DIEGO: A tool for DerIving chorEoGraphy-cOnforming web service systems

Pablo Rabanal*,1, José Antonio Mateo†,1, Ismael Rodríguez*,2 and Gregorio Díaz†,2
∗ *Departamento de Sistemas Informáticos, Universidad Complutense (UCM),*
Madrid, Spain, Email: prabanal@fdi.ucm.es[1], *isrodrig@sip.ucm.es*[2]
† *Departamento de Sistemas Informáticos Universidad de Castilla-La Mancha (UCLM),*
Albacete, Spain. Email: jmateo@dsi.uclm.es[1], *gregorio@dsi.uclm.es*[2]

***Abstract*—We present a tool to automatically derive choreography-conforming web services systems. The user provides a specification that describes peer-to-peer collaborations of the observable behavior of parties from a global viewpoint, in our case WS-CDL documents, and the tool automatically extracts the particular behavior of each participant, more concretely, WS-BPEL documents defining the behavior from a local viewpoint. We implement two automatic methods (centralized and decentralized) that derive conforming systems even in cases where *projecting* the choreography into each service would lead to a non-conforming system. This issue is addressed by adding some control messages that make services interact as required by the choreography. Experiments where the number of exchanged messages is measured are presented, and strategies to reduce the number of these messages are discussed.**

***Keywords*- Web services composition; WS-CDL; WS-BPEL; Conformance; FSM.**

I. Introduction

In recent years, *Service-Oriented Computing (SOC)* has emerged as a new technology to build distributed systems as a composition of independent services. The web services architecture has been widely accepted as a means of structuring interactions among services. The definition 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 [22]) or by using more general languages like UML Messages Sequence Charts (MSC). On the other hand, the orchestration concerns the *internal* behavior of a web service in terms of invocations to other services. It is supported, e.g., by WS-BPEL [2] (*Web Services Business Process Execution Language*), which is the *de facto standard* language for describing web service workflow in terms of web services compositions.

In this paper we present DIEGO, a *tool for DerIving chorEoGraphy-cOnforming web service systems*. Given a choreography defined in (a subset of) WS-CDL, it automatically extracts a set of services defined in WS-BPEL such that the interaction of these services necessarily leads to the behavior defined by the choreography. Though DIEGO transforms WS-CDL into WS-BPEL, it internally works with an extension of *finite state machines* (FSMs). First, DIEGO transforms the WS-CDL choreography into a variant of FSM where involved services are explicitly identified. Next it extracts, from this FSM-based model, services defined by means of a different kind of FSMs variant where input buffers are used to support asynchronous communications. Finally, these FSM extensions are transformed into WS-BPEL.

The main problem arisen in the derivation of services from a choreography is the fact that the *natural projection* does not necessarily produce a set of services conforming to the choreography. We can easily observe this problem in the example depicted in Figure 1. On top of the figure we depict $Chor$, a choreography involving three parties X, Y, and Z. It defines the required communication flow among these entities. For instance, the first transition denotes that the message a is sent by service X to service Y. In addition, systems X, Y, and Z shown at the bottom of the figure are services derived from the choreography by directly projecting it into each involved role. In order to enable asynchronous communication, we consider that each service is endowed with a buffer to store incoming messages. In the figure, each service transition is labeled by a tag $(S1, m1)/(S2, m2)$ stating that, if the service has a message $m1$ from $S1$ stored in its buffer, then it can send a message $m2$ to $S2$.

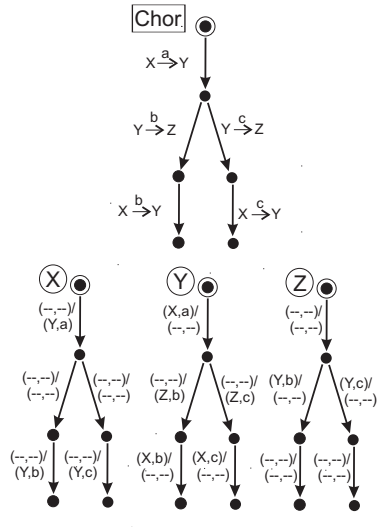

Figure 1. Example of a natural projection.

We can see that X, Y, and Z are structural copies of the choreography. In particular, each choreography transition is translated into a transition labeled with a communication action

(*reading* a message from its buffer and/or *sending* a message to another service). We write a null action $(--,--)$ in the service transition to show that the service does not read or send any message in the transition.

The aim of this example is to illustrate two problems that are inherent to this kind of derivation, as well as the necessity to use alternative methods to fix them. On the one hand, services could take non-deterministic choices of the choreography in a non-consistent way. In our example, the choreography can take two possible paths (its left branch or its right branch) depending on the action taken by service Y (sending b to Z or sending c to Z, respectively). In both branches, service Y sends messages to Z, but neither Y nor Z contacts X afterwards to inform it about the action taken by Y. Since Y and Z communicate, they will follow the same branch. However, since X does not need to have any specific message in its buffer to take any of its two available transitions (both are labeled with $(--,--)/(--,--)$), X could follow the *opposite* branch as the one followed by Y and Z. On the other hand, we have a *race condition* problem, i.e., the order of events defined by the choreography could be violated because a service reaches a subsequent state and overtakes the rest of services. Coming back to our example, even if service X selects the proper branch, it is possible that this service makes progress to the last transition *before* service Y has taken any of its two available actions. This is because service X just *sends* messages and it does not need any message in its buffer to evolve to the final state. This violates the choreography requirement that X must send its message only after Y has already sent its own message.

The derivation methods presented in this paper fix both problems by adding some control messages that make services interact with each other as required by the choreography. This work continues our previous works [11], [12]. In these works, FSM-based models for defining orchestrations and choreographies are presented. In these models, service communications are asynchronous, in the sense that received messages are stored in input buffers and the sender is not blocked after it sends a message. However, according to the proposed operational semantics, when a message is sent, it is immediately stored in the buffer of the destination service. Some conformance relations to decide whether a set of services necessarily produces the behavior defined by a choreography are presented, and derivation algorithms to automatically extract the former from the latter are given.

With respect to these previous works, this paper makes the following contributions. First, we present a *tool*, based on our formal framework, to automatically derive WS-BPEL orchestrations from WS-CDL choreographies. Let us note that our previous works present a *theoretical* framework for dealing only with *FSM-based* models. Our tool allows web service designers to rapidly obtain executable service prototypes from a choreography definition, which may help them in early stages of the development process (such as requisite analysis or architecture design) or even serve as executable skeletons for constructing the implementations from them. Moreover, the tool includes three additional modules which do not only serve as the basis for the current tool but will also be the root for our future developments: *parser*, *web services simulator*, and *graph viewer*. In addition, following the ideas of the (also theoretical and FSM-based) improved formal model given in [20], the WS-BPEL derivation algorithms of the tool produce correct services even if sent messages can be delayed any arbitrarily long time before they are received by their addressees. We also apply the tool to empirically analyze the performance of our derivation algorithms. We observe the number of additional messages exchanged by derived services in systems of different size, and we use this information to sketch improved versions of our algorithms where the number of these additional messages is reduced.

The rest of the paper is organized as follows. Next we comment some related works. In Section II we define our formal model as well as the *centralized* and *decentralized* derivation algorithms. The presentation focuses on informally introducing the main formal notions (detailed formal definitions and correctness proofs can be found in our previous/current works [12], [11], [20]). In Section III we describe our derivation tool. We introduce the basic concepts to understand its behavior and we illustrate its usage. In Section IV we present a case study to ease the understanding of the tool. Besides, we empirically analyze the performance of the tool. In Section V we briefly discuss some alternative strategies to improve it. Finally, we present our conclusions and future work.

A. Related Work

In this section we comment on several works related to our proposal. Regarding methods to derive services from a given choreography, let us remark that many works in the literature focus on studying the *conditions* that a choreography must fulfill for making the natural projection work (see e.g. [9], [4], [16]), rather than fixing the problems that make natural project not to work. Some works concern the projection and conformance validation between choreography and orchestration with *synchronous* communication. Carbone et al. [8] study the description of communication behaviors from a global point of view of communication and end-point behavior levels. Three definitions for proper-structured global description and a theory for projection are developed. Bultan and Fu [6], [5] specify web services as conversations using UML collaboration diagrams and analyze the conditions under which collaborations are realizable via finite state machines. Authors enhance the proposal by introducing a tool offering the possibility to use bounded buffers and reason about them. Their tool, called *cd2lotos*, transforms Collaboration Diagrams into LOTOS (Language of Temporal Ordering Specifications) [17]. The main issue in LOTOS is the inability to manage asynchronous communication, so authors needed to implement bounded FIFO queues. The main difference with our proposal is that cd2lotos does not show the representation of the local behavior of services, but it only checks if the choreography is

realizable. On the contrary, let us remark that our derivation method works regardless of the form of the choreography. This is achieved by introducing some additional control messages that make services interact as required by the choreography. Thus, *all* choreographies are realizable for us. Furthermore, our tool offers to users a wider range of features like graphical notion, BPEL derivation, etc.

In [19], Zongyan et al. identify and face the problems appearing when deriving an implementable projection from a choreography. Authors define the concept of restricted natural choreography, which must fulfill two structural conditions, and show that this kind of choreography is easily implementable. For dealing with projection issues in non-restricted choreographies, they propose the existence of the *dominant role* of a choice, i.e., the role that makes the decision. There are two crucial differences between this work and our proposal. On the one hand, the orchestration communication style is *synchronous* in their work, while we consider asynchronous communications, allowing us to explicitly distinguish between the times when messages are sent and the times when they are processed. On the other hand, the solution of the non-deterministic choices problem considered in [19] is based on *explicitly* adding extra information to the choice operator to identify the dominant role at the choreography level. On the other hand, in our derivation methods, a decision-making mechanism to make services coordinate in choice points is introduced at the orchestration level, but the choreography is not modified (and, still, derived services can perform all interactions defined in the choreography). In addition, we provide correctness proofs of our derivation algorithms (see [20]), not given in [19].

Finally, we compare our proposal with works in the domain of communication systems and reactive systems in general that address similar problems and use related formalizations. In [14] Gotzhein and Bochmann present a method to automatically derive the behavior of each party from the model of the distributed system. However, no correctness proof is given. Furthermore, authors make some assumptions about the communication medium: Separate input FIFO queues for each source are assumed, and the order in which messages are sent is preserved in their destination, i.e., message delays are not considered. In [3] local and non-local choices are discussed by Ben-Abdallah and Leue, but only the detection of problems in the system description is concerned, not the synthesis of the behavior of parties in such a way that these problems do not appear. In [15] the synthesis problem is considered by Gouda and Yu, but distributed systems can have only two parties, which strongly eases the task of providing a proper coordination between all existing parties.

Let us note that most of the previous works are only theoretical. On the contrary, in this paper we apply our theoretical FSM-based models and their (formally proved) derivation algorithms to develop a tool that automatically transforms WS-CDL choreographies into WS-BPEL orchestrations. To the best of our knowledge, DIEGO is the first tool that derives correct WS-BPEL systems of services without requiring any well-formed conditions to choreographies, under an asynchronous environment where messages can be delayed arbitrarily long times.

II. Automatically Deriving Choreography-Conforming Systems of Services under the presence of delayed messages

In this section we briefly introduce our formal model as well as our centralized and decentralized derivation algorithms for this model. The interested reader can find a formal presentation of these notions, as well as correctness proofs, in [11], [12], [20]. Next, we briefly describe our orchestration and choreography models. Let us note that both models are based on finite state machines, FSMs.

First we introduce the orchestration model. The internal behavior of a web service in terms of its interaction with other web services is represented by a finite state machine where, at each state *s*, the machine can receive an input message *i* and produce an output message *o* as response before moving to a new state s'. The set of services is identified by *ID* and, in order to provide asynchronous communication between services, all services are endowed with an *input buffer* to store received and not processed inputs. The actual configuration of an orchestration consists of two elements, its current state and the contents of its input buffer. Each transition explicitly defines which input *i* should be in the input buffer of the service in order to trigger the transition, as well as the required sender *snd* of this input. Besides, the transition specifies the output message *o* that the service will send when the transition is triggered, as well as the addressee *adr* of this message.

Contrarily to systems of orchestrations, a choreography model focuses on representing the interaction of services as a whole. Thus, a single machine, instead of the composition of several machines, is considered. Each choreography transition denotes a message action where some service sends a message to another one. A choreography machine consists of a set of states and transitions similar to the machines explained above, though each transition consists of just three elements: The message that is being exchanged, the sender, and the addressee. The actual configuration of a choreography consists just in the current state of its FSM model.

In both models, *traces* (i.e. sequences of events that can be produced by systems) are represented in a similar way, that is, each event is denoted by a message, a sender, and an addressee (snd, m, adr). However, in the orchestration model, events may refer to either the moments when messages are *sent* or the moments when messages are *processed* by their corresponding receivers (both moments differ because we consider an asynchronous environment). Based on the traces of choreographies and orchestrations, formal relations to determine whether a system of services necessarily produces the behavior required by a choreography are defined. In fact, several *conformance relations* are defined. On the one hand, the conformance of the system of services may be established in terms of *sending* times or *processing* times (or both). On

the other hand, systems of services may be required to be able to perform *all* non-deterministic paths defined in the choreography, or only at least *one* of them. Depending on these options, different conformance relations are defined.

Derivation algorithms are defined to automatically extract, from any given choreography model, a system of services such that it necessarily produces the behavior defined by the choreography with respect to each conformance relation. Unfortunately, the problems of natural projection presented in the introduction (inconsistent non-deterministic choices and race conditions) cannot be solved unless we introduce some additional *control messages* to be in charge of coordinating services as required by the choreography. In this way, systems conforming to the choreography (up to *original* choreography messages, i.e. ignoring additional control messages) can be derived. Two derivation methods, *centralized* and *decentralized*, are constructed.

The centralized algorithm is described next. New control messages are added to make all services follow the same non-determinism choices of the choreography, as well as making services wait until they have to act according to the order of events defined by the choreography. A new service, called *orchestrator*, is introduced. It is responsible of making all non-deterministic choices of the choreography. For each state s_j of the choreography model with several outgoing transitions, an equivalent transition is non-deterministically taken by the orchestrator (say, the *p-th* available transition). Next, the orchestrator takes several consecutive transitions to announce its choice to the other services. In each of these transitions, the orchestrator sends a control message a_{jp} to another service, meaning that the *p-th* transition leaving state s_j must be taken by the service. After (a) the orchestrator announces its choice to all services; and (b) the orchestrator receives a message b_{jp} from the addressee of the choreography transition (indicating that it has processed the message), the orchestrator advances to the next state. The same process is followed again in the new state. By adding the orchestrator, we ensure that all services follow the same non-deterministic choices of the choreography, and thus a system consisting of the orchestrator and the corresponding derived services conforms to the choreography (in fact, with respect to all proposed conformance relations).

In the decentralized model, the orchestrator role is removed and its responsibilities are distributed among the services themselves. Instead of using an orchestrator to select which transition is taken, we operate as follows: We sort all transitions leaving the current choreography state in some arbitrary way and we make the first sender choose between (a) taking any of the transitions where it is the sender; or (b) refusing to do so. In case (a), it announces its choice to the rest of services, playing the orchestrator role in this step. In case (b), it notifies its rejection to the second service, thus delegating the *decision making* process on it. In this case, the second service chooses either (a) or (b) in the same way, and so on up to the last sender (which is forced to take one of its transitions if all previous services refused to do so). In this alternative design, a service can receive messages indicating the necessity to take a specific non-deterministic choice from *several* services, and thus all corresponding transitions must be created. Let us note that, in this decentralized derivation method, more additional control messages are required, but non-deterministic choices are taken by services themselves rather than by an omniscient orchestrator, which may be more realistic in several scenarios. The correctness of both the centralized and the decentralized methods under the proposed formal model is proved in [11], [20].

During the implementation of the tool and the experimentation with real services derived by it, we realized that our underlying formal model might not be realistic in some scenarios. The operational semantics of the previous models assumes that sent messages are immediately stored in the input buffer of message addressees. In particular, this means that messages are stored in input buffers in the same order as they were sent. However, message delays could break this property. In order to remove the necessity of assuming this hypothesis, an alternative model is considered in [20] where, when a message is sent, it is not immediately stored at the destination service. On the contrary, the message may stay *in the communication medium* for any arbitrarily long time. A configuration of a system depends not only on the configuration of each service, but also on the multiset of messages that have already been sent by services but have not reached their destination yet. The operational semantics is modified to enable two kind of events: Either a message is sent by some service or a message in the medium reaches its destination (and is stored in the input buffer of that service). According to this alternative semantics, the former centralized and decentralized derivation algorithms are redefined, and their correctness is proved. In the centralized derivation, new additional acknowledgement messages from services to the orchestrator are added to solve reordering problems. In the decentralized derivation, due to the use of acknowledgement messages indicating that the "*decision-making token ring*" of the previous step is finished, it turns out that the previous decentralized derivation works correctly as well for the new semantics. The details can be checked in [20].

DIEGO is based on the theoretical model given in [20], where messages can be delayed any arbitrarily long times. Consequently, the transformations from WS-CDL to WS-BPEL developed in this paper are based on the FSM-based model transformations given in [20], which are proved to be correct with respect to this semantics. In addition, in Section V we will sketch and discuss refined versions of these algorithms where the number of control messages is reduced.

III. DIEGO: A TOOL FOR DERIVING CHOREOGRAPHY-CONFORMING WEB SERVICE SYSTEMS

In this section we present the tool that puts the derivation algorithms given in the previous section into practice. The tool DIEGO (see Figure 2) derives WS-BPEL orchestration services from a choreography defined in (a variant) of WS-CDL. The utility of DIEGO is twofold. From a business

point of view, the derived WS-BPEL services serve as early prototypes that may help web services designers in the analysis, design and development phases of the software life cycle. Moreover, since WS-BPEL services can be executed indeed, these prototypes can be taken as preliminary implementations that later will be refined by implementers. From a research point of view, being provided with an executable version of our derivation algorithms allows us to observe their behavior and better understand them, which helps us to create more efficient versions of our algorithms (this will be considered in the next section). The tool has been implemented by using Microsoft Visual Studio in language Visual Basic over .NET platform. DIEGO is available at *http://www.dsi.uclm.es/retics/diego/*.

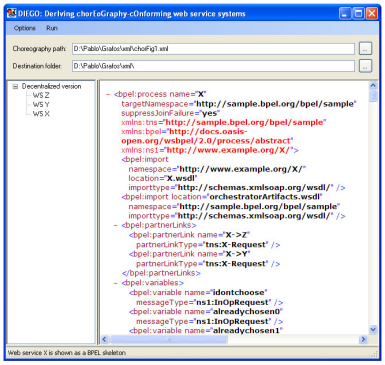

Figure 2. DIEGO main screen.

Let us describe the inputs that must be given to our tool. DIEGO receives, as input file, an XML file representing a *WS-CDL** choreography. We denote by WS-CDL* a simple modification of WS-CDL that explicitly supports the representation of choreography evolutions as *state transitions*. Let us note that WS-CDL does not offer a consistent way to model state cycles, because WS-CDL workunits need an entrance and a loop condition. We could use the arrival of messages as conditions, but this would limit non-deterministic choices. Thus, we opted to modify WS-CDL with a new tag *transition* to model state transitions. WS-CDL* is a modification of standard WS-CDL where interactions are labeled with the initial and final state, as it can be seen in the next example (in boldface):

```
<interaction name="a_interaction" operation="a"
  channelVariable="X2YChannel">
  <participate relationshipType="XY"
    fromRole="XRoleType" toRole="YRoleType" />
  <exchange name="aExch" action="request" />
  <transition initialstate="s0" finalstate="s1" />
</interaction>
```

As mentioned in the introduction, the tool includes three additional modules which help to investigate the behavior of the derivation algorithms. First, the *Parser* module transforms WS-CDL* into an internal FSM-based representation. Apart from this (which is actually necessary for the derivation), this feature will be used in future versions of the tool to provide useful information about possible incorrectness of WS-CDL* documents.[1] In the current version of DIEGO, the correctness of the WS-CDL* document is checked and, if it is not correct, then the tool generates an error message.

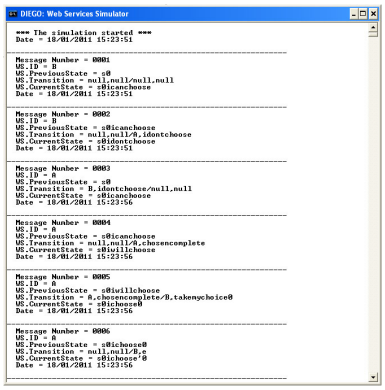

Figure 3. Messages displayed in DIEGO Web Services Simulator.

Secondly, the *Web Services Simulator* allows the user to observe the communications of derived services as if they were executed. Without this application, the only way to observe the behavior of derived services would be directly deploying them in a real platform and observing the traffic among participants by means of a sniffer. Some of these packet analyzers are Capsa [10], or the powerful testing tool SOAPUI [13]. Normally, this traffic consists on SOAP messages, so the sniffer output would have to be further analyzed by the user in order to extract the corresponding information. This difficulty is solved in DIEGO by providing the simulator. Once web services have been automatically derived from the choreography, DIEGO allows us to simulate these web services in order to observe how they work (see Figure 3). The result of the simulation can be observed in the Windows console and also is stored in a log file where communications among services are recorded.[2] DIEGO allows the user to set three different parameters in the simulation: The maximum possible delay for a message (measured in seconds; the user can also indicate that there is not limit for delays), the maximum simulation time of web services (measured in seconds), and maximum log size (measured in kilobytes). If the maximum log size is overflowed, the log is overwritten as needed.

Finally, a *Graph Viewer* is used to graphically observe the behavior of derived services. This information helps to understand how derived services solve race conditions and non-determinism problems, as well as how derived services could be manually refined to provide more efficient tailored solutions for some specific parts. Figure 4 depicts a graph sample of an automatically generated web service.

DIEGO uses another tool, previously developed in our research group, to design and create choreography inputs. The

[1] In particular, it will help users to construct syntactically correct choreographies by offering, in a log file, detailed information about detected errors.

[2] A future version of the tool, currently under development, will be multi-platform (in particular, web-based).

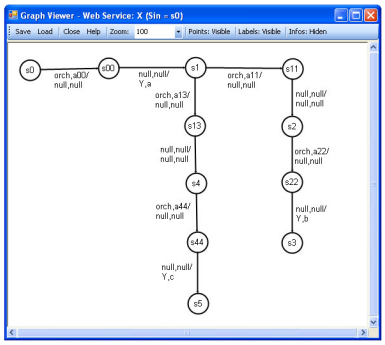

Figure 4. Example of web service graph.

tool WST [7] allows to create a UML Sequence Diagram as a model for services interactions and, after that, it directly translates this representation into WS-CDL. Thus, by integrating WST within DIEGO, users can provide DIEGO either with WS-CDL documents or with UML sequence diagrams (which, in turn, are automatically translated into WS-CDL by WST). Next, WS-CDL files are translated into the WS-CDL* format used in DIEGO. This is done by adding an initial and final state to every interaction, as discussed above. Once the WS-CDL* input is given, the user selects the corresponding algorithm version (centralized or decentralized) to automatically generate the set of web services that conforms to the choreography. On the left side of the interface, a tree (called *centralized* or *decentralized*, depending on the former choice) is displayed. By expanding this tree, the derived services appear and, by clicking once on any of them, the appropriate BPEL orchestration can be examined on the right side of the screen (see Figure 2). These abstract BPEL files can be used by developers as templates for programming the *actual* web services from them, in such a way that the control messages required for behaving as required by the choreography are given by DIEGO from scratch. The programmer can manipulate and enrich the abstract BPEL documents provided by DIEGO, so she can *execute* them in any BPEL executer, such as ActiveBPEL [1], Apache ODE [21], or Oracle BPEL Process Manager [18].[3] By double-clicking on any of the BPEL documents, the corresponding graph is shown. In the "Run" menu, the option "Simulate Web Services" invokes the "Web Services Simulator" commented before.

In fact, the tool supports the development of more efficient versions of our derivation algorithms. This is because derivation developers can examine the messages exchanged in order to identify redundant control messaging, which is a very useful feedback to develop strategies to reduce the redundancy. In fact, the derivation algorithms presented in this paper are a refinement of the algorithms presented in [11] (in particular, based on the improved theoretical model given in [20]) where some control messages were removed thanks to the experimentation with DIEGO. Further improvements of the derivation algorithms are commented in the next section.

[3]According to our development plan, our next version of DIEGO will be integrated with one of these BPEL engines in order to support the execution of derived services from a centralized interface in DIEGO.

IV. Case Study

In this section we present an empirical study of the number of messages exchanged among services automatically generated by applying the centralized and decentralized derivations provided by DIEGO. We show empirically that the number of messages exchanged among the services increases linearly with the number of web services involved in the choreography, as well as with the *depth* of the choreography.

Next we present our experiments. In the first experiment, we create choreographies that are similar to the one shown in Figure 1. We change labels of transitions of this choreography to create a new choreography with the same form where only two web services are involved. This choreography is called $cExp1.2WS$. Next we change one transition of choreography $cExp1.2WS$ by adding a new involved web service, thus three web services participate in the new choreography. The resulting new choreography is called $cExp1.3WS$. We continue changing the transitions and creating new choreographies with $4, 5, ..., 10$ involved web services (called $cExp1.4WS$, $cExp1.5WS$, ..., $cExp1.10WS$, respectively). For all of these choreographies, we execute our centralized and decentralized algorithms in DIEGO and run the Web services simulator to compare the number of messages exchanged. The results obtained are shown in Table I, where column *Version* shows the algorithm version employed, column *Choreography* shows the name of the choreography used in the experiment, and column *Num msgs* shows the number of messages exchanged between the services when running the simulation. Let us note that, in column *Num msgs*, sometimes an interval appears. This means that the derivation version under consideration needed different number of messages in different executions (let us note that the choreography non-determinism is simulated by making *random* choices in the simulator, and thus the number of messages upon termination may vary depending on the chosen path). The results presented show empirically that the number of messages exchanged among derived web services increases linearly with the number of services involved in the choreography.

In the second experiment, we create several choreographies similar to $cExp1.2WS$ where only two web services are involved in each one. The differences among these choreographies lies in the *depth* of the choreography (that is, the depth of the tree denoting the choreography form). We create choreographies similar to $cExp1.2WS$ with depth 3 called $cExp2.depth$=3, with depth 4 (as in $cExp1.2WS$) called $cExp2.depth$=4, and so on up to depth 11 ($cExp2.depth$=11). The results obtained are also shown in Table I. We observe that the number of messages exchanged among web services also increases linearly with the *depth* of the choreography.

The WS-CDL* documents used in the experiments and some other examples are available at *http://www.dsi.uclm.es/retics/diego/*.

As it can be seen in Table I, the difference between the number of messages exchanged in the centralized and the

Version	Choreography	Num msgs	Choreography	Num msgs
Centr.	cExp1.2WS	27	cExp2.depth=3	18
Decentr.	cExp1.2WS	27	cExp2.depth=3	18
Centr.	cExp1.3WS	36	cExp2.depth=4	27
Decentr.	cExp1.3WS	38-39	cExp2.depth=4	27
Centr.	cExp1.4WS	45	cExp2.depth=5	36
Decentr.	cExp1.4WS	48	cExp2.depth=5	36
Centr.	cExp1.5WS	54	cExp2.depth=6	45
Decentr.	cExp1.5WS	57-59	cExp2.depth=6	45
Centr.	cExp1.6WS	63	cExp2.depth=7	54
Decentr.	cExp1.6WS	68-70	cExp2.depth=7	54
Centr.	cExp1.7WS	72	cExp2.depth=8	63
Decentr.	cExp1.7WS	77-82	cExp2.depth=8	63
Centr.	cExp1.8WS	81	cExp2.depth=9	72
Decentr.	cExp1.8WS	86-91	cExp2.depth=9	72
Centr.	cExp1.9WS	90	cExp2.depth=10	81
Decentr.	cExp1.9WS	100-101	cExp2.depth=10	81
Centr.	cExp1.10WS	99	cExp2.depth=11	90
Decentr.	cExp1.10WS	109-110	cExp2.depth=11	90

Table I
EXPERIMENTS: MESSAGES EXCHANGED BETWEEN WS

decentralized versions is low, but it increases as the number of web services involved in the choreography raises. This issue has been observed in other experiments as well, though they are not presented here due to lack of space. Finally, let us comment that the number of possible choices available for each service does not influence the number of messages exchanged. In both the centralized and the decentralized derivation methods, services have to be informed about chosen paths, but the process of informing them does not depend on the number of choices available for each service at each point (only the choice actually taken is communicated to other services, and each one is communicated by using a specific message).

V. DISCUSSION: IMPROVING DERIVATION ALGORITHMS

In this section we sketch some alternative derivation methods where the number of additional control messages exchanged between services is reduced. These alternative methods, not given in previous papers, are the result of experimenting with DIEGO, as their goal is eliminating some of the control messages that were observed more frequently when simulating derived services with DIEGO. Formally defining these algorithms, as well as proving their correctness, is out of the scope of this paper. Our goal is illustrating that DIEGO actually supports the development of enhanced versions of web services in general, and better derivation algorithms in particular.

We observed that most of *avoidable* control messages in the derivation algorithms are sent to services that are not involved in the current choreography step (that is, in the natural projection, a $(--,--)/(--,--)$ label is attached to the corresponding service transition, according to the notation given in the introduction). For each service, any sequence of consecutive *null* transitions like that, departing at state s, and followed by a non-null transition (i.e. a transition that actually involves the service) reaching state s', can be replaced by a *single* transition from s to s'. We call it a *direct* transition. Let us note that, if transitions are replaced like this, each direct transition leads the service to immediately advance a *different* number of choreography steps. Thus, services must be informed not only about the transition to be taken at each time, but also about what choreography state this transition belongs to.

According to this change, in the *centralized* derivation, the orchestrator does not have to inform about its choices to services that are not involved in the current choreography state. On the contrary, it just has to inform about what transition it takes (and at what state) to the next services: (a) the sender of the message, according to the choreography transition; and (b) the receiver of that message. Besides, in order to avoid the races condition problem, the orchestrator must receive a signal from the receiver, indicating that the message has been received. Consequently, for each service, each *direct* transition denoting that the service *sends* a message is preceded by a new transition where the orchestrator informs the sender about its choice. Similarly, each direct transition where the service receives a message is preceded by the same kind of transition, and it is followed by a transition where it sends a signal to the orchestrator, informing that the message was received (so, the orchestrator may continue).

Adapting this idea to the decentralized derivation is a bit more complex. In this case, when the receiver of the previous message receives it, it initiates the decision-making token ring of the next choreography step by sending a message to the first service of the token ring of that step. Only services that can send a message at the current choreography state participate in the decision-making. When a service in the token ring chooses one of its choices, it informs about is choice to the rest of services in the ring, as well as to the receiver of the message it has to send. Next, it sends the message to the receiver. When the receiver receives it, it initiates the decision-making token ring of the next step, and so on.

These alternative solutions, which are currently under development in DIEGO, illustrate how the tool simulator can provide relevant feedback to design more efficient derivation algorithms. In particular, DIEGO might also help web service designers to try and test *tailored* messaging solutions, that is, solutions that are specifically designed to deal with their particular web services (rather than with the general case, as we do). Further developing these alternative algorithms (and proving their correctness) is part of our future work.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a tool that automatically extracts WS-BPEL orchestrations from WS-CDL choreographies. It is based on our previous algorithms to automatically derive FSM-based orchestration models from FSM-based choreography models. These algorithms derive correct systems of services even if there might be arbitrarily long delays between each message sending and the corresponding message reception. Besides, both the centralized and the decentralized algorithms produce correct sets of services regardless of whether the natural projection would not work due to race conditions or choices coordination problems. To the best of our knowledge, DIEGO is the first tool that allows users to derive correct sets of WS-BPEL services from a WS-CDL

choreography by automatically solving both races and choices problems (in an integrated way and without requiring any human interaction) in an asynchronous environment where messages can be arbitrarily delayed. The tool supports early prototyping and, moreover, the documents it constructs are valid skeletons for creating implementations where message sending/reception is given from scratch by the tool. This releases programmers from explicitly dealing with messages coordination issues.

We consider many lines of future work. The first line consists in formally defining the improved derivation algorithms sketched in Section V, proving their correctness as we did for previous algorithm versions, and empirically comparing their performance with the algorithms currently implemented in DIEGO, described in Section II. Regarding the tool itself, perhaps the most ambitious goal is integrating the tool with a wider-purpose tool supporting web services *testing*. Since our derivation algorithms produce sets of services that are capable of producing *any* interaction defined in the corresponding choreography (ignoring additional control messages), we could compare real web services systems against choreography specifications *indirectly*, i.e. by comparing the behavior of these real web services with the behavior of our automatically derived services. Besides, the capability of simulating the behavior of a system of BPEL services will allow us to systematically study the fulfillment of some properties in web services systems (e.g. absence of deadlocks, correct error handling, etc). Finally, we wish to extend our orchestrations and choreographies FSM-based models by introducing details such as parameters, variables, and temporal conditions, which would allow the tool to explicitly consider these details in the derivation.

REFERENCES

[1] Active Endpoints. ActiveBPEL, 2010. http://www.activevos.com/open-source.php.

[2] T. Andrews and F. Curbera. Web Service Business Process Execution Language, Working Draft, 2004. Version 2.0, 1.

[3] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217, pages 259–274. Springer, 1997.

[4] M. Bravetti and G. Zavattaro. Contract compliance and choreography conformance in the presence of message queues. In *International workshop on Web Services and Formal Methods (WS-FM'08)*. Springer, 2008.

[5] T. Bultan and X. Fu. Choreography modeling and analysis with collaboration diagrams. *IEEE Data Engineering Bulletin*, 31(3):27–30, 2008.

[6] T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. *Service Oriented Computing and Applications*, 2(1):27–39, 2008.

[7] M. Cambronero, G. Díaz, E. Martínez, V. Valero, and L. Tobarra. WST: A Tool Supporting Timed Composite Web Services Model Transformation. In *SIMULATION: Transactions of the Society for Modeling and Simulation International*, page To appear. 2011.

[8] M. Carbone, K. Honda, and N. Yoshida. Theoretical aspects of communication-centred programming. In *Electronic Notes Theoretical Computer Science*, volume 209, pages 125–133. Elsevier Science, 2008.

[9] H. Castejon, R. Braek, and G. Bochmann. Realizability of collaboration-based service specifications. In *Asia-Pacific Software Engineering Conference (APSEC'07)*, pages 73–80. IEEE Computer Society, 2007.

[10] Colasoft. Capsa, 2010. www.colasoft.com/capsa/.

[11] G. Díaz and I. Rodríguez. Automatically deriving choreography-conforming systems of services. In *IEEE International Conference on Services Computing (SCC'09)*. IEEE Computer Society, 2009.

[12] G. Díaz and I. Rodríguez. Checking the conformance of orchestrations with respect to choreographies in web services: A formal approach. In *IFIP International Conference FMOODS and FORTE (FMOODS/FORTE'09)*. Springer, 2009.

[13] Eviware. SOAPUI, 2005. http://www.soapui.org/.

[14] R. Gotzhein and G. von Bochmann. Deriving protocol specifications from service specifications including parameters. In *ACM Transactions on Computer Systems*, volume 8, pages 255–283. ACM, 1990.

[15] M. Gouda and Y. Yao-Tin. Synthesis of communicating finite-state machines with guaranteed progress. *IEEE Transactions on Communications*, 32(7):779 – 788, 1984.

[16] I. Lanese, C. Guidi, F. Montesi, and G. Zavattaro. Bridging the gap between interaction- and process-oriented choreographies. In *IEEE International Conference on Software Engineering and Formal Methods (SEFM'08)*. IEEE Computer Society, 2008.

[17] LOTOS. A formal description technique based on the temporal ordering of observational behaviour. IS 8807, TC97/SC21, 1988.

[18] Oracle. Oracle BPEL Process Manager, 2011. http://www.oracle.com/appserver/bpel_home.html.

[19] Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *International World Wide Web Conference (WWW'07)*. ACM, 2007.

[20] I. Rodríguez, G. Díaz, P. Rabanal, and J. A. Mateo. A centralized and a decentralized method to automatically derive choreography-conforming web service systems, 2011. Submitted for publication, available at `http://www.dsi.uclm.es/retics/diego/publications.htm`.

[21] The Apache Software Foundation. Apache ODE, 2011. http://ode.apache.org/.

[22] W3C. Web Services Choreography Description Language, 2004. `http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217`.