DynamoAOP - User Manual

Domenico Bianculli, Carlo Ghezzi

January 28, 2008

Contents

1 DynamoAOP overview

Dynamo-AOP[1] is a framework for monitoring functional properties of external services which a BPEL [2] process interacts with, to realize a composite service. It is based on the conceptual model proposed in [3], but, with respect to the original design, its architecture is based on the dynamic aspectization of the BPEL engine executing the monitored service compositions, achieved by using AspectJ [5] as an AOP [6] language.

2 Technical info

Provider USI

Introduction Framework for monitoring functional properties of external services with which a BPEL process interacts

Development status Prototype completed

Intended audience Service aggregators/providers that describe service compositions in BPEL

License GPLv3 (open source)

Language Java, AspectJ

[1] Part of this work has been developed by Lorenzo D'Ercole and Luca Gallupi, as part of their master theses, under the supervision of Luciano Baresi and Sam Guinea.

Environment (set-up) The following components need to be installed in order to run Dynamo-AOP:

- the Apache Tomcat servlet container (assumed to be installed in the directory `$TOMCAT-DIR` and running on port `7080`), available at `http://tomcat.apache.org/`
- the JBoss application server (assumed to be installed in the directory `$JBOSS-DIR` and running on port `8080`), available at `http://www.jboss.org/`
- the ActiveBPEL BPEL engine, available at `http://www.active-endpoints.com`
- (optional) a mail server, to support the `notify` recovery strategy.

The current version of Dynamo-AOP has been tested using Apache Tomcat ver. 5.5.23, JBoss Application Server ver 4.2, ActiveBPEL ver 2.1.
The following libraries are also required:

- ANTLRv2, available at `http://www.antlr2.org/`
- Apache Axis, available at `http://ws.apache.org/axis/`
- Apache XMLBeans, available at `http://xmlbeans.apache.org/`
- Apache Xerces 2, available at `http://xerces.apache.org/xerces2-j/index.html`
- Castor, available at `http://www.castor.org`
- Jakarta Commons Discovery, available at `http://commons.apache.org/discovery/`
- Jakarta Commons Logging, available at `http://commons.apache.org/logging/`
- CommonJ Timer and Work Manager for Application Servers, available at `http://dev2dev.bea.com/wlplatform/commonj/twm.html`
- Javamail, available at `http://java.sun.com/products/javamail/`
- JAXP, available at `https://jaxb.dev.java.net/`
- Jaxen, available at `http://jaxen.org/`
- JAX-RPC, available at `https://jax-rpc.dev.java.net/`
- JBoss EJB3, available at `http://labs.jboss.com/jbossejb3/`
- JBoss Web Services, available at `http://labs.jboss.com/jbosssws/`
- Saxon XSLT processor, available at `http://saxon.sourceforge.net/`
- StAX, available at `http://stax.codehaus.org/`
- Sun Java Streaming XML Parser (JSR 173), available at `http://java.sun.com/webservices/docs/1.5/sjsxp/ReleaseNotes.html`
- WSDL for Java API, available at `http://sourceforge.net/projects/wsdl4j`
- XML Pull Parser (XPP), available at `http://www.extreme.indiana.edu/xgws/xsoap/xpp/`
- XSUL, available at `http://www.extreme.indiana.edu/xgws/xsul/`

Platform Java 5

Download both sources and binaries are available at `http://plastic.isti.cnr.it/download/tools`

Documents see WP4 tools repository web site and deliverables D4.1 and D4.2

Tasks N/A

Bugs N/A

Patches N/A

Contact `domenico.bianculli@lu.unisi.ch`

3 Deployment

3.1 Install

To actually install the Dynamo-AOP monitoring framework you have to:

- copy from the Dynamo-AOP distribution `ConfigurationManager.jar`, `HistoricalVariable.jar`, `MonitorLogger.jar` in the directory `$JBOSS-DIR/server/default/deploy`.
- copy from the Dynamo-AOP distribution `ae_rtbpel.jar`, in the directory `$TOMCAT-DIR/shared/libs`, overwriting the original ActiveBPEL file.
- copy from the Dynamo-AOP distribution `invoker.jar` in the directory `$TOMCAT-DIR/common/libs`.
- install a copy of `antlr-2.7.6.jar`, `antlrdebug_1.0.0.jar`, `jaxb-api.jar`, `jaxb-impl.jar`, `jaxb-xjc.jar`, `jaxb1-impl.jar`, `mail.jar`, `jsr173_api.jar`, `saxon8-dom.jar`, `saxon8-jdom.jar`, `saxon8-sql.jar`, `saxon8-xom.jar`, `saxon8-xpath.jar`, `saxon8.jar`, `xbean_path.jar`, `xbean.jar`, `xmlpublic.jar`, `aspectjrt.jar`, `xpp3-1.1.3.4.M.jar`, `xsul-2.0.9_2.jar`, `stax-1.1.1.jar` in the directory `$TOMCAT-DIR/common/libs`
- install a copy of `axis.jar`, `commons-logging.jar`, `commons-discovery-0.2.jar`, `jaxrpc.jar`, `saaj.jar`, `wsdl4j-1.5.1.jar`, `castor-0.9.6-xml.jar`, `commonj-twm.jar`, `jaxen-1.1-beta-8.jar`, `ssaj-api.jar`, `xercesImpl.jar`, `saxon8-dom.jar` in the directory `$TOMCAT-DIR/shared/libs`.

3.2 Configure/Usage

1. launch the JBoss application server using the command `$JBOSS-DIR/bin/run.sh` and wait for the completion of the starting phase; check that JBoss is running by typing in your browser `http://localhost:8080`: you should see the start page of the application server.
2. launch the Apache Tomcat servlet container using the command `$TOMCAT-DIR/bin/catalina.sh run` and wait until the message "ActiveBPEL In-Memory Configuration Started" is displayed on the console. Check that Tomcat is running by typing in your browser `http://localhost:7080`. To check that ActiveBPEL is running, visit `http://localhost:7080/BpelAdmin`: you should see the ActiveBPEL administration tool. On it, click on Configuration and uncheck the box labelled "Validate Input/Output messages against schema".
3. deploy your BPEL process, as illustrated in ActiveBPEL user guide.
4. configure monitoring for the deployed process. This step assumes an interaction with the ConfigurationManager, which — in the first prototype of Dynamo-AOP — is done by directly accessing the API of the component, as shown in the "Dynamo Supervision Manager" application, also distributed within the framework (see next section).

4 Tutorial

In this section we will first review WSCoL, the language used to specify monitoring properties; then, we will describe the demo application bundled with Dynamo-AOP.

4.1 WS-CoL

WSCoL (Web Service Constraint Language) is the language used inside Dynamo-AOP to define monitoring properties; it is based on JML [7], with some conceptual and syntactical differences due to the adaption to the world of web services. Its main features are:

- Allowing to define and predicate on variables containing the data originating both within and outside the monitored BPEL process, and to retrieve data previously stored in a storage component.
- Using predefined variable functions for data manipulation.
- Using typical boolean, relational and arithmetic operators.
- Predicating on sets of variables through the use of the universal and existential quantifiers, and aggregate operators.

WSCoL allows to attach monitoring properties to the activities of a BPEL process that interact with external services. Properties can be pre- and post-conditions. Indeed, one can attach a pre-condition to an *invoke* activity, and a post-condition to an *invoke*, a *receive* and a *pick* activity.
All monitoring properties have three parameters:

- *priority*: it represents the "importance" of the rule and can be an integer ranging from 1 to 5. Each process can then define a threshold value that makes monitoring properties active or not, allowing for dynamically changing the amount of activities performed for monitoring the process.
- *validity*: it defines time constraints on *when* a monitoring properties should be considered.
- *trusted providers*: it's a list of service providers for which monitoring is not necessary.

In its simplest definition, a monitoring property states relationships that must hold between variables. WSCoL supports three kinds of variables:

- *internal*: an internal variable corresponds to a datum that originates within the process being monitored. Usually, WSCoL internal variables define a form of *data extraction* from complex BPEL variables, by using XPATH [8] expressions. For example, to extract the value of a sub-element `easting` from the sub-element `start` of the element `parameters` of a complex BPEL variable named `getRouteIn`, one can use the notation `$(getRouteIn/parameters/start/easting)`, i.e. by concatenating the variable name prefixed by a dollar sign, with the XPATH expression matching the value of interest.
- *external*: An external variable indicates a monitoring datum that originates outside of the process in execution, such as a contextual datum. WSCoL assumes that the data source of an external variable can be queried through a web service interface and provides a function to invoke it: `(return<X> (W, O, Ins, O` where
  - `X` indicates the XSD type returned by the data source web service; it can be `Int`, `String`, `Bool`.
  - `W` represents the location of the WSDL of the data source web service.
  - `O` represents the name of the web method of the data source web service.
  - `Ins` represents the input message that one has to send when calling the data source web service.
  - `Out` represents an XPATH expression indicating the data extraction to apply to the output message returned from the data source web method, to get the desired value.
- *historical*: Historical variables consist of monitoring data that are related to previous activations of the Dynamo-AOP framework, related either to other processes or previous steps in the same process. Historical variables are defining using the `store` construct, as follows: `store $east_historical=($getRoute` Previously stored historical variables can be retrieved using the `retrieve` function `(retrieve(pID,uID,iID,kID,type,alias,n))` where `pID` identifies the process family, `uID` is the user-id of the user who run the processes, `iID` identifies the instance within the process family, `kID` identifies an *invoke* activity in the process, `type` specifies if the historical variables was stored in a pre- or post-condition; `alias` is the name of the variable used in the `store` operation, `n` is the maximum number of results that should be returned by the query.

Moreover, variables may be aliased. This feature is provided both for allowing for less verbose expressions and for referring multiple times to a variable, whose value has been collected only once. Aliases are defined using the `let` keyword as shown below:
`let $east=($getRouteIn/parameters/start/easting);.`

Variables can be manipulated using data-type specific functions, invoked on the variable using the dot notation. Numeric functions include `abs()`, `ceiling()`, `floor()` and `round()`. The available string functions are: `compare(string)`, `replace(pattern, replace)`, `substring(start, len)`, `length()`, `contains(string)`, `startsWith(string)`, `endsWith(string)`. For example, to get the length of the string value referred by the alias `east`, one can write `$east.length()`.

WSCoL allows to use universal and existential quantifiers to express constraints over sets of value. The syntax is: `(quantifier $alias_name in range_def; constraint_def)`
where `quantifier` can be `forall` or `exists`. A universal quantifier indicates a parametric constraint that must be true for each and every value (at least one, in the case of the existential quantifier) the parameter can assume in a given range. `alias_name` and `range_def` defines respectively a variable alias and a finite range for the values it can assume; `constraint_def` defines the parametric constraint that must hold.

Aggregate operators allows to define assertions on set of values. The syntax is
`(operator $alias_name in range_def; assertion_def)`
where `operator` is one among `max`, `min`, `avg`, `sum`, `product`; `range_def` is a variable that returns a set of values, and `alias_name` is an alias that can be used as a parameter in `assertion_def`.

When a violation of a property is detected, some sort of recovery action should be performed. In our framework we have not investigated any specific recovery strategy but, by simplifying the work presented in [4], we just provides three simple primitives: `ignore`, which ignores the violation and allows the process to continue, `halt`, which stops the execution of the process, and `notify(msg, addr)`, which sends an email with the text `msg` to the recipient `addr`. These strategies can be structured using an `if-elseif-else` statement and/or boolean operators. For example, if a value is found to have a value below a certain threshold, we might want to differentiate the recovery strategy on the basis of the difference with respect to the threshold, as shown below:

```
if ($hRes < 80;)
   { halt() and notify("Low resolution, dba@localhost)}
elseif ($hres > 80; && $hRes < 120;)
  {ignore ()  and notify("Medium resolution, dba@localhost)}
else {ignore()}
```

The complete grammar of WSCoL is listed in the apppendix of this section.
Let's now see some examples of WSCoL properties.

```
Location: /process/sequence/invoke[@name='InvokeMap']
Supervision priority: 2
Monitoring rule:
   let $hRes=returnInt(
   'http://127.0.0.1:8080/ImageVerifierServiceBean?wsdl',
   'getHRes',
   $MapResponse/result,
   /Response/result);
   $hRes<= 150;
Recovering rule: {ignore()}
```

In this example, a property is associated to the *invoke* activity named *InvokeMap*. The `getHRes` web method of an external data collector service `ImageVerifierServiceBean` is called by passing it — as a parameter — the `result` element of the internal variable `MapResponse`. Once the service sends back its return message, the desired value is obtain through data extraction (`/Response/result`). This value is assigned to the `hRes` variable (actually, an alias), by means of the `let` statement. The actual property then checks if this variable is less than or equal to 150.
The recovery rule of this property is to ignore the fault and continue with the execution of the process. Moreover, a priority of 2 is associated with this property. The meaning is that every time the process is executed with a global priority of 2 or less the property is verified. On the other hand, if the global process

priority is higher than or equal to 3 then the property is ignored.
Another sample property is

```
Location: /process/sequence/invoke[@name='InvokeGPS']
Supervision priority: 4
Monitoring rule:
   ($CoordResponse/result/easting).length==7) &&
   ($CoordResponse/result/easting).endsWith('E');
Recovering rule: {halt()}
```

The above property states that the `easting` element (supposed to be of type string) of the `result` element of the `CoordResponse` internal variable, must be seven characters long and that it must end with the character 'E'. The recovery rule just makes the process execution terminate. Moreover, a priority of 4 is associated to this property.

4.2 Demo

In the software distribution, you will find a complete web application that can be used to see how the monitoring framework works.

The application contains a BPEL process, `PizzaDeliveryCompany`, which you should deploy in the BPEL engine; at the end of the deployment you should see the process listed in the Deployed Processes section of the ActiveBPEL control panel, as shown in Figure 1. Before using the demo application,

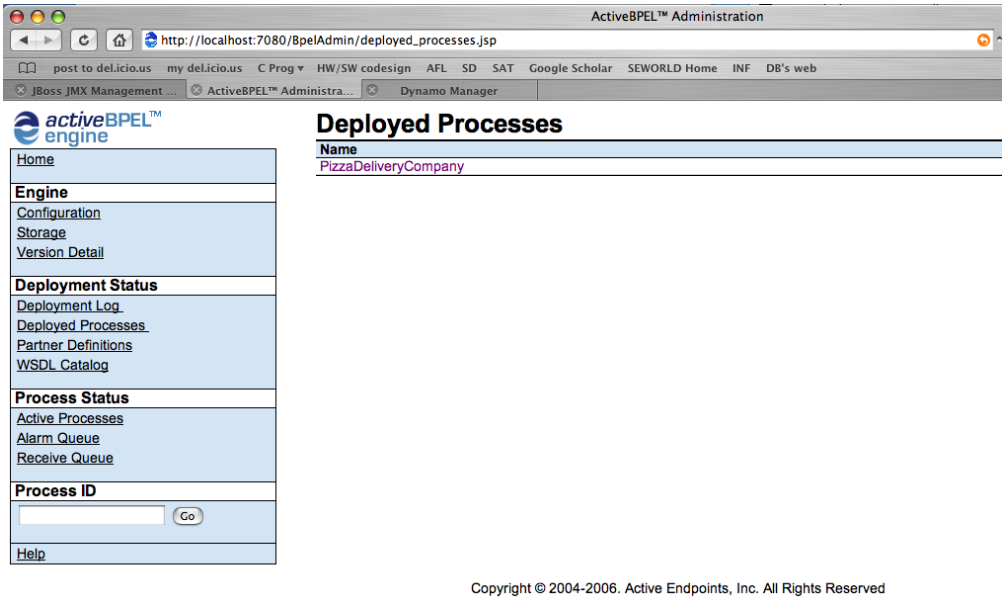

Figure 1: Process `PizzaDeliveryCompany` deployed successfully.

you should deploy some web services in the application server, by copying all the `*.jar` file from the `demo` directory to `$JBOSS-DIR/server/default/deploy`. In the same directory, you should also copy `dynamo.war`, which contains the "Dynamo Supervision Manager", a web application developed to to control the behavior of the monitoring framework; it can be reached at `http://localhost:8080/dynamo`.

The main page of this application contains two links: one to set the properties of the monitored processes and the other to access the Dynamo-AOP demo, also available directly at `http://localhost:8080/dynamo/DemoManager.jsp`. This page contains the various steps through which you can interact with the BPEL process. The execution of step #1 will attach some monitoring rules to the process, as shown in Figure 2. The execution of the monitor process can then be monitored on the output console of Tomcat, as shown in Figure 3. The structure and the priority of the rules attached to a monitored process can be modified using the "Dynamo Supervision Manager", as shown in Figure 4.

Figure 2: The result of inserting two monitoring rules.

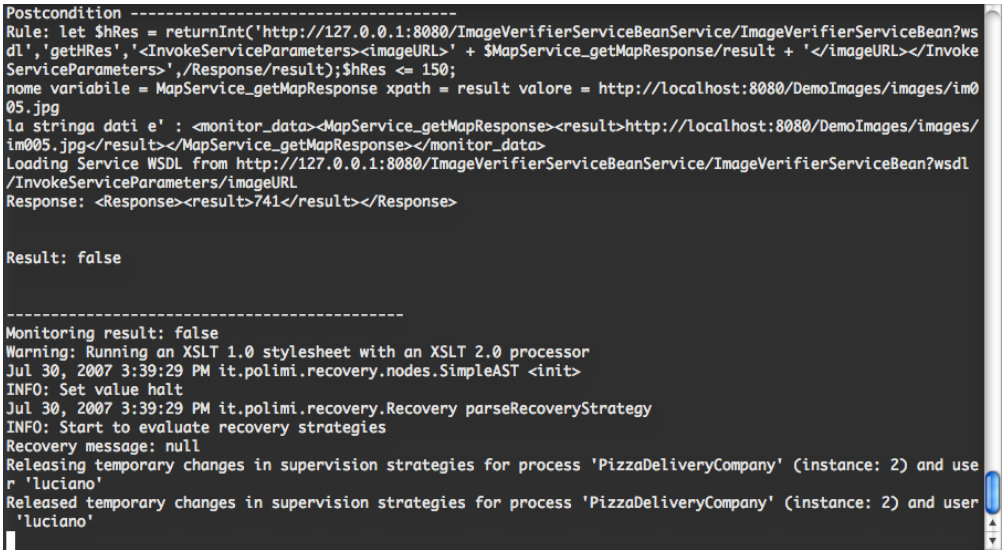

Figure 3: Output console of the monitored process

5 Appendix

5.1 WS-CoL grammar

⟨analyzer⟩ ::⇒ ⟨rules⟩ | ⟨recovery⟩
⟨recovery⟩ ::⇒ ⟨complete-strategy⟩ | strategy
⟨complete-strategy⟩ ::⇒ ⟨ifStrategy⟩ ⟨elseIfStrategy⟩* ⟨elseStrategy⟩$^{?}$
⟨ifStrategy⟩ ::⇒ **if** ⟨condition⟩ ⟨strategy⟩
⟨elseIfStrategy⟩ ::⇒ **elseif** ⟨condition⟩ ⟨strategy⟩
⟨elseStrategy⟩ ::⇒ **else** ⟨strategy⟩
⟨condition⟩ ::⇒ **(** ⟨rules⟩ **)**
⟨strategy⟩ ::⇒ { ⟨steps⟩ }
⟨steps⟩ ::⇒ ⟨rec-step⟩ (**or** ⟨rec-step⟩)*
⟨rec-step⟩ ::⇒ ⟨actions⟩
⟨actions⟩ ::⇒ action (**and** ⟨action⟩)*
⟨action⟩ ::⇒ ⟨identifier⟩ **(** ⟨list⟩$^{?}$ **)**
⟨rules⟩ ::⇒ ⟨sub-rule⟩ ((**==>** | <**==** | <**==>**) ⟨sub-rule⟩)* **;**
⟨sub-rule⟩ ::⇒ ⟨and-expression⟩ (|| ⟨and-expression⟩)*

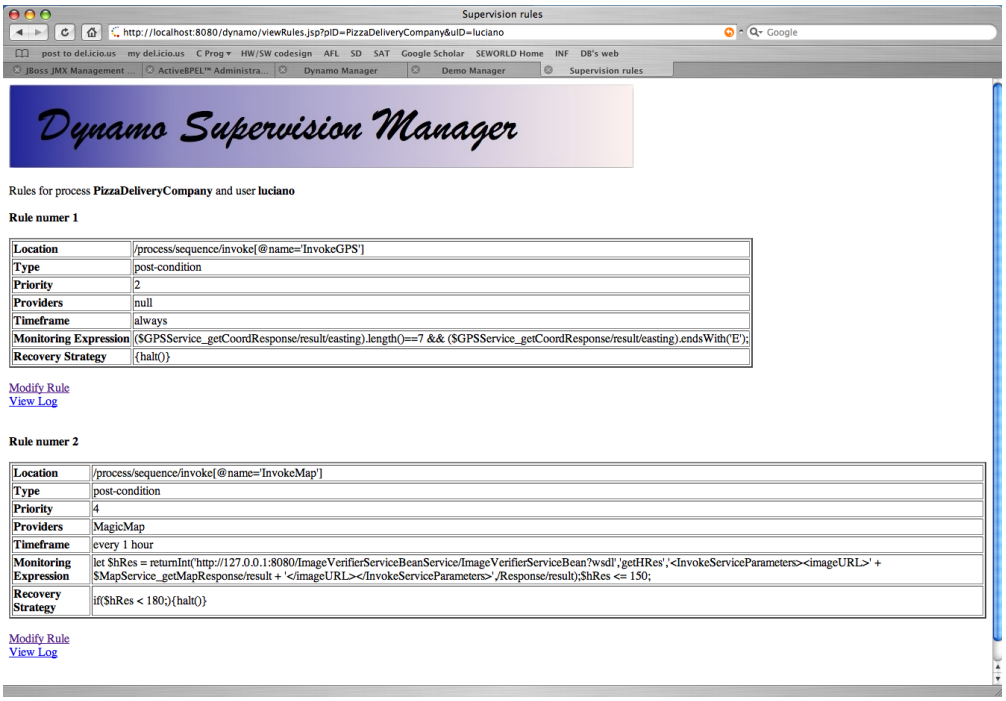

Figure 4: Monitoring rules modified with the "Dynamo Supervision Manager"

⟨and-expression⟩ ::⇒ ⟨equals-expression⟩ (**&&** ⟨equals-expression⟩)*
⟨equals-expression⟩ ::⇒ ⟨relational-expression⟩ ((**==** | **!=**) ⟨relational-expression⟩)?
⟨relational-expression⟩ ::⇒ ⟨operator-expression⟩ ((> | >**=** | < | <**=**) ⟨operator-expression⟩)?
⟨operator-expression⟩ ::⇒ ⟨basic-expression⟩ ((**+** | **-** | ***** | / | **%**) ⟨basic-expression⟩)*
⟨basic-expression⟩ ::⇒ ⟨dot-expression⟩ | ⟨variable⟩ | ⟨exists⟩ | ⟨forall⟩ | ⟨let⟩ | ⟨store⟩ | ⟨avg⟩ | ⟨min⟩ | ⟨max⟩ | ⟨sum⟩ | ⟨product⟩ | **true** | **false** | ⟨NUMBER⟩ | ⟨string-value⟩
⟨forall⟩ ::⇒ **(forall $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨exists⟩ ::⇒ **(exists $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨dot-expression⟩ ::⇒ ⟨variable⟩ **.** ⟨identifier⟩ **(** ⟨list⟩? **)**
⟨let⟩ ::⇒ **let $** ⟨identifier⟩ **=** ⟨sub-rule⟩
⟨store⟩ ::⇒ **store $** ⟨identifier⟩ **=** ⟨sub-rule⟩
⟨avg⟩ ::⇒ **(avg $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨sum⟩ ::⇒ **(sum $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨min⟩ ::⇒ **(min $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨max⟩ ::⇒ **(max $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨product⟩ ::⇒ **(product $** ⟨identifier⟩ **in** ⟨variable⟩ **;** ⟨sub-rule⟩ **)**
⟨variable⟩ ::⇒ **(** (⟨ivar⟩ | ⟨evar⟩ | ⟨hvar⟩) **)** | (⟨ivar⟩ | ⟨evar⟩ | ⟨hvar⟩)
⟨ivar⟩ ::⇒ **$** ⟨identifier⟩ ⟨xpath-expression⟩?
⟨evar⟩ ::⇒ ⟨returnType⟩ **(** ⟨string-value⟩ **,** ⟨string-value⟩ **,** ⟨string-value⟩ **,** ⟨xpath-expression⟩ **)**
⟨returnType⟩ ::⇒ **returnInt** | **returnBool** | **returnString**
⟨hvar⟩ ::⇒ **retrieve (** ⟨string-value⟩ (**,** ⟨string-value⟩)? (**,** ⟨NUMBER⟩) **,** ⟨xpath-expression⟩ **,** ⟨NUMBER⟩ **, $** ⟨identifier⟩ (**,** ⟨NUMBER⟩) **)**
⟨alias⟩ ::⇒ **$** ⟨identifier⟩
⟨list⟩ ::⇒ ⟨sub-rule⟩ (**,** ⟨sub-rule⟩)*
⟨string-value⟩ ::⇒ ⟨sub-string-value⟩ (**+** ⟨sub-string-value⟩)*
⟨sub-string-value⟩ ::⇒ ⟨identifier⟩ | ⟨literal⟩ | ⟨variable⟩
⟨xpath-expression⟩ ::⇒ ⟨union-expression⟩
⟨location-path⟩ ::⇒ ⟨absolute-location-path⟩ | ⟨relative-location-path⟩
⟨absolute-location-path⟩ ::⇒ (/ | //) (⟨i-relative-location-path⟩ | ε)
⟨relative-location-path⟩ ::⇒ ⟨i-relative-location-path⟩
⟨i-relative-location-path⟩ ::⇒ ⟨step⟩ ((/ | //) ⟨step⟩)*
⟨step⟩ ::⇒ ((⟨axis⟩ | ε) ((((⟨identifier⟩ **:**)? (⟨identifier⟩ | *****))) | ⟨special-step⟩) ⟨predicates⟩*) | ⟨abbr-step⟩
⟨predicates⟩*

⟨special-step⟩ ::⇒ **processing-instruction (** ⟨identifier⟩$^{?}$ **)** | (**comment** | **text** | **node**) **( )**
⟨axis⟩ ::⇒ ⟨identifier⟩ **::** | **@**
⟨predicate⟩ ::⇒ { ⟨predicate-expr⟩ }
⟨predicate-expr⟩ ::⇒ ⟨expr⟩
⟨expr⟩ ::⇒ ⟨or-expr⟩
⟨primary-expr⟩ ::⇒ ⟨variable-reference⟩ | **(** ⟨expr⟩ **)** | ⟨literal⟩ | ⟨number⟩ | ⟨function-call⟩
⟨literal⟩ ::⇒ ⟨LITERAL⟩
⟨number⟩ ::⇒ ⟨NUMBER⟩
⟨variable-reference⟩ ::⇒ **$**⟨identifier⟩
⟨function-call⟩ ::⇒ ⟨identifier⟩ **(** ⟨arg-list⟩$^{?}$ **)**
⟨arg-list⟩ ::⇒ ⟨argument⟩ (**,** ⟨argument⟩)*
⟨argument⟩ ::⇒ ⟨expr⟩
⟨union-expr⟩ ::⇒ ⟨path-expr⟩ (| ⟨path-expr⟩)*
⟨path-expression⟩ ::⇒ ⟨location-path⟩ | ⟨filter-expr⟩ ⟨absolute-location-path⟩$^{?}$
⟨filter-expr⟩ ::⇒ ⟨primary-expr⟩ ⟨predicate⟩$^{?}$
⟨or-expr⟩ ::⇒ ⟨and-expr⟩ (**or** ⟨and-expr⟩)*
⟨and-expr⟩ ::⇒ ⟨equality-expr⟩ (**and** ⟨equality-expr⟩)*
⟨equality-expr⟩ ::⇒ ⟨relational-expr⟩ ((**=** | **!=**) ⟨relational-expr⟩)$^{?}$
⟨relational-expr⟩ ::⇒ ⟨additive-expr⟩ ((< | <**=** | > | >**=**) ⟨additive-expr⟩)$^{?}$
⟨additive-expr⟩ ::⇒ ⟨mult-expr⟩ ((**+** | **-**) ⟨mult-expr⟩)$^{?}$
⟨mult-expr⟩ ::⇒ ⟨unary-expr⟩ ((***** | /) ⟨unary-expr⟩)$^{?}$
⟨unary-expr⟩ ::⇒ ⟨union-expr⟩ | **-** ⟨unary-expr⟩

5.2 Architecture

Figure 5 depicts the Dynamo-AOP monitoring framework, by illustrating the dependencies existing between the various components and the technologies used in the implementation. The Configuration Manager is a

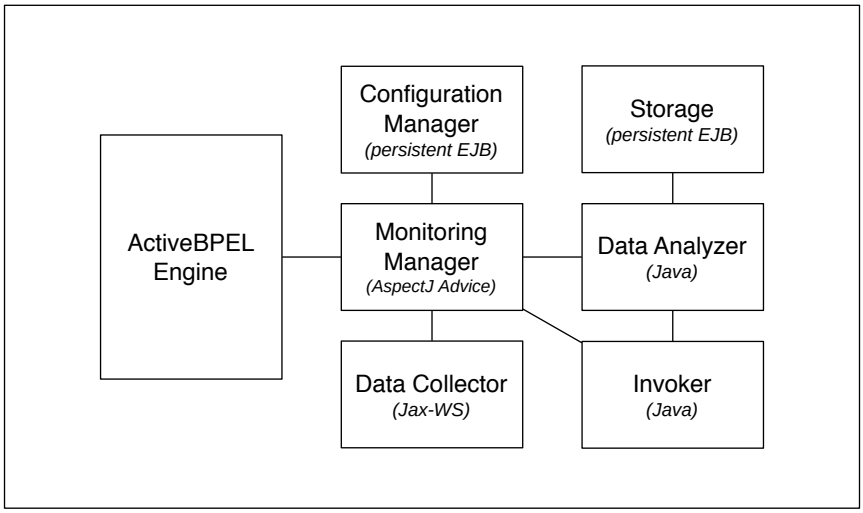

Figure 5: Dynamo-AOP components architecture

storage component for all the properties that have to be monitored. The ActiveBPEL engine is a modified version of ActiveBPEL [1] in which we embed monitoring. This is achieved by following an aspect oriented programming approach [6]. The engine is a Java program in which we weave the cross-cutting monitoring features via AspectJ [5]. ActiveBPEL works by creating an internal tree representation of the process being executed. In this tree, each node represents a single BPEL activity in the process definition, and is an appropriate extension of the `AEActivityDefinition` class. Each node contains the information necessary to perform the particular activity it is associated with. At run time, the tree is visited and the definition classes are used by the engine to instantiate appropriate `AEActivityImpl` classes, all of which implement a common interface. Amongst other things, this interface provides an `execute` method where the activity's primary action is performed. For example, a *scope* activity will set up its internal variables,

while an *invoke* activity will perform the appropriate external invocation. To perform monitoring, we intercept the process after the `execute` method is called for the various BPEL activities. These are the points where the Monitoring Manager (implemented as an AspectJ advice) is activated. Its main responsibility is data collection, both from within the process and from the outside world, through the Data Collector. The collected data, together with the monitored property, are sent to the Data Analyzer, which first retrieves external data and/or historical variables by calling the Invoker and the Storage, respectively, and then analyzes the property, passing the result of the evaluation back to the Monitoring Manager. At this point, before returning control to the process, the Monitoring Manager executes the recovery code if a monitoring rule has been violated.

References

[1] Active Endpoints. Activebpel engine architecture. http://www.activebpel.org/docs/architecture.html, 2007.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1, May 2003.

[3] L. Baresi and S. Guinea. Towards dynamic monitoring of WS-BPEL processes. In *ICSOC 2005: Proceedings of the 3rd International Conference on Service Oriented Computing*, volume 3826 of *Lecture Notes in Computer Science*, pages 269–282. Springer, 2005.

[4] S. Guinea. *Dynamo: a Framework for the Supervision of Web Service Compositions*. PhD thesis, Politecnico di Milano, 2007.

[5] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *ECOOP 2001 - Object-Oriented Programming, 15th European Conference, Proceedings*, volume 2072 of *Lecture Notes in Computer Science*, pages 327–353. Springer, 2001.

[6] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *ECOOP'97 - Object-Oriented Programming, 11th European Conference, Proceedings*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242. Springer, 1997.

[7] G. T. Leavens, A. L. Baker, and C. Ruby. JML: A notation for detailed design. In H. Kilov, B. Rumpe, and I. Simmonds, editors, *Behavioral Specifications of Businesses and Systems*, pages 175–188. Kluwer Academic Publishers, Boston, 1999.

[8] W3C. XML path language (XPATH). on-line at: http://www.w3.org/TR/xpath, 1999.