Customizing choreography: Deriving conversations from organizational dependencies

Customizing Choreography: Deriving Conversations from Organizational Dependencies

Ayman Mahfouz, Leonor Barroca, Robin Laney, Bashar Nuseibeh
Computing Department, The Open University. Walton Hall, Milton Keynes, MK7 6AA, UK
amahfouz@gmail.com, {L.Barroca, R.C.Laney, B.A.Nuseibeh}@open.ac.uk

Abstract

Evolving business needs call for customizable choreographed interactions. However, choreography descriptions do not capture the problem-domain knowledge required to perform the customization effectively. Hence, we propose performing the customization to models of organizational requirements motivating the interaction. To facilitate the derivation of the resulting choreography description, we propose an alignment between conversations and organizational dependencies. We employ the domain knowledge and formal semantics of requirements models to find customization alternatives and reason about them. Using the alignment, we derive constraints on conversations systematically from customized requirements models.

1. Introduction

A choreography description specifies the joint behavior of a group of "roles" in an electronic interaction from a neutral point of view [1]. Mutual obligations of the roles are specified in terms of constraints on the sequences of messages that can be exchanged between them [2]. Each sequence of messages specified in a choreography description constitutes a valid type of "conversation".

Conversations taking place between actual participants have to abide by the constraints specified on the behavior of their corresponding roles. Ideally, a choreography description will be deployed to a context that matches the original context it was designed for. Realistically, a deployment context will embody specialized business requirements that have to be reflected as additional constraints on the behavior of participants in that context. It is naturally desirable to reuse the original choreography description by customizing it for the new context rather than creating one from scratch for every context.

Generally, for a particular context a number of alternatives for representing the required customization will exist. To choose the alternative that best satisfies the additional requirements imposed by the context we need to evaluate how well each alternative addresses the stakeholders' (i.e. participants) needs. However, choreography descriptions are operational specifications that do not capture problem domain knowledge necessary for this kind of reasoning. In particular, physical activities that the participants undertake during the interaction are not necessarily reflected in choreography. To this purpose, we propose an approach for performing the required customization to models of organizational requirements that motivate the interaction.

Organizational requirements models capture the intentions of the interacting participants, the mutual dependencies driving them to interact, and the activities they undertake to fulfill their obligations, all of which are essential knowledge required for performing the customization. Hence, our approach uses the Tropos framework [3] as it provides suitable notations for representing and reasoning about this kind of problem-level knowledge. We also make use of the formal notations provided by Formal Tropos (FT) [4] for describing and arguing about constraints that govern the behavior of participants in the interaction.

We employ the formal semantics of FT in discovering alternative ways for capturing specialized business needs imposed by a deployment context. We put forward a technique by which a systematic traversal of FT models yields a set of potential alternatives for performing the required customization. We then use problem-domain knowledge embodied in Tropos models to reason about the alternatives and select the one that best matches stakeholders' needs.

To obtain a customized choreography description, we need to operationalize customizations made to organizational requirements into constraints on conversations. To enable automation and provide for effective reasoning about correctness, we need means to perform the operationalization systematically. For this reason, central to our approach is a proposed alignment between organizational dependencies and choreographed conversations. The alignment allows us to derive constraints on conversation from constraints specified in the requirements models in a systematic way.

The rest of the paper is structured as follows: In section 2, we introduce the notion of choreography customization and present the running example we use throughout the paper. In section 3 we show how organizational requirements are modeled in Tropos. Section 4 details our proposal for aligning conversations with organizational dependencies. Section 5 presents our proposed technique for finding and reasoning about customization alternatives to organizational requirements. We discuss related work

in section 6 then conclude and outline future work in section 7.

2. Customizing choreographed conversations

A choreography description specifies a contract between a group of interacting roles in terms of sequences of messages they are allowed to exchange. For example, consider an interaction between three roles: a patient, a medical provider (MP), and an in insurance company (IC). One potential interaction between these roles can be choreographed as follows: A patient who needs to visit an MP has to get an authorization from her IC first. When the patient receives an authorization number from the IC, she requests an appointment and provides her insurance information to the MP. Before confirming the requested appointment the MP verifies the patient info with the IC. After getting the confirmation the patient visits the MP to get examined by a doctor who later sends a prescription. The MP then bills the IC and gets back an electronic payment (Figure 1).

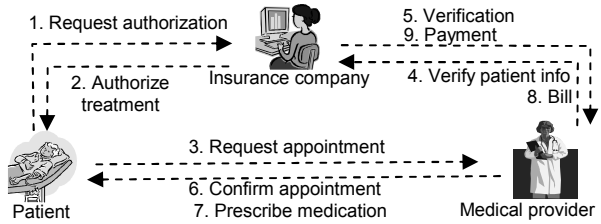


Figure 1. Choreographed medical interaction

A choreography description represents a contract between the interacting roles. Messaging between actual participants that play the choreographed roles at runtime has to abide by this contract

2.1. Choreographed conversations

The messaging sequence in a choreography description is logically divided into conversations. A conversation is "a set of communication events occurring at two or more participants that all correspond to achieving the same goal" where a "communication event" is a message sent or received by a participant [5]. A particular event E_s initiates a conversation C while another event E_f signals its termination. Other communication events belonging to C may occur only in between E_s and E_f. From the point where it is initiated till it gets terminated C is said to be "active". For example, an "Appointment" conversation involves four communication events; the patient sends a request, MP receives the request, MP sends a confirmation, and the patient receives it.

2.2. Choreography deployment context

A choreography description is deployed to a context that binds a subset of the universe of possible participants to the choreographed roles. Generally, a deployment context may entail special business needs in addition to what was originally specified. As a result, the original choreography description needs to be customized to impose additional constraints on the contract of the interaction. For example, consider the need to customize the medical interaction for a context that calls for protecting MPs from slow-paying ICs. One possible way to achieve this business need is by placing a limit on the number of "Payment" conversations that can be active between any IC-MP pair at one time. This may be represented as a constraint on messaging where the MP is disallowed from initiating any "Payment" conversations with a particular IC if that IC reached their limit. It is hard to rationalize this, or any other, choice for capturing the customization without considering how well it satisfies the business needs of the participants.

2.3. Choreography vs. requirements

To rationalize a customization, it is crucial to consult problem-domain knowledge. However, choreography is concerned with operational descriptions that embody little of this knowledge. Choreography only addresses "how" an interaction is realized in terms of message exchanges. On the other hand, organizational requirements provide more abstract descriptions that focus on the "why" and "what" aspects of the interaction. Models of organizational requirements motivating the interaction embody essential knowledge about the problem domain including:

a) Motivations driving the participants to interact,
b) Inter-dependencies between the participants that make it possible to achieve their goals from interacting, and
c) Activities they undertake to fulfill their obligations towards the interaction contract, including physical activities not captured in a choreography description.

This information is crucial to assessing and selecting from among alternative ways for capturing the required customization. Hence, we propose that customizations to the interaction contract be made to models of organizational requirements motivating the interaction, rather than directly to the constraints on messaging. Moreover, requirements models embody a precise representation of participants' behavior which allows for formalized reasoning.

3. Organizational requirements in Tropos

Tropos is a goal-driven, agent-oriented software development methodology that covers a range of representations including organizational requirements at various levels of abstraction. Tropos provides a suitable framework for representing the business context that originates an interaction. Tropos models can be used to capture goals of distributed actors, the mutual dependencies that motivate them to interact, and the activities they undertake to fulfill their goals. Furthermore, the contract of the interaction can be captured using the formal counterpart of Tropos, Formal Tropos (FT). The behavioral obligations of participants can be specified in FT using formal logic. First, we introduce Actor-Dependency modeling in Tropos then we show how behavioral dynamics of the model are described using FT.

3.1. Tropos – actor-dependency modeling

Tropos builds on the strategic dependency modeling of the i* framework [6], originally intended to emphasize the "why" aspect of requirements of distributed actors. At the heart of i* are the concepts of actors, intentional elements, and dependencies. i* Actor-Dependency (AD) diagrams provide a notation for representing and analyzing the organizational requirements motivating the interaction between actors and the inter-dependencies that make the interaction possible. Figure 2 is an AD diagram for the high-level requirements motivating the medical interaction. An actor is an active entity that performs actions to achieve its goals. The patient, the MP, and the IC are all actors. Intentional elements include goals, softgoals, tasks, and resources. Intentional elements can either be internal to an actor or define dependencies whose fulfillment is delegated to other actors. An actor may depend on another for fulfilling a goal, performing a task, or making some resource available [7]. A goal is a state of the world desired by one of the participants. For

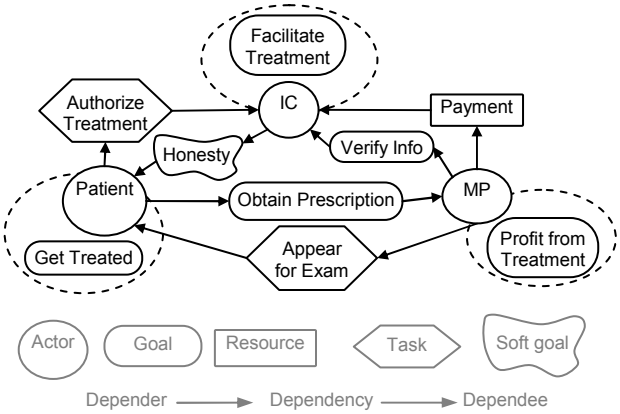


Figure 2. AD diagram for medical interaction

example, the "Get Treated" goal represents the patient's desire to get cured from an ailment. A softgoal represents an objective with no clear-cut satisfaction criteria. The IC's expectation that the patient does not abuse the insurance is modeled via an "Honesty" softgoal. A task is an abstraction of a course of action with well-defined pre- and post-conditions. The activity performed by the patient to visit the MP's office is represented by the "Appear for Exam" task. A resource is an informational or physical entity. For example, the "Payment" resource represents the compensation that the MP gets from the IC in return for providing services to the patient.

AD models can be successively refined into detailed models that describe the interaction more concretely [8]. In the process, goals are refined into sub-goals and eventually into tasks. Tasks can be further refined into sub-tasks that are either implemented by software or carried out by a human agent. Softgoals don't have clear-cut achievement criteria and will still exist in the refined model [4]. Figure 3 is a refinement of Figure 2 where model elements internal to an actor are refined inside the dotted circle corresponding to that actor. Each actor takes responsibility for carrying out their internal tasks. For example, the "Get Treated" goal was refined into tasks to get an authorization from the IC followed by getting a

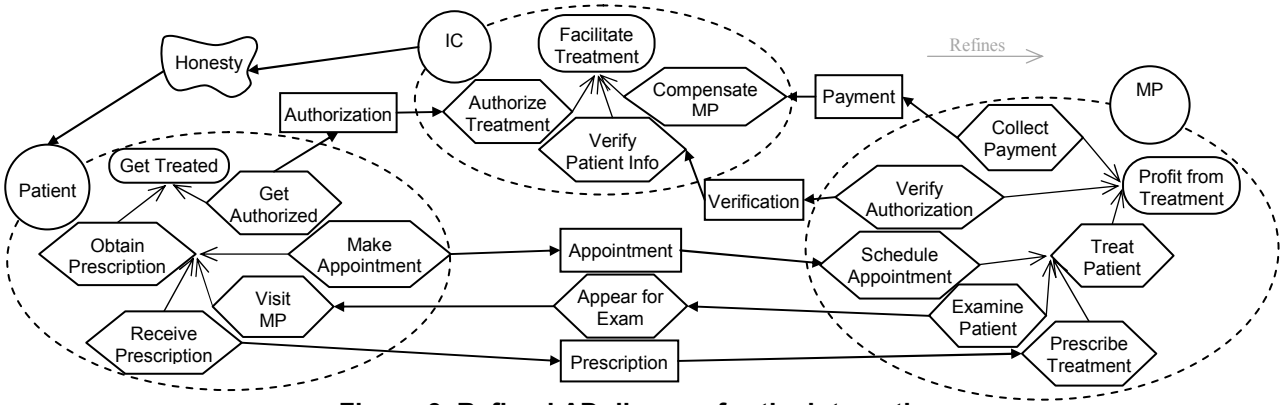


Figure 3. Refined AD diagram for the interaction

prescription from the MP. The latter is further refined into tasks for setting up an appointment followed by visiting the MP and then receiving a prescription from the MP. Ordering of tasks is not represented in the diagram to reduce clutter. In addition to detailing the activities involved in the interaction, the refined model details inter-dependencies between activities. It can now be seen from Figure 3 that "Make Appointment" task relies on the MP's "Schedule Appointment" for providing the "Appointment" resource.

3.2. Formal Tropos – behavioral modeling

FT allows for extending AD models with formal annotations for precisely describing the behavior of model elements and the relations between them. Each task, goal, and resource in the model is represented as an FT class, of which many instances may be created during an "execution" of the model. FT classes and instances are analogous to classes and objects in object-oriented languages. An execution of an FT model specifies a possible progression of the corresponding choreographed interaction at runtime. Figure 4 shows the FT specification for the "MakeAppointment" task class and the "Appointment" dependency class, parts of which can be automatically obtained from AD diagrams by applying some heuristics [4].

Each class has a list of attributes which hold the state of instances of that class as well associations with other instances in the model. For example, the "Appointment" class has an "ailment" attribute that specifies the type of ailment the patient suffers from and a "makAp" attribute that references the associated instance of "MakeAppointment" class. The special attribute "Actor" associates an intentional element internal to an actor with the actor who performs it. For example, the patient is the actor for "MakeAppointment". The special attributes "Depender" and "Dependee" represent the two participants in a dependency class.

Dependency Appointment
  Depender Patient
  Dependee MP
  Attribute
    constant makAp: MakeAppointment
    constant ailment: AilmentType
  Invariant makAp.actor = depender
  Invariant ailment = makAp.ailment
  Creation condition ¬**Fulfilled**(makAp)
  Fulfillment condition ∃ sa:SchedulApp
    (sa.actor = dependee ∧ ailment = sa.ailment ∧ **Fulfilled**(sa))

Task MakeAppointment
  Actor Patient
  Attribute constant ailment: AilmentType
  Creation condition ¬**Fulfilled**(super)
  Fulfillment condition ∃ a:Appointment
    (a.depender = actor ∧ a.makAp = self ∧ **Fulfilled**(a))

Figure 4. Sample FT specification

FT classes declare constraints that describe valid behaviors of the model using typed first-order linear-time temporal logic. An invariant constraint must hold throughout the lifetime of any instance of the class declaring it. For example, the "Appointment" dependency class specifies that its depender is always the same as the actor in the associated "MakeAppointment" task. More relevant to our purposes is that FT specifies the lifecycle of intentional elements by defining circumstances in which they arise and conditions that lead to their fulfillment. Creation and fulfillment conditions of a class define when an instance of the class is created (instantiated) and when it becomes fulfilled. The creation of the goal or a dependency is interpreted as the moment at which the actor begins to desire the goal or need the dependency to be fulfilled. For a task, creation is the moment when the actor starts to perform the task. The creation condition has to be satisfied for an instance of a class to be created. For example, an "Appointment" dependency will be created if there is an instance of "MakeAppointment" task that needs to be fulfilled.

Fulfillment condition marks the end of the lifecycle of an intentional element. The meaning of the fulfillment condition depends on what class declares it. Fulfillment condition should hold whenever a goal is achieved, a task is completed, or a resource is made available. For example, the "MakeAppointment" task is fulfilled when the associated "Appointment" dependency has been fulfilled (i.e. the appointment information was received by the patient) whereas an instance of "Appointment" is fulfilled when the MP has completed the task of scheduling an appointment. Note that an instance may refer to itself using the keyword "self" and may refer to the intentional element of which it is a sub-element using the keyword "super".

4. Deriving conversations from requirements

Central to our proposal is an approach to derive customizations of a choreography description from customizations made to organizational requirements. In order to achieve that systematically, we need to relate requirements concepts to those of choreography. An immediate observation on Figure 2 is that each actor in the AD diagram is operationalized into a role in a choreography description but other relations are not readily obvious. In what follows we will argue that the lifecycle of a choreographed conversation can be tied to that of a corresponding organizational dependency instance. This alignment allows for straightforward derivation of choreography descriptions from the Tropos models. But first, we will present a classification of dependencies to help scope our discussion.

4.1. Classification of organizational dependencies

Dependencies can be classified in at least three ways: according to the type of the corresponding intentional element, the physical/logical nature of the dependency, and the mode of fulfillment.

According to the type of intentional element dependencies can be classified into: goal, task, and resource dependencies [7]. Goal dependencies are abstractions that get successively refined into task dependencies and/or resource dependencies, where the fulfillment of the operational dependency contributes to that of the goal. For example, the "Obtain Prescription" goal of Figure 2 was refined into "Appointment" and "Prescription" dependencies in Figure 3. Since choreography is concerned with operational descriptions, which include only task and resource dependencies, we will focus on relating conversations to these two types. The relation of conversations to goal dependencies may then be inferred from examining how the goals are refined, which we do not address here.

A dependency can also be classified as being either of a physical or an informational nature. From the point of view of the depender, a task or resource dependency is said to be fulfilled when the depender detects a transition in the state of the world at which a certain condition (i.e. dependency fulfillment condition) becomes true. How the depender detects the transition depends on the nature of the dependency. A physical dependency is satisfied when the depender has observed a physical occurrence that indicates the fulfillment of the dependency. A patient arriving at the MP's office for examination is an example of a physical occurrence that indicates fulfillment of "Appear for Exam" dependency.

On the other hand, an informational dependency is fulfilled when some required information has been made available to the depender by the dependee. In a message-oriented realization of the requirements, the information becomes available when a message sent by the dependee carrying the required information is received by the depender. For instance, the "Authorization" dependency is fulfilled when the patient receives a message containing an authorization number thereby indicating treatment was authorized. Similarly, the MP receives a message from the IC verifying the patient info thereby indicating that "Verify Patient Info" task was completed.

Physical activities that participants perform in the course of the interaction are not necessarily reflected in the choreography description in a direct way. Practically, one cannot require a patient to send some electronic message when she starts her car (or hops on a bus) to go visit the MP! Since choreography specifies only electronic messaging and not physical activities we only need to consider informational dependencies for alignment with conversations.

Finally, dependencies can be characterized by a "mode" [4] which can be either "achieve" or "maintain". The lifecycle of an "achieve" dependency ends when it is fulfilled, whereas that of a "maintain" dependency extends over many conversations and possibly also over many instances of the choreographed interaction. In what follows we will only address the "achieve" dependencies and leave the discussion of "maintain" dependencies for later work.

4.2. How dependencies motivate conversations

Intuitively, a participant initiates a conversation when interaction with another participant is required in order to satisfy some business need. By initiating a conversation the depender requests that the dependee perform some task or provide some informational resource. When the dependency has been fulfilled, the conversation terminates as it has served the purpose it was initiated for. To argue for this alignment we pose and answer these four questions:

4.2.1. Can a dependency be fulfilled without a conversation? By definition, an informational dependency is fulfilled when the depender receives the required information via a message sent by the dependee. Without receiving that message, the depender would not get the required information and the dependency will not be fulfilled. That message delivers the required information and terminates the conversation. Therefore, the fulfillment of informational dependencies has to be associated with an exchange that involves sending and receiving at least one message, and hence a conversation.

4.2.2. Can a dependency be fulfilled without having the depender initiate a conversation? Participants in a choreographed interaction are independent entities. There is no single control point and no globally-held state and each participant is responsible for their own state and internal flow control. Only the participant who requires some resource to be furnished or a task to be performed would know the point in time where this needs to happen. Hence, it is normally the depender who has to initiate a conversation. Furthermore, the depender typically has to provide information to the dependee without which the dependee cannot fulfill the dependency. For example, the MP provides the patient info to be verified when verification is requested from the IC.

4.2.3. Is there a reason other than the need to fulfill a dependency that motivates initiating a conversation? Every conversation is initiated to fulfill a certain business need that requires exchanging information between

participants. In absence of such a need a participant proceeds as an independent entity and does not interact with others as there is no requirement motivating message exchange.

4.2.4. Would a conversation terminate for a reason other than that the dependency was fulfilled? A conversation may terminate abnormally if one of the participants fails to fulfill their obligations by providing a wrong response or not providing the response in a timely manner [9]. A conversation may also terminate if the need to achieve the objective ceases to exist, i.e. the dependency has been fulfilled in some other way or the dependency is no longer required to be fulfilled, all of which we consider to be "exceptional" conditions. Otherwise, if a conversation was initiated to fulfill a business need it terminates when the need has been fulfilled. By definition, a conversation terminates when the last communication event relevant to achieving its objective has occurred.

4.3. The proposed alignment

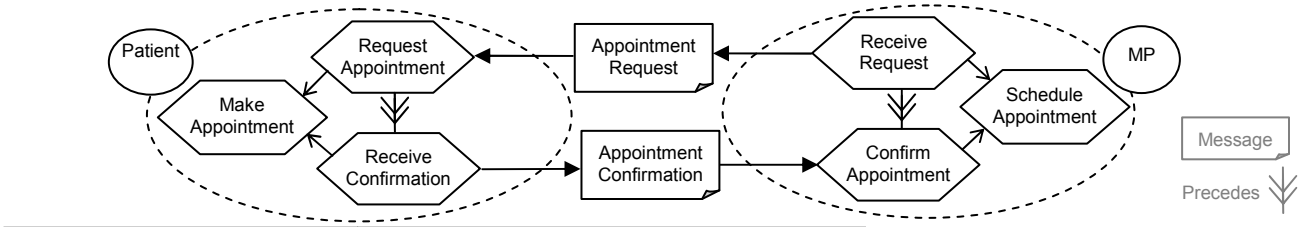
From the discussion above, we deduce that an informational task/resource dependency whose mode is "achieve" can be systematically operationalized into a single conversation which is initiated when the dependency is instantiated and terminates when the dependency has been fulfilled.

As an example, consider the operationalization of the "Appointment" dependency into a conversation depicted in Figure 5 using a service-oriented extension of Tropos[8]. "Make Appointment" and "Schedule Appointment" tasks at the ends of the dependency from Figure 3 are each operationalized into tasks for sending and receiving messages. In order to complete "Make Appointment" task the patient performs "Request Appointment" to send an "Appointment Request" message then performs "Receive Confirmation" task to receive the "Appointment Confirmation" message. Similarly, for every appointment the MP schedules they have to perform "Receive Request" followed by "Confirm Appointment". Note that solid arrows represent dependency rather than message flow, so the "Receive Confirmation" task depends on "Confirm Appointment" for receiving the "Appointment Confirmation" message.

From the diagram and the associated FT fragments the alignment between the dependency and the conversation is manifested as follows. First, the instantiation of "Make Appointment" triggers the instantiation of both an "Appointment" dependency, as shown in Figure 4, as well as a "Request Appointment" task. The instantiation of the latter results in instantiating (and sending) an "Appointment Request" message. Therefore, sending the message that initiates the conversation causally follows the instantiation of the dependency. Second, the conversation terminates when the "Appointment Confirmation" message is received, which also fulfills the "Receive Confirmation" task. At the same time, when "Receive Confirmation" is fulfilled the patient has received the information necessary for fulfilling the "Appointment" dependency. Therefore, dependency fulfillment causally follows the termination of the conversation. Similar diagrams and FT specification can be constructed for other conversations.

The general pattern of alignment is that once a dependency is instantiated the depender will initiate a conversation C by sending a message representing E_s of C. Eventually a message containing the information required for fulfilling the dependency is received by the depender, where the message represents E_f that terminates C.

In the general case, it takes more than a single request/response to fulfill a dependency. Realistically, in response to a request for an appointment the MP will provide a list of available time slots. By the time the patient selects a time slot and sends a request to reserve it the time slot may have already been taken. The patient will then have to request another time slot and it may take several messages back and forth before the dependency is fulfilled. Discussing conversation refinement possibilities is outside the scope of this paper.



Task MakeAppointment **Actor** Patient **Fulfillment condition** ∃ rc: ReceiveConfirmation (rc.super = self ∧ **Fulfilled**(rc))	**Task** RequestAppointment **Actor** Patient **Super** MakeAppointment **Creation condition** ¬**Fulfilled**(super) **Fulfillment condition** ∃ ar:AppointmentRequest (**Sent**(ar))	**Task** ReceiveConfirmation **Actor** Patient **Super** MakeAppointment **Fulfillment condition** ∃ ac:AppointmentConfirmation (**Received**(ar))

Figure 5. Appointment messaging tasks and associated FT fragments

Figure 6 relates E_s and E_f of each of the conversation types in the example to the instantiation and fulfillment of dependencies respectively. The lifetime of each dependency is represented by a horizontal line where the start of the line represents the instantiation of the dependency and its end represents dependency fulfillment. An arrow pointing upwards represents a message sent and an arrow pointing downwards is a message received, both from the point of view of the depender in each dependency. The horizontal axis represents time and the dotted arrows show the causality between events.

Note that all conversations in our medical interaction take place sequentially. If the patient was allowed to request available appointment time slots before getting an authorization, the "Appointment" and "Authorization" conversations may then take place concurrently. We plan to formalize this possibility in future work.

5. Customizing the requirements model

Having proposed an alignment between conversations and dependencies we can now perform customizations to the requirements model and use the alignment to derive resulting constraints on the choreographed conversations. The class of customizations to the requirements model we cover here are incremental modifications to the FT specification that further constrain the behavior of participants.

Performing the customization at the requirements-level benefits from the formality of the specification as well as from problem-domain knowledge captured in the requirements. We employ the formal semantics of the FT specification for systematically *finding* alternatives for representing the customization. On the other hand, we use the domain knowledge, including physical activities not captured in choreography, to guide the *selection* among the alternatives.

Revisiting the example where it is required to protect an MP from a slow-paying IC, we can now state the required customization in stakeholder-friendly problem-domain terms. At the requirements-level we can specify that it is required to "limit the number of outstanding payments" rather than "limit the number of active payment conversations" which we had to deal with at the choreography-level. An "outstanding payment" refers to an instance of the "Payment" dependency that has not been fulfilled.

One way to enforce this requirement is to customize the FT model by constraining the creation of a "Payment" when the specified limit has been reached. But better alternatives for enforcing this requirement may exist. We propose a technique for systematically finding and selecting from among alternatives for performing this kind of customization.

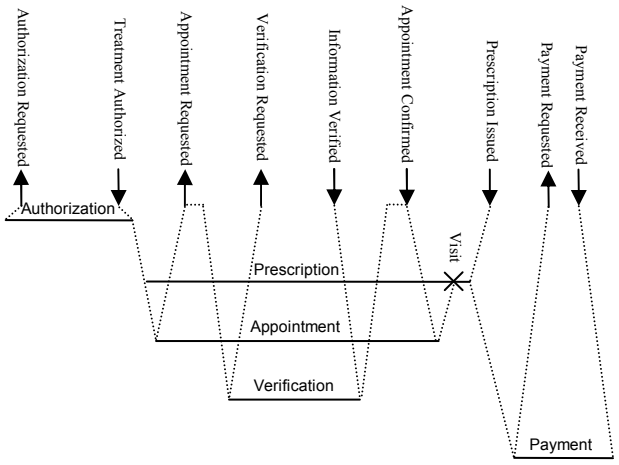


Figure 6. Conversation-Dependency Alignment

5.1. Finding customization alternatives

Several alternatives for constraining instantiation of a dependency may exist. We present a technique for finding these alternatives by traversing the FT model in a systematic way. Furthermore, we use the problem-domain knowledge captured in Tropos models for assessing the viability of each alternative.

Assume X is the dependency class whose instantiation is to be constrained. Assume the creation condition of X is Cr(X) and the fulfillment condition of X is Fi(X). Let Θ be the condition that needs to be true for the instantiation of X to be allowed. Let S be the set of all possible modifications to the FT model that enforce Θ, where enforcing Θ implies prohibiting the instantiation of X when Θ is false. We apply the following steps for traversing the FT specification to populate S:

- Add to S the alternative in which the original Cr(X) is modified to be $Cr(X) \wedge \Theta$
- If the fulfillment of an instance of a class Y is required for Cr(X) to be true, add to S the alternative where the original Fi(Y) is modified to be $Fi(Y) \wedge \Theta$
- Repeat the above for every class Z of which an instance is referenced in Cr(X)

5.2. Selecting a customization alternative

The viability of each alternative we add to S is assessed in light of available domain knowledge. The following factors are considered when assessing an alternative:

- Capability of participant: The participant responsible for enforcing Θ must have at their disposal the information required for detecting a violation.
- Risk to stakeholders: A participant *P* who would be negatively affected if Θ is violated requires strong

assurance that the condition is always enforced. This favors alternatives where the responsibility of enforcement lies on *P* rather than on another participant.

- Early detection: Alternatives that detect a violation early are obviously advantageous.
- Rationale: Alternatives that are easier to rationalize to stakeholders should be favored. We are currently working to make the rationalization less subjective.

5.3. Applying the technique

Applying our technique to the example at hand, X is "Payment" and Θ is WithinPaymentLimit(), which denotes that the IC has not yet reached the allowed number of open payments. FT fragments for the relevant elements of Figure 3 are given in Figure 7.

Applying our technique for traversing the FT model fragments the first alternative we find is to modify Cr("Payment") to be:

¬Fulfilled (cp) ∧ WithinPaymentLimit(this.dependee)

Even though this customization does prohibit the creation of "Payment", it is inappropriate from the point of view of the MP. This customization allows prescriptions to be issued for which the IC will not be billed (since no payments will be instantiated). The second step yields an alternative involving modifying Cr("Collect Payment") which suffers the same problem. The next alternative involves modifying Cr("Prescribe Treatment") which is still not satisfactory for the MP as it results in a model where a doctor wastes his time performing the "Examine Patient" task. The next alternative involves prohibiting the Cr("Examine Patient") which is not satisfactory to the patient since she will have already completed "Appear for Exam" task. At that point in the interaction the patient has already arrived physically at the MP's office and denying her the exam is unfair.

Continuing the traversal recursively we find several other unsuitable alternatives. In particular, all alternatives that lay the responsibility of keeping track of the number of outstanding payments on the patient are clearly rejected as this information is only known to the MP and IC.

Dependency Payment
Attribute constant cp: CollectPayment
Creation condition ¬**Fulfilled** (cp)

Task CollectPayment
Creation condition ∃ pt:PrescribeTreatment
(super = pt.super.super ∧ **Fulfilled** (pt))

Task PrescribeTreatment
Creation condition ∃ ep:ExaminePatient
(super = ep.super ∧ **Fulfilled** (ep))

Task ExaminePatient
Creation condition
∃ afe:AppearForExam (patient = afe.patient ∧ **Fulfilled** (afe))

Figure 7. FT fragments used in traversal

Fully traversing the model yields three potentially suitable alternatives:

1) Modify Fi("Verification") to prohibit the fulfillment of "Verification" dependency,
2) Modify Fi("Appointment") to prohibit the fulfillment of "Appointment" dependency, and finally
3) Modify Fi("Authorization") to prohibit the fulfillment of "Authorization" dependency.

Alternative #3 is superior to #1 in that it brings the interaction to an end earlier, thereby saving the patient's time by avoiding the wasted messaging involved in both #1 as well as #2. It can also be argued that alternative #3 is easier to explain to the patient. Getting rejected from the MP after being authorized for treatment by the IC, which is the case in alternatives #1 and #2, is harder to rationalize. On the other hand, alternative #2 can be argued to be superior because it lays the responsibility of enforcement on the main stakeholder of the customization, i.e. the MP. The MP will be negatively affected if the payment limit is exceeded and therefore it is desirable to have them be responsible for detecting the violation and ending the interaction.

Hence, it can be argued that an alternative that combines #2 and #3 is the best choice. This choice has the benefit of ending the interaction early while still allowing the MP to protect against an IC that does not fulfill their obligation of ending the interaction when payments limit is reached. In general, alternatives in S are not mutually exclusive and the desired customization can be achieved by applying one or more of the alternatives.

5.4. Deriving constraints on conversations

Having customized the requirements model we need to operationalize the customization to obtain a customized choreography description. Applying the alignment between dependencies and conversations we can deduce how the customization made to the requirements model is operationalized into constraints on conversations:

- Dependency creation to conversation initiation: A condition constraining the creation of a dependency prevents the depender, i.e. the participant responsible for initiation the corresponding type of conversation, from initiating a conversation.
- Dependency fulfillment to conversation termination: A condition constraining the fulfillment of a dependency prevents the dependee, i.e. the participant who sends the last message in the corresponding conversation type, from sending that message.

Applying the first rule to alternative #3 above we derive the additional constraint on the choreographed interaction: if the IC receives an "Authorization Request" when WithinPaymentLimit() is false the IC must not reply to the patient until the condition becomes true, i.e. until some payments have been made. Practically, rather than leaving the patient waiting indefinitely for a reply, the choreography specification may require the IC to provide some "rejection" reply when WithinPaymentLimit() is false, either immediately or after some specified timeout.

6. Related work

Choreography is drawing more attention especially in the areas of representation[10], generation of process skeletons [11], and verifying that the collective behavior of a set of distributed processes is compliant with a choreography description [2]. However, choreography customization has not been adequately addressed. Early work [12] on propagating changes in private interacting processes gives choreography a second-class treatment and also lacks the support for considering stakeholders requirements for selecting among alternatives.

The Nile System [13] promotes customization of choreographed interaction by capturing reusable semantic constraints on the interaction in a knowledge base. However, the whole approach is specific to RosettaNet and its applicability is limited to XML representations.

Most of the work addressing customization of service interactions has focused on adapting business process orchestrations rather than choreography descriptions. Rule-based approaches were suggested including [14], but such descriptions have been found to be hard to operationalize [15]. An aspect-oriented approach was used to make processes easily adaptable [16] but, as most other approaches are, it is closely tied to WS-BPEL.

More importantly, these approaches focus on design and implementation technologies of the interaction. Little attention is given to the business needs of the participants and the organizational dependencies motivating the interaction, which are crucial for reasoning about the customization. Hence, we chose Tropos for our approach. Its organizational requirements models provide formality not found in current choreography technologies such as WS-CDL [17]. UML activity diagrams, although a popular choice for representing interactions, also lack the formality [18] and the capacity to represent stakeholders' intentions.

Tropos has been used to represent and validate requirements for service-oriented interactions [8] but the systematic derivation of choreographed conversations from requirements models has not been addressed. The techniques for finding and arguing about customization alternatives are also unique to our approach.

Finally, whereas our approach bridges two levels of abstraction, the approach in [19] maps between a business constraints language and a choreography language that are both operational event-based descriptions. Neither of the two languages is suitable for representing or arguing about stakeholders' goals.

7. Conclusions and future work

The need to apply a choreography specification in different contexts calls for systematic techniques for performing the required customizations. Several alternatives for achieving the desired customization will exist and we have to rationalize the selection from among them. To ensure the selected alternative meets the needs of the participants we have to consult problem-domain knowledge. As choreography is limited to operational messaging specification, we proposed performing the customization to organizational requirements models motivating the interaction. Organizational requirements embody essential problem-domain knowledge, including specification of physical activities not captured in choreography, which we used for reasoning about the customization alternatives in stakeholder-friendly terms.

Moreover, we employed formal behavioral descriptions of FT for systematically finding alternatives to represent certain kinds of customizations. Through a traversal of the FT model, our proposed technique yields a set of potential modifications that can be applied to capture the customization. We will investigate how the proposed traversal can be improved in order to make the resulting set of alternatives more complete.

Performing the customization to problem-level concepts has the side benefit of hiding peculiarities of the underlying choreography language. Nevertheless, there is a need to operationalize customizations made to the requirements model into constraints to be added to the choreography description. For this reason, we proposed an alignment between organizational dependencies among actors in a Tropos model and conversations between roles in a choreography description. We concluded that conversation initiation/termination corresponds to dependency instantiation/fulfillment. We applied the alignment to derive constraints on conversations from constraints on the lifecycle of corresponding dependencies. We presented a classification of dependencies that allowed us to limit the scope of our discussion in this paper to classes of dependencies that are most relevant to choreography. We are currently expanding the scope of the alignment to cover a wider range of relations between organizational requirements and conversations. In particular, we are currently addressing:

- How dependencies whose mode is “achieve” relate to conversations.
- How to determine from the FT specification that some conversations may take place concurrently.
- How failure semantics [9] of conversations affect stakeholders’ goals.
- Whether the pattern for a conversation, e.g. request-response; iteration; negotiation; etc. can be deduced from the organizational requirements.

References

[1] C. Peltz, "Web Services Orchestration and Choreography," *IEEE Computer*, vol. 36, pp. 46-52, 2003.

[2] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Model-Based Analysis of Obligations in Web Service Choreography," presented at AICT-ICIW’06, Guadeloupe, French Caribbean, 2006.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, "Tropos: An Agent-Oriented Software Development Methodology," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 8, pp. 203-236, 2004.

[4] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso, "Specifying and analyzing early requirements in Tropos," *RE Journal*, vol. 9, pp. 132-150, 2004.

[5] A. Barros, G. Decker, M. Dumas, and F. Weber, "Correlation Patterns in Service-Oriented Architectures," in *Fundamental Approaches to Software Engineering (FASE)*, 2007, pp. 245-259.

[6] E. Yu, "Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering," presented at 3rd IEEE Int. Symp. on Requirements Engineering, Washington D.C., USA, 1997.

[7] E. Yu and J. Mylopoulos, "Understanding “Why” in Software Process Modelling, Analysis, and Design," presented at 16th International Conference on Software Engineering ICSE'94, Sorrento, Italy, 1994.

[8] R. Kazhamiakin, M. Pistore, and M. Roveri, "A Framework for Integrating Business Processes and Business Requirements," presented at Enterprise Distributed Object Computing Conference (EDOC'04), Monterey, California, USA, 2004.

[9] F. Cristian, "Understanding fault-tolerant distributed systems," *Commun. ACM*, vol. 34, pp. 56-78, 1991.

[10] X. Zhao, H. Yang, and Z. Qiu, "Towards the Formal Model and Verification of Web Service Choreography Description Language," presented at 3rd International Workshop on Web Services and Formal Methods (WS-FM'06), Vienna, Austria, 2006.

[11] J. M. Zaha, A. P. Barros, M. Dumas, and A. H. M. t. Hofstede, "Let’s Dance: A Language for Service Behavior Modeling," presented at OTM (1), Montpellier, France, 2006.

[12] S. Rinderle, A. Wombacher, and M. Reichert, "On the Controlled Evolution of Process Choreographies," presented at 22nd International Conference on Data Engineering (ICDE’06), Atlanta, GA, USA, 2006.

[13] D. Trastour, C. Preist, and D. Coleman, "Using Semantic Web Technology to Enhance Current Business-to-Business Integration Approaches," presented at EDOC'03, Brisbane, Australia, 2003.

[14] B. Orriëns and J. Yang, "A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaborations," presented at IEEE International Conference on Services Computing (SCC), Chicago, Illinois, USA, 2006.

[15] W. M. P. v. d. Aalst and M. Pesic, "DecSerFlow: Towards a Truly Declarative Service Flow Language," presented at 3rd International Workshop on Web Services and Formal Methods (WS-FM'06), Vienna, Austria, 2006.

[16] A. Charfi and M. Mezini, "Aspect-Oriented Web Service Composition with AO4BPEL," presented at The European Conference on Web Service (ECOWS'04), Erfurt, Germany, 2004.

[17] A. Barros, M. Dumas, and P. Oaks, "Standards for Web Service Choreography and Orchestration: Status and Perspectives," presented at Business Process Management Workshops, Nancy, France, 2006.

[18] V. Vitolins and A. Kalnins, "Semantics of UML 2.0 Activity Diagram for Business Modeling by Means of Virtual Machine," presented at EDOC'05, Enschede, The Netherlands, 2005.

[19] A. Berry and Z. Milosevic, "Extending Choreography With Business Contract Constraints," *International Journal of Cooperative Information Systems (IJCIS)*, vol. 14, pp. 131-179, 2005.