T[C] \upharpoonright_r = \begin{cases} X[C \upharpoonright_r] & \text{if } r \in type(X) \\ \mathbf{1} & \text{otherwise} \end{cases}$$

and is an homomorphism on the other operators. The endpoint specification resulting from a choreography C is obtained by composing in parallel roles $[\![C \upharpoonright_r]\!]_r$, where $r \in roles(C)$.

As an example, the endpoint projection obtained from the prescribe choreography introduced in §2.1 is $[\![P_{\mathtt{N}}]\!]_N \| [\![P_{\mathtt{D}}]\!]_D \| [\![P_{\mathtt{P}}]\!]_P$ where processes $P_{\mathtt{N}}$, $P_{\mathtt{D}}$, and $P_{\mathtt{P}}$ are as fol-

lows (we omit unnecessary $\mathbf{1}$ processes):

$$P_{\mathtt{N}} = X[((prescribe_D)^+ \,;\, (sign_D + up_D \,);\, \overline{trust}_D)^+] \,;\, \overline{medicine}_P$$
$$P_{\mathtt{D}} = X[((\overline{prescribe}_N)^+ \,;\, (\overline{sign}_N + X_{D,N}\{\overline{sign}_N, sign_D\} \,;\, \overline{up}_N); trust_N)^+]$$
$$P_{\mathtt{P}} = medicine_N$$

One can see that the system S obtained by projecting a choreography is an endpoint specification. Ideally, traces of the projected system should correspond to the traces of the original choreography. Actually, we conjecture that this occurs only for choreographies satisfying suitable connectedness conditions that we plan to formalize extending those in [14]. This is not an actual restriction, since choreographies that do not respect the conditions can be transformed into choreographies that respect them [15].

Conjecture 1. Traces of projection of connected choreographies correspond to traces of the original choreography.

We point out two main aspects of the correspondence. First, labels $X_r\{C\}$ of transitions of the choreography should be mapped to labels $[X_{(r_1,\ldots,r_n)}\{P_1,\ldots,P_n\}]_r$ of the transitions of the endpoint specification, where $type(X) = \{r_1,\ldots,r_n\}$ and $P_1 = C \upharpoonright_{r_1}, \ldots, P_n = C \upharpoonright_{r_n}$ are obtained by projection from C. Second, endpoint traces should not consider unmatched input and output labels.

3 Typing a Concrete Language

As demonstrated by our examples, choreography and endpoint terms provide a useful language for expressing protocols with adaptation. In this section, we investigate the idea of using such protocols as specifications for a programming language with adaptation. We plan to follow the approach taken in multiparty session types [12], where choreographies (and endpoints) are interpreted as behavioral types for typing sessions in a language modeled as a variant of the π-calculus. In the sequel, we investigate the core points of such a language by giving an implementation that uses the protocols specified in the examples of the previous sections. In particular, we discuss what are the relevant aspects for developing a type system for such a language, whose types are the choreographies introduced in § 2.1.

In both prescribe and test protocols, the doctor plays a key role since (s)he initiates the workflow with prescriptions, decides when tests have to be requested, and decides when the protocols have to be interrupted due to an emergency. A possible implementation of the doctor could be given by the following program:

1. $P_{\mathtt{D}} = \overline{\mathsf{pr}}(k);\ X[\,\mathtt{repeat}\,\{\mathtt{repeat}\,\{$
2. $\quad k : \overline{prescribe}_N\langle e_{pr}\rangle;\ \overline{\mathsf{test}}(k');$
3. $\quad X'[k' : \overline{orderTest}_L\langle e_o\rangle;$
4. $\quad\quad (k' : results_L(x) +$
5. $\quad\quad\quad X'_{(D,L)}\{X_{(D,N)}\{k : \overline{sign}_N\langle e_s\rangle, k : sign_D(z)\}, \mathbf{1}\})]$
6. $\quad \}\ \mathtt{until}\ \mathsf{ok}(x);$
7. $\quad (k : \overline{sign}_N\langle e_s\rangle + X_{(D,N)}\{k : \overline{sign}_N\langle e_s\rangle, k : sign_D(z)\} \,;\, k : \overline{up}_N\langle\rangle);$
8. $\quad k : trust_N(t)\}\ \mathtt{until}\ \mathsf{trusted}(t)\ ]$

In the code there are two kinds of communication operations, namely protocol initiation operations, where a new protocol (or session) is initiated, and in-session operations where protocol internal operations are implemented. The communication $\overline{\mathsf{pr}}(k)$ is for initiating a protocol called pr and its semantics is to create a fresh protocol identifier k that corresponds to a particular instance of protocol pr. In-session communications are standard.

The novelty in the process above is in the scope $X[\ldots]$ and update $X_{\ldots}\{\ldots\}$, which state respectively that the program can be adapted at any time in that particular point, and that an adaptation is available. Interestingly enough, the way the program P_D uses the protocols needs care. If the doctor wants to adapt to emergency while waiting for tests, both the test protocol and the prescription protocol need to be adapted as shown in line 5. If the doctor adapts to emergency after having received tests that are ok, then only the prescription protocol needs to be adapted. One can see that session pr can be typed using the prescribe endpoint specification and session $test$ using the test endpoint specification. The update of X in line 4 does not appear in the protocol test since it acts as an external update for a different protocol.

4 Concluding Remarks and Related Work

Adaptation is a pressing issue in the design of service-oriented systems, which are typically open and execute in highly dynamic environments. There is a rather delicate tension between adaptation and the correctness requirements defined at service composition time: we would like to adapt the system's behavior whenever necessary/possible, but we would also like adaptation actions to preserve overall correctness.

In this paper, we have reported ongoing work on adaptation mechanisms for service-oriented systems specified in terms of choreographies. By enhancing an existing language for choreographies with constructs defining adaptation scopes and dynamic code update, we obtained a simple, global model for distributed, adaptable systems. We also defined an endpoint language for local descriptions, and a projection mechanism for obtaining (low-level) endpoint specifications from (high-level) choreographies.

We now briefly comment on related works. The work in [9] is closely related, and indeed was a source of inspiration for the current work. It develops a framework for rule-based adaptation in a choreographic setting. Both choreographies and endpoints are defined; their relation is formally defined via projection. The main difference w.r.t. the work described here is our choice of expressing adaptation in terms of scopes and code update constructs, rather than using rules. Also, we consider choreographies as types and we allow multiple protocols to interleave inside code. These problems are not considered in [9].

Our work is also related to the recent work [8], which 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. A main difference is that in their approach the code does not change because processes should be able to implement all the global descriptions since the very beginning.

Our approach bears some similarities with works on multiparty sessions [12, 4], and in particular with works dealing with exceptions in multiparty sessions [3]. Our focus so far has been on formally relating global and local descriptions of choreographies via projection and trace correspondence; investigating correctness properties (e.g., communication safety) via typing in our setting is part of ongoing work. We also note that exceptions and adaptation are similar but conceptually different phenomena: while the former are typically related to foreseen unexpected behaviors in (low-level) programs, adaptation appears as a more general issue, for it should account for (unforeseen) interactions between the system and its (varying) environment.

We have borrowed inspiration also from [17], in which adaptive case management is investigated via Dynamic Condition Response (DCR) Graphs, a declarative process model.

Finally, the adaptation constructs we have considered for choreographies and endpoints draw inspiration from the *adaptable processes* defined in [1]. The application of adaptable processes in session-typed models of structured communications (focusing on the case of binary sessions) has been studied in [10].

An immediate topic for future work is the full formalization of the concrete language and its typing disciplines. Other avenues for future research include the investigation of refinement theories with a testing-like approach, enabled by having both systems and adaptation strategies modeled in the same language, and the development of prototype implementations.

Acknowledgments. This work was partially supported by COST Action IC1201: Behavioural Types for Reliable Large-Scale Software Systems (BETTY). Jorge A. Pérez was partially supported by grants SFRH / BPD / 84067 / 2012 and CITI of the Portuguese Foundation for Science and Technology (FCT).

References

1. M. Bravetti, C. Di Giusto, J. A. Pérez, and G. Zavattaro. Adaptable processes. *Logical Methods in Computer Science*, 8(4), 2012.
2. M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *SC*, volume 4829 of *LNCS*, pages 34–50. Springer, 2007.
3. S. Capecchi, E. Giachino, and N. Yoshida. Global escape in multiparty sessions. In *FSTTCS*, volume 8 of *LIPIcs*, pages 338–351. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.
4. M. Carbone, K. Honda, and N. Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012.
5. M. Carbone and F. Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274. ACM, 2013.
6. S. Carpineti and C. Laneve. A basic contract language for web services. In *ESOP*, volume 3924 of *LNCS*, pages 197–213. Springer, 2006.
7. G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In *POPL*, pages 261–272. ACM, 2008.
8. M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Self-adaptive monitors for multiparty sessions. In *PDP*, 2014. To appear.

9. M. Dalla Preda, I. Lanese, J. Mauro, M. Gabbrielli, and S. Giallorenzo. Safe run-time adaptation of distributed applications, 2013. Submitted. Available at http://www.cs.unibo.it/~lanese/publications/fulltext/adaptchor2.pdf.gz.
10. C. Di Giusto and J. A. Pérez. Disciplined structured communications with consistent runtime adaptation. In *SAC*, pages 1913–1918. ACM, 2013.
11. A. S. Henriksen, L. Nielsen, T. Hildebrandt, N. Yoshida, and F. Henglein. Trustworthy pervasive healthcare services via multiparty session types. In *FHIES*, volume 7789 of *LNCS*, pages 124–141. Springer, 2013.
12. K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.
13. 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.
14. I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *SEFM*, pages 323–332. IEEE Computer Society, 2008.
15. I. Lanese, F. Montesi, and G. Zavattaro. Amending choreographies. In *WWV*, volume 123 of *EPTCS*, pages 34–48. Open Publishing Association, 2013.
16. K. M. Lyng, T. Hildebrandt, and R. R. Mukkamala. From paper based clinical practice guidelines to declarative workflow management. In *ProHealth*, pages 36–43. BPM 2008 Workshops, 2008.
17. R. R. Mukkamala, T. Hildebrandt, and T. Slaats. Towards trustworthy adaptive case management with dynamic condition response graphs. In *EDOC*, pages 127–136. IEEE, 2013.
18. OMG. Case management model and notation 1.0 - beta 1, January 2013.
19. K. D. Swenson. *Mastering the Unpredictable - How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done*. Meghan-Kiffer Press, 2010.