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

Pablo Rabanal, Jose A. Mateo, Ismael Rodríguez and Gregorio Díaz

Reference Manual Version 3.0

Research partially supported by projects TIN2012-39391-C04-04, and TIN2012-36812-C02-01.

- *P. Rabanal and I. Rodríguez are with Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain, 28040.*
  E-mail: prabanal@fdi.ucm.es;isrodrig@@sip.ucm.es
- *J.A. Mateo and G. Díaz are with Universidad de Castilla-La Mancha, Albacete, Spain, 02071.*
  E-mail: [joseantonio.mateo,gregorio.diaz]@uclm.es

Thanks to all users and reviewers for their fruitful feedback which can help us to improve this tool and the algorithms in which is based.

1 PREREQUESITES

As we have developed DIEGO in order to work on Windows platforms, it is necessary to install the .NET framework whether in your current Windows version is not included. Moreover, we have used Microsoft .NET Framework 4.5 during the implementation phase, so we encourage the users to setup this framework (or higher) in their computers. One can find the download page in the next direction: http://msdn.microsoft.com/en-us/netframework/aa569263

As well, we have tested DIEGO on Windows XP and Windows 7 without finding incompatibility errors. Let us note that you must run DIEGO as administrator in Windows 7 (click the right mouse button on the *.exe* file and, then, click on *Run as Administrator*).

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

We outline in this manual the tool DIEGO, which has been implemented by using Microsoft Visual Studio in language Visual Basic over .NET platform. Different versions of DIEGO, including the last one introduced here, are available at http://www.dsi.uclm.es/retics/diego/.

DIEGO was originally conceived to put our derivation algorithms into practice. DIEGO (see Figure 1) automatically derives a set of WS-BPEL services from a choreography defined in a variant of WS-CDL. The utility of DIEGO is twofold. On the one hand, the derived WS-BPEL services serve as early prototypes that may help designers in the analysis and design 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. On the other hand, being provided with an executable version of our derivation algorithms allows us to observe how our algorithms behave. By observing their behaviour, we have been able to create more efficient versions in contrast to the former versions. In fact, in [5] we used DIEGO to redesign the algorithms presented in [2], [4]. In particular, the initial algorithms were improved by removing some redundant control messages. These redundant control messages were discovered after observing the execution traces of services derived by DIEGO in some case studies.

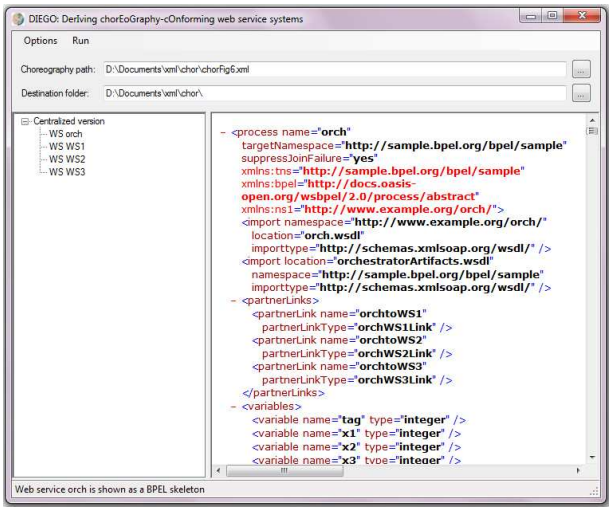


Fig. 1: DIEGO main screen.

Despite the fact that it is presented as a single tool, DIEGO can be considered an integrated composition of separated applications, where each one is used to carry out a particular mission. In particular, we can divide DIEGO into two complementary groups of tools. On the one hand, the *Derivation Engine* takes as input the Extended Finite State Machine (EFSM) model of the choreography, and it extracts the

EFSMs models of the derived web services. On the other hand, other modules are provided in order to help designers to debug their choreographies. Along the following lines, a more detailed description is presented.

As commented previously, some designers might not be willing to deal with EFSMs to model their choreographies as either they are not familiar with the formalism or they require more standardization. Thus, we had to decide first which language we should use to reach a wider audience. We chose WS-CDL since it is considered by many designers and researchers the *de-facto* standard to define choreographies. In technical terms, DIEGO takes as input a XML file representing a *WS-CDL** choreography. Notice that we denote by WS-CDL* a simple modification of WS-CDL that explicitly supports the representation of choreography evolutions as *state transitions*. Thus, we opted to modify WS-CDL with a new tag *transition* to model state transitions 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>
```

Variable declarations, conditions, and assignments to variables are defined conforming to the WS-CDL standard. Below we can see an example of how an integer variable is defined (*variable* tag) and initialized (*assign* tag). Our tool only handles integer variables for simplicity, although it can be easily extended to handle other types. The *roleTypes* tag defines the service owner of the variable. In the standard, multiple owners of the same variable can be defined, but DIEGO is limited for simplicity to a single owner. In the *assign* activity, value 2469 (given at *source expression*) is assigned to the variable *Track_WS* (*target variable*).

```
<variable name="Track_WS"
  informationType="xsd:int" roleTypes="WS" />
<assign roleType="WS">
 <copy name="Track_WS_assign">
  <source expression="2469"/>
  <target variable="Track_WS"/>
 </copy>
</assign>
```

In the next example we show how a conditional behavior (*guard* tag) is defined by using the *workunit* construction. In this example, *c* and *a* are variables, and *<* represents the less-than sign (<). Therefore, the guard of this example represents the expression $\neg(c = 0 \wedge a < 10)$. In our framework, any expression can be composed by using *not* and *and* operators (note that both form a complete set of logical operators).

```
<workunit name="opt1"
 guard="not(c=0,a&lt;10)" >
 ... (body of the workunit) ...
</workunit>
```

The following example presents how a variable assignment is done in an *interaction* where value $x + 1$ (*send variable*) is assigned to variable y (*receive variable*). Operations allowed in DIEGO are +, -, *, and /.

```
<interaction name="b_interaction" operation="b"
 channelVariable="X2YChannel">
 <participate relationshipType="XY"
  fromRole="XRoleType" toRole="YRoleType" />
 <exchange name="bExch" action="request" >
  <send variable="x+1"/>
  <receive variable="y"/>
 </exchange>
 <transition initialstate="s2" finalstate="s3" />
</interaction>
```

We offer two possible ways to model a choreography, either with WS-CDL documents or with UML sequence diagrams. In the latter case, UML sequence diagrams are automatically translated into WS-CDL by using WST [1], a tool developed by our research group. It provides a GUI to create UML Sequence Diagrams that can be automatically translated into its WS-CDL representation.

DIEGO also has a set of tools in order to help designers to build correct choreography specifications. This toolkit is composed by four independent modules.

First, the *Parser* module transforms WS-CDL* specifications into internal EFSM-based representations used as input for the *Derivation Engine*. In addition, the *Parser* provides useful information about possible anomalies of WS-CDL* documents in an error log file. If there exists some kind of mistake that it is not controlled by the tool, it will help users to construct correct choreographies by offering detailed information about detected errors in this log. In the current version, if the WS-CDL* document is not syntactically correct, then the tool generates an error message.

Fig. 2: Messages displayed in DIEGO Web Services Simulator.

Second, the *Web Services Simulator* allows to observe the interactions of derived services as if they were executed. The result of the simulation can be observed in the Windows console (see Figure 2), and it is also stored in a log file where communications among services are recorded. A future version of the tool, currently under development, will be multi-platform (in particular, web-based). DIEGO permits to set three different parameters for simulation purpose: The maximum possible delay for a message (measured

in seconds; the user can also indicate that there is no delay), the maximum simulation time (measured in seconds), and the maximum log size (measured in kilobytes). If the maximum log size is overflowed, then the log is overwritten as needed. Simulations can be executed either stepwise, upon termination, or upon some given point. These functionalities helped us to enhance the algorithms themselves. More importantly, without this functionality, the only way a user could evaluate the system behavior would be to directly deploy it in a real platform and observing the traffic among participants by means of a sniffer, which adds effort and time to the development process.

Third, DIEGO lets users check whether traces observed in specific executions of derived services are conformed with the requirements imposed by some conformance relations. The tool supports the execution of the system of derived services by manually or automatically (in particular, randomly) applying the language operational semantics. In the first case, the user chooses which operational rule is applied, whereas in the latter case DIEGO randomly chooses the execution path among all available execution paths. After the execution of a trace, the user can ask DIEGO to automatically compare it with those permitted by the choreography under different conformance requirements: in particular, *sending* (that is, with respect to moments when messages are sent), *processing* (w.r.t. moments when messages are processed), and *synchronization* (both).

Finally, with the help of the *Graph Viewer*, a graphical representation of the choreography and the derived services is given. These models help designers to understand how derived services solve issues such as race conditions, non-determinism coordination, and variables fetching. Moreover, graphical models can be used together with the simulator to develop more efficient service designs or even to envisage more efficient derivation algorithms for particular or general cases.

3 How to use DIEGO tool

Roughly speaking, DIEGO is used as follows. First, the user should select the path of the WS-CDL document corresponding to the choreography he/she wants to derive (see Figure 3). Note that some example choreographies are provided within the tool.

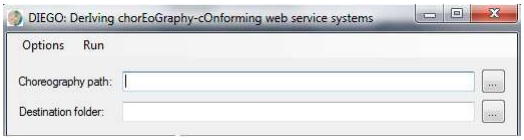


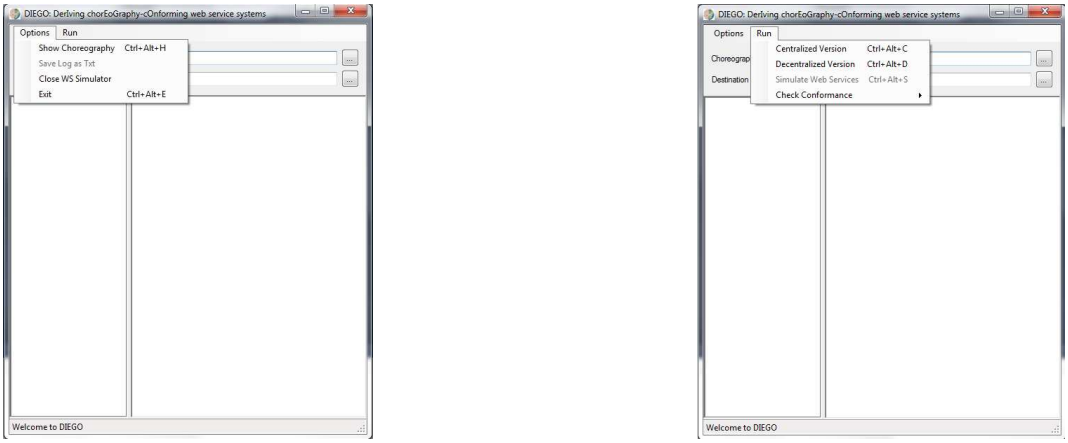
Fig. 3: Choreography path selection in DIEGO.

For checking choreography correctness, the user can press the button *Options* → *Show Choreography* (see Figure 4a) or the user can run the algorithm (centralized or decentralized version indistinctly). After that, a text file called *ErrorLog* is created in choreography folder. In this file, some errors or recommendations are presented.

If the WS-CDL* document is correct, the user can press the *Run* menu (see Figure 4b) executing one of the versions of the algorithm: Centralized or Decentralized.

After generating the services, XML files containing the BPEL description are created in the destination folder (see Figure 3), and an organizational tree (called *centralized* or *decentralized*, depending on the former choice) is displayed on the left side of the interface (see Figure 1). By expanding this tree, the derived orchestrations 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 1). By double-clicking on any of the tree items, the corresponding EFSM is shown.

Finally, it is mandatory to simulate the WS-BPEL services created in order to check if they are conform with the choreography. To this end, the option *Simulate Web Services*, in the *Run* menu, invokes the *Web Services Simulator* commented before (see Figure 5). The simulation of web services has different parameters. In Figure 6, these parameters are depicted. First, the user should decide to introduce some



(a) Options menu in DIEGO. (b) Run menu in DIEGO.

Fig. 4: Two menus of DIEGO.

delays when sending the messages (-1 means no delay). Moreover, user can control the simulation duration and the size of the log. This log can be stored in a text file by clicking the option *Save log as TxT* in the *Options menu*. Lastly, user can make a simulation step by step by marking the check box *Step-by-Step Simulation*.

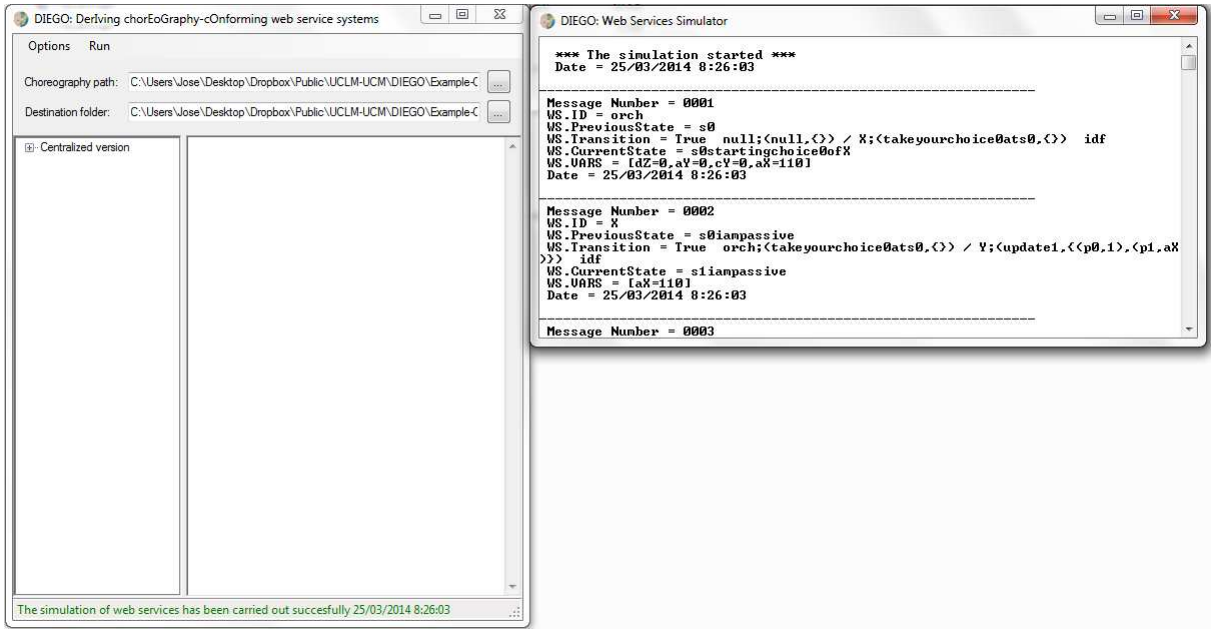


Fig. 5: Simulator screen in DIEGO.

Once the services have been simulated, the user can check whether the traces generated in the simulation are consistent with the choreography (*Run* menu, option *Check Conformance*).

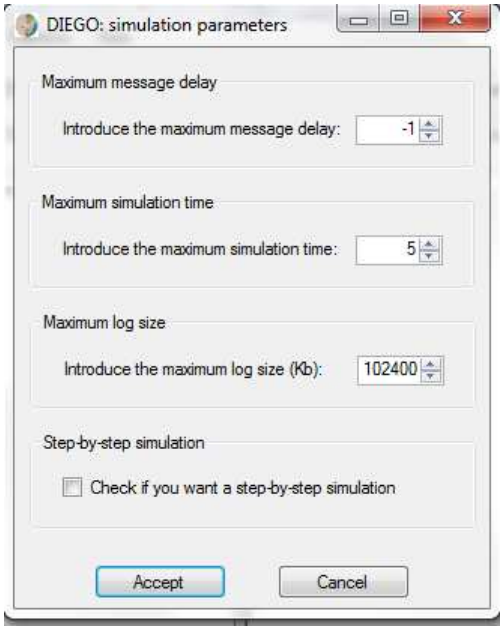


Fig. 6: Simulator parameters screen in DIEGO.

REFERENCES

[1] 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. Sagem, 2011.

[2] 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.

[3] R. Hailstone and R. Perry. *IBM and the Strategic Potential of Web Services: Assessing the Customer Experience*. IDC: Analyze the Future, 2002.

[4] P. Rabanal, I. Rodríguez, J. A. Mateo, and G. Díaz. Improving the automatic derivation of choreography-conforming web services systems. *Procedia CS*, 9:449–458, 2012.

[5] I. Rodríguez, G. Díaz, P. Rabanal, and J. A. Mateo. A centralized and a decentralized method to automatically derive choreography-conforming web service systems. *J. Log. Algebr. Program.*, 81(2):127–159, 2012.