Dynamo: Dynamic Monitoring of WS-BPEL Processes

Luciano Baresi and Sam Guinea

Dipartimento di Elettronica e Informazione - Politecnico di Milano
Piazza L. da Vinci 32, I-20133 Milano, Italy
`baresi|guinea@elet.polimi.it`

Abstract. Dynamo advocates that pre-deployment validation and testing are intrinsically inadequate for tackling the ephemeral and rapidly changing context in which service oriented applications are deployed. Validation must be shifted to run-time and continuous monitoring must be introduced. We propose a simple architecture that, through specific and simple annotations, allows for the automatic creation of instrumented WS-BPEL processes. These processes interact with a special-purpose proxy that enacts the monitoring activities and permits us to dynamically set the level of monitoring through use of a web-based interface.

1 Introduction

Run-time monitoring of functional and non-functional behavior is becoming an important and researched topic in the context of service-based systems. The extreme dynamism in service provisioning (services can change in a great number of ways) makes it difficult to grasp an overall knowledge of the system. This is why testing cannot foresee all the possible variations that may occur during execution, why validation must be shifted towards run-time, and why the idea of *continuous monitoring* must be introduced.

In [2], we propose a proxy-based solution for dynamic monitoring of WS-BPEL processes. Here we demonstrate such solution through its supporting framework called *Dynamo*. Within this work we introduce the idea of *Monitoring Rules*. These define run-time constraints on WS-BPEL process executions and are expressed using a specific language called WSCoL, inspired by the lightweight version of JML [3]. These rules are kept purposely separate from the process definition in order to avoid mixing the business logic with the monitoring logic. This is done because it helps the designer concentrate on solving the business problem without having to contemporarily tackle monitoring, and in order to allow different monitoring directives to be associated with a single process and therefore realize "personalized" monitoring. The monitoring rules are kept external to the WS-BPEL process definition up until deployment time, when they are weaved into the process. The result is an instrumented process which is capable of collaborating with our monitoring proxy (called *monitoring manager*) in order to verify the monitoring rules at run-time.

The monitoring manager is responsible for providing *dynamic monitoring*. We believe it to be of paramount importance to be able to tailor the degree of monitoring after a process has been deployed and, more specifically, at run-time. This is why we associate meta-level parameters to the single monitoring rules. These parameters are consulted

at run-time by the monitoring manager to decide if a monitoring activity is to be performed or ignored. They can also be modified at run-time through a specific graphical interface. Therefore, the real degree of monitoring depends on the values these parameters assume during the execution of the process, which in turn depend on the context of execution (when, where, and by whom the process is performed).

Since data can originate both within the process and outside the process, the monitoring manager is built modularly with respect to the *data collectors* it can use for collecting data at different levels of abstraction, and to the *data analyzers* it can use for verifying the monitoring rules. In this demo we make use of a data analyzer implemented using `xlinkit` [1], and of a dynamic invoker component that can be used as a data collector capable of retrieving data from any source that exposes a web service interface.

2 Context

This demo is based on a slightly modified version of the Pizza Delivery Company example originally proposed in [5].

Suppose that a client wants to eat pizza. With a WAP enabled mobile phone, the client dials the *Pizza Company* and, after suitable identification (`Authenticate Service`), his/her profile (`Profile Service`) determines which kinds of pizza the client likes. The `Pizza Catalog Web Service` then offers the client four kinds of pizza; after selecting the favorite one (Double Cheese), the client provides his/her credit card number (included in the client's profile) which is validated by the `Credit Card Validation Service`. If everything is okay, the client's account is debited and the pizza company's account is credited. Meanwhile, the pizza baker is alerted to the order, because after the selection the pizza appears in his browser, which is integrated with his cooking gear. Using the address contained within the user's profile, the `GPS Service` is called to get the coordinates of the delivery point. These coordinates are then passed onto a `Map Service`, which processes them and sends a map with the exact route to the pizza delivery boy on his PDA. The boy then only needs to deliver the pizza. In the mean time the client is sent an SMS text message on his/her mobile phone to alert about the delivery of the pizza within 20 minutes. This is done using the `SMS Service`.

To demonstrate the actual capabilities of Dynamo, we present two simple examples of monitoring rules and briefly explain their meaning. The first example is a post-condition to the operation `getCoord` published by the `GPS Service`. This operation receives an address as input and returns a UTM (Universal Transverse Mercator) set of coordinates. The set of coordinates comprises a number indicating the zone to which they refer, and two string coordinates called respectively easting and northing. Both are a seven character string made up of six numbers and one character. The final character is either an E or a N, depending if it represents and easting or a northing. The postcondition is that the easting and northing be syntactically correct . The following rule excerpt only checks the easting. Northings are checked in a similar way.

```
Location:
   type = "post-condition"
   path = "pathToInvokeActivity"
Parameters:
   priority = 2
Expression:
   @ensures easting.length()==7 &&
   easting.charAt(6)=='E';
```

The above states that the easting must be seven characters long and that it must end with a capital 'E'. A priority of 2 is associated with the post-condition. The meaning is that every time the process is executed with a global priority of 2 or less the post-condition is verified. On the other hand, if the global process priority is higher than or equal to 3 then the post-condition is ignored.

The second example is a post-condition to the operation `getMap` published by the `Map Service`. This operation takes a UTM set of coordinates and returns a JPEG image of the location. Since the map must be used on the pizza boy's small portable device, the resolution of the map that is returned must not be higher than 80 by 60 pixels. The following rule only presents the horizontal resolution constraint, while the vertical constraint is defined in a similar way.

```
Location:
   type = "post-condition"
   path = "pathToInvokeActivity"
Parameters:
   priority = 4
Expression:
   @ensures \returnInt(wsdlLoc, getResolution, 'image',
   GetImageResponse.GetImageReturn,
   HResolution) <= 80;
```

The above makes use of a special keyword `\returnInt` which can be called to interact with external collectors which are seen as web services. This makes it possible to write assertions that require information that is not obtainable from the process but that must be searched for elsewhere. In this case an external service is used to calculate the horizontal resolution of the map returned by the service. The operation `getResolution` is called on the service published at `wsdlLoc` and the returned `HResolution` is checked against the desired horizontal resolution of 80 pixels. A priority of 4 is associated with the post-condition.

3   Scenario

Dynamo works as follows[1]:

---

[1] In the following, we will use the concept of *priority* as a simplified version of our monitoring parameters.

– The first step is to design the unmonitored version of the process. This can be done using one of the many visual design tools already available on the market. In our demo we will use Oracle's BPEL Designer [4] which is available as an Eclipse Plugin.
– The next step is to import the unmonitored WS-BPEL process into Dynamo's Visual Annotation Tool (see Figure 1). By clicking on invoke activities (visualized as boxes), the tool provides information on the partnerlinks established between the process and the external web services. By clicking on the small black dots positioned above and below WS-BPEL invoke activities, it is possible to define either pre- or a post-conditions respectively. After all the monitoring rules have been defined, the tool creates the *monitoring definition file* automatically. The tool also requires that some initial global process parameters be added. They are used at runtime to determine what should be monitored and what not. In the demo, we will start by associating an initial global process priority of 4 to the monitored process. The priorities associated with the example monitoring rules are presented in the previous section.
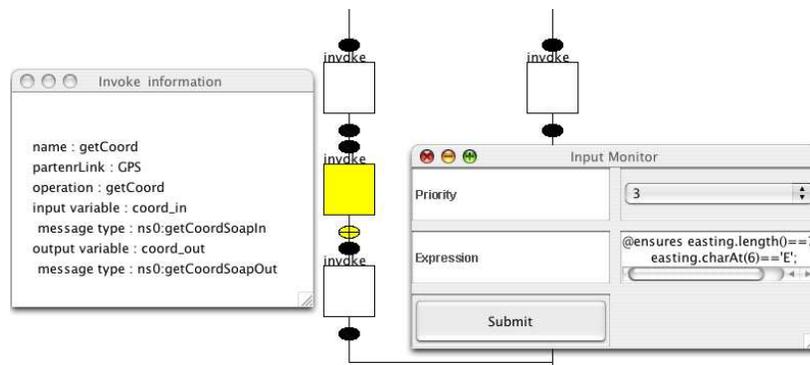

Fig. 1. Annotating the WS-BPEL process

– The third step consists in having Dynamo's BPEL[2] weave the rules contained in the monitoring definition file into the process, thus creating the monitored version of the WS-BPEL process. For each service invocation that has to be monitored (be it a pre- or a post-condition), the instrumented process calls the *Monitoring Manager* instead. This monitoring manager decides wether a monitoring rule has to be checked by confronting its monitoring parameters with the global process parameters. In our simple example, since the global process priority is 4, all monitoring rules with a lower priority are ignored. In this case the post-condition on operation `getCoord` is ignored and only the operation `getMap` is checked. In the demo we will show that the process does not terminate correctly because of an error arising somewhere during the execution. However, some of our monitoring activities are not being run due to the global process priority being too high. By changing

Fig. 2. Monitoring Manager Run-Time Interface

the global process priority, we can re-activate some monitoring activities that are switched off and discover where the problem lies.

– The global process priority of the process in execution can be changed by re-instrumenting the process or by accessing the monitoring manager at run-time through a web based interface. In the latter case (see Figure 2), we can choose the process from the list on the left, and set the new global process priority to 2 in order to activate the monitoring of the post-condition on operation `getCoord`.

– Re-running the process, we notice that the process once again does not complete. This time, though, it behaves differently. An error occurs while checking the post-condition on operation `getCoord`. The reason is that the easting returned by the operation is malformed. The process terminates in a more graceful manner and presents us with an error message that explains what is going wrong (see Figure 3).

4   Conclusions

This short demo briefly presents the main capabilities of Dynamo, our toolset for run-time monitoring of WS-BPEL processes. During the demo sessions, the framework will be presented in its entirety by means of the proposed pizza delivery scenario and more monitoring rules. Dynamo is available for download at `http://www.elet.polimi.it/upload/guinea`.
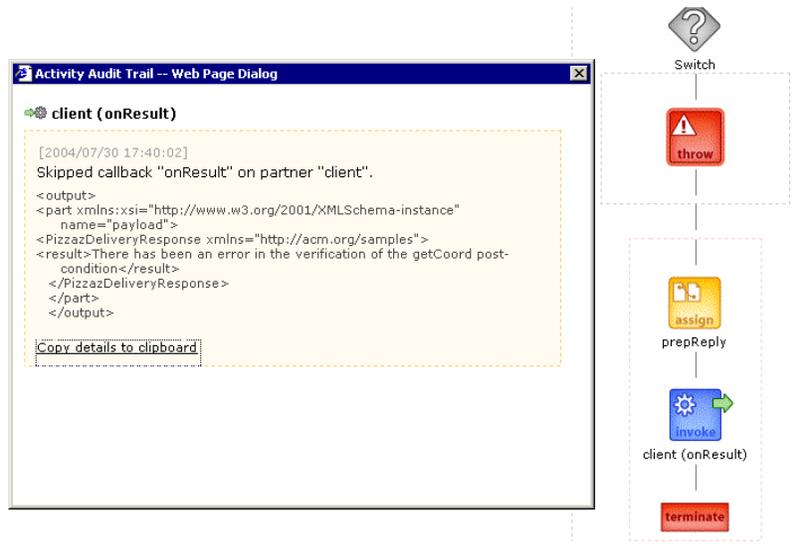
Fig. 3. Error in post-condition at run-time

References

1. XlinkIt: A Consistency Checking and Smart Link Generation Service. *ACM Transactions on Software Engineering and Methodology*, pages 151–185, May 2002.
2. L. Baresi, and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. *In Proceedings of the 3rd International Conference on Service Oriented Computing*, 2005.
3. Gary T. Leavens, Albert L. Baker, and Clyde Ruby. Preliminary Design of JML: A Behavioral Interface Specification Language for Java. *Department of Computer Science, Iowa State University, TR 98-06-rev27*, April, 2005.
4. Oracle. Oracle BPEL Process Manager. 2005. `http://www.oracle.com/technology/products/ias/bpel/index.html`.
5. IBM T.J. Watson Research Center. The Futuristic Pizza Company Example. 2004. `http://researchweb.watson.ibm.com`.