Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

Data-aware automatic derivation of choreography-conforming systems of services

Pablo Rabanal[a], Jose A. Mateo[b], Ismael Rodríguez[a], Gregorio Díaz[b,*]

[a] Universidad Complutense de Madrid, Madrid 28040, Spain
[b] Universidad de Castilla-La Mancha, Albacete 02071, Spain

A R T I C L E I N F O

Keywords:
Automatic Web service composition
Formal specification
Web service choreography
Conformance
Data-aware algorithms
FSM

A B S T R A C T

We present two data-aware algorithms to automatically derive web service compositions from global specifications. We show that a natural projection oriented derivation does not work in general, since some issues arise when projecting the set of implementations from the global specification, and we show how our approach deals with them. We use the constructions of two well-known languages as basis to define our models. In particular, given a WS-CDL choreography, our algorithms automatically extract a set of WS-BPEL compliant processes such that the interaction among these processes reproduces the behavior depicted in the choreography. This is achieved by introducing some control messages which make services coordinate as expected. With respect to our previous work on this kind of derivations, the main improvement of the models and derivations given in this work is their *data-awareness*, that is, the introduction of variables within the model, which strongly improves the expressiveness of our previous model based on FSMs. As a result, the formal model is enriched with new constructions such as workunits, which enable the definition of a complex conditional behaviour, and the derivation algorithms are necessarily more complex and sophisticated. The new derivation algorithms are implemented in the new version of our public derivation tool DIEGO.

1. Introduction

In recent years, *Service-Oriented Computing* (*SOC*) has emerged as a new paradigm to build distributed systems as a composition of independent services in order to save time and money. Despite these services can run as stand-alone units to achieve a particular task, they are usually included within service compositions, and hence the necessity to define neat and succinct languages to define how the participants shall interact arises. In particular, the specification of a web service-oriented system involves two complementary views: *Choreography* and *Orchestration*. On the one hand, the choreography concerns the *observable* interactions among services and can be defined by using specific languages, e.g., *Web Services Choreography Description Language* (WS-CDL [34]) or by using more general languages like UML Messages Sequence Charts (MSC) or, more recently, using more specific visual languages like Business Process Model and Annotation 2.0 (BPMN [22]). On the other hand, the orchestration concerns the *internal* behaviour of a web service in terms of invocations to other services. It is supported, e.g., by WS-BPEL [3] (*Web Services Business Process Execution Language*), which is the *de facto standard* language for describing web service workflows in terms of web services compositions.

Many researchers have invested their effort to develop efficient algorithms to construct the local implementation of each participant given a global specification of the system (related works are discussed in Section 6). The main issue of such kind of derivations is the fact that *natural projection* [24] does not necessarily produce a set of services conforming to the choreography. Natural projection, as its name suggests, is a process in which the choreography is projected into n views (one for each participant). Each projection mimics the structure of the choreography, although each transition models *only* the behavior of the chosen participant in the corresponding transition of the choreography. We can observe how natural projection works and the problems related to it in the example depicted in Fig. 1. These systems show the communication flow of a common internet purchase process involving three parties, Customers, Sellers and Carriers, represented by X, Y and Z, respectively. In this example, a customer orders a units of a product and pays by either debit (*msg1*) or credit (*msg2*). Orders are then processed by the seller depending the type of payment used and the quantity of units ordered. Orders whose quantity exceeds 10 units are processed as preferential, unless it is a credit payment —in this case, the processing can be either preferential o regular depending on

* Corresponding author.
E-mail addresses: prabanal@ucm.es (P. Rabanal), joseantonio.mateo@uclm.es (J.A. Mateo), isrodrig@ucm.es (I. Rodríguez), gregorio.diaz@uclm.es (G. Díaz).

http://dx.doi.org/10.1016/j.csi.2017.03.001
Received 23 November 2016; Received in revised form 28 February 2017; Accepted 2 March 2017
Available online 14 March 2017

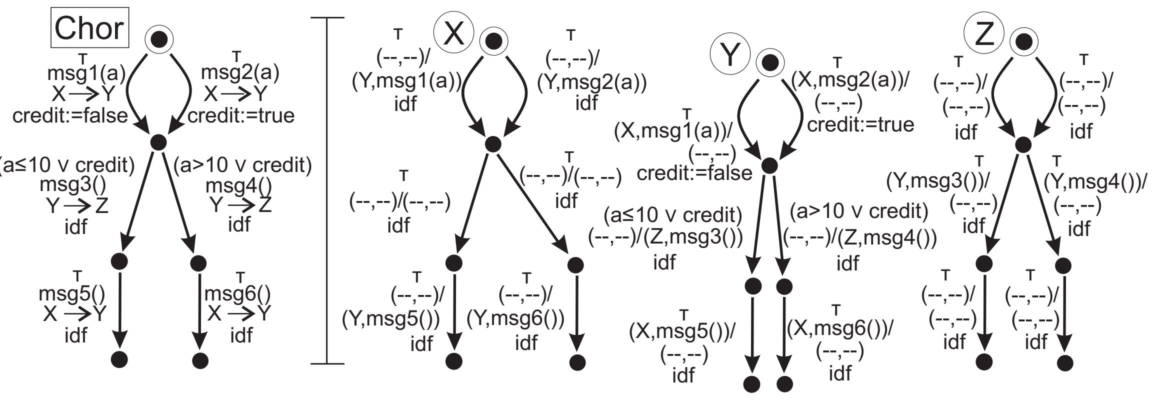

Fig. 1. Example of a natural projection.

internal information regarding the seller, as for instance overproduction. On the left side of the figure, the choreography *Chor* defines the required communication flow among these entities. Before we go on, let us informally introduce the models used here (they will be formally defined in subsequent sections). Each transition in the choreography *Chor* has the form $\bullet \xrightarrow[snd \quad adr]{cnd\ m\ trf} \bullet$, where • represents a state, *snd* is the sender of the message, *adr* is the addressee, *m* is the message (it could include *parameters*) and *cnd* is the condition that has to be satisfied to send the message. The attribute *trf* is the function which *transforms* the variables when the transition is taken. When *trf* does not modify the variable values, it is represented with *idf*, the identity function. For instance, the first left transition denotes that the message a is sent from service X to service Y. The condition is always true (⊤) and the assignment $credit{:=}false$ means that the variable *credit* is assigned to *false* when the transition is taken. Regarding the system orchestrations *X*, *Y*, and *Z*, shown at the right, each transition has the form $\bullet \xrightarrow{cnd\ (snd,im)/(adr,om)\ trf} \bullet$ stating that, if the service has a message *im* from *snd* stored in its input-messages buffer and the condition *cnd* holds, then it can send a message *om* to the service *adr*. As previously, the variables are updated via the function *trf*. Each orchestrator is provided with an input buffer in order to store temporarily the messages received. In particular (*snd*,*im*) can be (– – , – –), meaning that it is not required the presence of any pair in the input buffer of the service in order to take the transition. On the contrary, if (*adr*,*om*) is (– – , – –) then taking the transition does not cause any new message to be sent.[1] For instance, the first left transition in service *Y* means that this service processes the message $msg1(a)$ form the service *X* without producing a new message. Then, the variable *credit* is assigned to *false* when the transition is taken.

The aim of this example is to illustrate three problems that are inherent to a *natural projection* derivation, as well as the necessity to use alternative methods to fix them. On the one hand, services could take the non-deterministic choices of the choreography in a non-consistent way. In our example, the choreography at the second state has two possible paths (its left branch or its right branch) and each one depends on the action taken by service *Y* (sending $msg3()$ to *Z* or sending $msg4()$ to *Z*, respectively) as well as on the values of *a* and *credit*, representing the number of products and whether the transaction is performed on credit (*credit=true*) or debit (*credit=false*). In both branches, service *Y* sends messages to *Z*, but neither *Y* nor *Z* contact *X* afterwards to inform it about the action taken by *Y*. Services *Y* and *Z* will follow the same branch because they communicate. However, since *X* does not need to have any specific message in its buffer to take any of its two available transitions (both are labelled with (– – , – –)/(– – , – –)), *X* could follow the *opposite* branch as the one followed by *Y* and *Z*. In order to solve this problem, two main solutions have been proposed in the literature. First, Qiu et al [24] introduced the concept of *dominant role*, i.e. the service whose choice must be followed by other services. Authors argue that this dominant role can be selected by the choreographer in the design phase because, when a designer writes a choice involving multiple roles in a choreography, she/he has a clear idea about which role makes the real choice, whereas the others should follow its decision. An example of a customer-seller model is introduced. In that work, the customer is the dominant role, and the sellers are dominated by the customers' choice. So, there is a clearly differentiated dominant role in that case. However note that, in web services compositions, some level of fairness is typically assumed for the sake of decentralization. Consequently, our solution implements the more general notion of allowing all involved services to *participate* in the decision-making process. A second approach that has been widely followed by researchers consists in studying the set of restrictions and conditions that choreography representations must fulfill to make the natural projection work (e.g. [7,10,16]). In this case, the problem is precisely identified rather than solved. The choreographies fulfilling these requirements are called *well formed* and, essentially, they avoid situations where nondeterminism occurs. As we will see, some additional control messages will be added in our derivation in order to make services take the same choices, so *all* choreographies will be well formed in our setting. Consequently, our proposal does not regard the *realizability* of choreographies (i.e. whether each choreography can or cannot be converted into an equivalent orchestration), as those additional messages will let us realize *any* system described in our choreography formalism. In Section 6, a comparison of our approach with these works is presented.

The second problem related with these derivations is the presence of race conditions. A race condition arises when two messages sent in some order are received in different order. For instance, if we receive a message *a* from a service and next another message *b* from another service, then we cannot be sure that *a* was sent *before b* was sent, even if we know that the network is working properly. In this manner, a service can take a choice based on erroneous information, leading to unexpected behaviors. We will tackle this problem as follows. As we commented before, we assume that each service has a buffer which stores temporarily the received messages before being consumed. However, buffers do not guarantee that messages will be processed in the same order as they were sent, as the *communication medium* can create arbitrary delays for each message in such an asynchronous environment. As we will see later, we will introduce some additional control messages to make sure that the expected order is kept. Coming back to the natural projection example, even if service *X* selects the proper branch in the non-deterministic choice, it could progress to the last transition and send a message (either $msg5()$ or $msg6()$) *before* service *Y* has taken any of its two available actions. This is due to

[1] If none of them is (– – , – –) then the transition denotes processing some message and also, as a consequence, sending a message.