WS-Policy4MASC – A WS-Policy Extension Used in the Manageable and Adaptable Service Compositions (MASC) Middleware

Vladimir Tosic [1,2], Abdelkarim Erradi [1], Piyush Maheshwari [1,3]
[1] School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
[2] Department of Computer Science, University of Western Ontario, London, Ontario, Canada
[3] IBM India Research Lab, New Delhi, India
vladat@computer.org, aerradi@cse.unsw.edu.au, pimahesh@in.ibm.com

UNSW-CSE-TR-0705
Technical Report, January 2007

THE UNIVERSITY OF
NEW SOUTH WALES

School of Computer Science and Engineering
The University of New South Wales
NSW 2052, Australia

Abstract

WS-Policy4MASC is a new XML language that we have developed for policy specification in the Manageable and Adaptable Service Compositions (MASC) middleware. It can be also used for other Web service middleware. It extends the Web Services Policy Framework (WS-Policy) by defining new types of policy assertions. Goal policy assertions specify requirements and guarantees (e.g., maximal response time) to be met in desired normal operation. They guide monitoring activities in MASC. Action policy assertions specify actions to be taken if certain conditions are met or not met (e.g., some guarantees were not satisfied). They guide adaptation and other control actions. Utility policy assertions specify monetary values assigned to particular situations (e.g., execution of some action). They can be used by MASC for billing and for selection between alternative action policy assertions. Meta-policy assertions can be used to specify which action policy assertions are alternative and which conflict resolution strategy (e.g., profit maximization) should be used. In addition to these 4 new types of policy assertions, WS-Policy4MASC enables specification of additional information that is necessary for run-time policy-driven management. This includes information about conditions when policy assertions are evaluated/ executed, parties performing this evaluation /execution, a party responsible for meeting a goal policy assertion, ontological meaning, monitored data items, states, state transitions, schedules, events, and various expressions. We have evaluated feasibility of the WS-Policy4MASC solutions by implementing a policy repository and other modules in MASC. Further, we have examined their usefulness on a set of realistic stock trading scenarios.

1. Introduction and Motivation

In the area of management (monitoring and control) of networks and distributed systems, policy-driven management [1] has caught considerable attention during the last decade. A policy can be defined as a collection of high-level, implementation-independent, operation and management goals and/or rules expressed in a human-readable form. Policies can be viewed as decision-making guidelines for operation and management of a system. A policy-driven management system refines these high-level goals and rules into many low-level, implementation-specific, actions controlling operation and management of particular system elements. For example, a policy could be used to: ensure compliance, configure behavior, or achieve adaptability. Several classifications of policies exist. We find the classification from [2] particularly useful. It differentiates action policies (describing actions to be taken in a particular state), goal policies (describing desired states of the system), and utility function policies (defining value of each possible state).

During the last several years, there has also been significant progress on developing technologies for Extensible Markup Language (XML) Web services. These technologies are based on widely used industrial standards SOAP, the Web Services Description Language (WSDL), and the Web Services Business Process Execution Language (WSBPEL). In spite of achieved results, there are still many open issues related to practical uses of Web services. Many of these issues are associated with management, particularly control of the execution, of Web services and Web service compositions. Management is needed to achieve correct operation, recover from faults, optimize performance, increase security, perform accounting, and achieve maximal benefits from the managed systems. One of the prerequisites for performing management of Web services and Web service compositions is existence of a machine processeable and precise format for description of monitored requirements, guarantees, capabilities, and control actions. A number of languages have recently appeared to address some aspects of this need, usually specification of quality of service (QoS) requirements and guarantees. Some of these languages are accompanied by corresponding management middleware. For example, this is the case with the Web Service Level Agreement (WSLA) [3], the Web Services Management Language (WSML) [4], the Web Service Offerings Language (WSOL) [5], the Web Services Agreement Specification (WS-Agreement) [6], the Web Services Policy Framework (WS-Policy) [7, 8], and the Web Service Constraint Language (WS-CoL) [9]. However, Web service management is a complex area and the past results have predominantly addressed (simpler) monitoring or QoS-based selection of Web services and less (more challenging) control (e.g., adaptation). Further, they have mainly concentrated on management of individual Web services, while challenges of management of Web service compositions are relatively under-explored. In addition, the current solutions are almost exclusively focused on optimization of technical QoS metrics (e.g., response time) and provide only a very simple treatment of tangible business metrics (e.g., profit) without examining non-tangible business metrics (e.g., customer retention).

Our work aims to go beyond the past approaches to control of Web services and Web service compositions and provide additional agility and self-adaptation capabilities driven by maximizing business value. To achieve this, we have decided to leverage achievements of policy-driven management. We have examined a number of requirements for policy-driven management of Web services and Web service compositions and concluded that the none of the past languages and middleware tools fully addresses them. (A detailed list of language-related requirements can be found in [10], while a brief summary of some of them in [11].) To address these requirements, we have been developing our policy-driven Web service management middleware Manageable and Adaptive Service Compositions (MASC) [12, 13, 14, 10]. It provides monitoring of Web services and Web service compositions, a wide range of dynamic (run-time) adaptation mechanisms (to handle business exceptions, versioning, faults, performance problems), coordination of adaptation actions between the SOAP messaging layer and the process orchestration layer, and the capability to select between alternative control actions in a way that will maximize tangible and intangible business value in various ways. The latter three features are distinctive characteristic of MASC compared to the related work. Further, MASC builds on the established policy-driven management principles [1]. In particular, description of monitoring and control aspects (e.g., what are possible faults and how to handle them) is externalized from business process (e.g., Web service composition) descriptions into separate policies. This externalization yields higher degree of flexibility, promotes reusability and contributes to keeping the specification of the base process simpler and easier to maintain. Another distinctive characteristic of MASC is that it leverages and extends the power and flexibility of the brand new Microsoft .NET 3.0 platform [15], particularly its components the Windows Workflow Foundation (WF) and the Windows Communication Foundation (WCF). Since WF uses the Extensible Application Markup

Language (XAML) for description of Web service compositions (processes, workflows) instead of WSBPEL, MASC also uses XAML. This is in contrast to the other Web service management middleware tools, which are based on Java and WSBPEL. This difference in underlying technology caused some architectural uniqueness of our solutions.

WS-Policy4MASC is our novel language for description of policies used in the MASC middleware and, thus, for its automatic configuration. As we will explain in this paper, it is our domain-independent extension of WS-Policy [7, 8], which is an industrial specification standardized by the World Wide Web Consortium (W3C). While we have examined extending other Web service management languages (particularly, WS-Agreement) and other policy-driven management languages, we have found that WS-Policy is most suitable for the MASC middleware and is widely used in practice. WS-Policy4MASC is, in principle, a powerful general language for specification of various Web service management policies. However, during our development of the language details we put an emphasis on supporting aspects that differentiate MASC from other Web service middleware. Particularly, our language enables detailed specification of information for various types of adaptation of Web service compositions and for selection between alternative control actions based on various strategies for maximization of monetary and intangible business values. Consequently, WS-Policy4MASC provides a number of solutions that are not present in past related works.

This paper presents the WS-Policy4MASC language. In this section, we have briefly introduced the area of policy-based Web service management and motivated our development of the language. The following section summarizes the main related work, putting particular emphasis on WS-Policy. An overview of WS-Policy4MASC is given in Section 3. This section has 4 subsections. Subsection 3.1 introduces the main constructs, such as goal/ action/utility/meta- policy assertions. Subsection 3.2 provides examples of these constructs. Subsection 3.3 lists different dynamic adaptation actions that can be expressed in the WS-Policy4MASC language and executed by the MASC middleware. The last subsection (3.4) explains how WS-Policy4MASC utility policy assertions and meta- policy assertions are used for selection between alternative control actions based on various strategies for maximization of monetary and intangible business values. Then, we present an overview of the MASC middleware architecture and its .NET 3.0 implementation in Section 4. In the last Section 5, we summarize conclusions and outline our ongoing and future work.

2. Related Work

The main related work to our project is the Web Services Policy Framework (WS-Policy) [7, 8], an industrial specification standardized by the World Wide Web Consortium (W3C). It defines an extensible container to hold domain-specific policy assertions. It also provides a general framework for attaching attributes/metadata to services and for placing range of interaction constraints with respect to various aspects, such as security (e.g., encryption type, authentication mode) or reliable messaging. It is intended as a complement to WSDL and WSBPEL.

In the WS-Policy model, a policy is defined as a collection of policy alternatives, each of which is a collection of policy assertions. WS-PolicyAttachment defines a generic mechanism that associates a policy with subjects to which the policy applies, such as WSDL elements or Web service registry information. Various policy subjects are possible, such as service, endpoint, operation, message, or message part. A policy scope is a set of policy subjects to which a policy may apply.

WS-Policy has a number of good features. For example, it is flexible and extensible – policies can be specified both inside and outside WSDL files. Further, it has some reusability mechanisms, such as inclusion and grouping of policies. Nevertheless, it must be noted that WS-Policy is only a general framework, while the details of the specification of particular categories of policies will be defined in specialized languages – domain-dependent extensions of WS-Policy. Currently, only extensions for security, reliable messaging, and a few other management areas that were not the focus of our project had been published. WS-PolicyAssertions can be used for the formal specification of functional constraints, but the contained expressions can be specified in any language. It is not clear whether and when some specialized languages for the specification of quality of service (QoS) policies, prices/penalties, and other management information will be developed. Some unification and standardization of common elements, such as expressions, of various WS-Policy languages would reduce the overhead of supporting this framework. Further, WS-Policy does not detail where, when, and how are policies monitored and evaluated. Since many policies have to be monitored and controlled during run-time, WS-Policy needs better support for management applications, including explicit specification of such management information. Consequently, we had to develop a new domain-independent WS-Policy extension, which we named WS-Policy4MASC.

The WS-Policy specification is currently evolving within the W3C standardization process [7]. We have decided to work with the past stable version described in [8], but plan to align our WS-Policy4MASC with new WS-Policy versions once they are stable.

Another important related work is Web Service Constraint Language (WS-CoL) [9], a domain-independent WS-Policy extension for specifying client-side monitoring policies, particularly those related to security. At deployment time, WS-CoL constraints attached to a process are translated into WSBPEL invoke activities that call the Monitoring Manager to evaluate the monitoring policies and detect anomalous conditions. This approach is similar to ours in that monitoring policies are specified externally rather than being embedded into the process specification. It achieves the desired reusability and separation of concerns. However, it only provides support for monitoring and focuses mainly on security. It does not provide full support for adaptation and business-driven management that are distinctive characteristics of our research.

The Web Services Policy Language (WSPL) [16], developed at Sun Microsystems, is a WS-Policy competitor. It is suitable for specifying a wide range of policies, e.g., acceptable and supported encryption algorithms or privacy guarantees. While there are some conceptual similarities with WS-Policy, the syntax of WSPL is a subset of the Extensible Access Control Markup Language (XACML). WSPL's key strength is its ability to support negotiation of mutually acceptable policies that represent intersection of two source policies. However, WSPL is not as popular as WS-Policy and it did not become a standard.

Another recent research trend to address adaptation issues is augmenting Web services middleware with autonomous behavior capabilities such as self-healing and self-configuring [17, 18, 19]. For example, IBM's Policy Management for Autonomic Computing (PMAC) [19] is a policy driven framework intended to simplify creation, storage, distribution, and execution of policies. It uses the Autonomic Computing Policy Language (ACPL) as the underlying policy language. ACPL is a strongly-typed XML-based language for specifying policy rules and expressions. Its key concepts are: scope, condition, decision and business value. The scope specifies the policy subject. The condition expresses when a policy is to be applied. The decision describes observable behavior or desired outcome of a policy in the form of a management action or a configuration profile. The business value expresses utility functions to make economic trade offs between policy alternatives. We found ACPL tightly coupled to PMAC and hence more suitable for configuring resources and managing applications. It offers limited support for monitoring and adaptation of Web service compositions. Furthermore, the policy subject is specified within the policy definition and this hinders the policies reusability and maintainability. On the contrary, WS-Policy offers more flexibility and has better acceptance by industry. Our work belongs to this emerging autonomic Web services research area, but has unique characteristics, outlined in the previous section.

Over the last decade, multiple approaches have been suggested for specifying policies for different application domains. Most of these proposals target network management, security, privacy, and trust. For example KAoS [20] enables specification and enforcement of authorization and obligation policies for Grid Computing and Semantic Web services. Our approach targets monitoring and adaptation of Web services and Web service compositions. Consequently, it is closest to a few works that have appeared in this area. For example, the proposed policy-driven Web service transactional frameworks (e.g., [21]) only address coordination of activity termination and possibly compensation to ensure that participating Web services reach consistent states after a failure. Our work complements them with suitable repair policies. However, we have only studied policy-driven local repair and adaptation strategies.

There is a body of work on policy refinement and policy conflict detection and resolution. Our MASC middleware supports basic policy refinement via mapping WS-Policy4MASC assertions into calls to MASC middleware management interfaces. In MASC, policy conflict detection and resolution uses WS-Policy4MASC meta-policies that describe how to maximize business value. However, we left design-time analysis of policies to detect and resolve policy conflicts for future work. We plan to reuse and/or adapt existing solutions in this area, e.g., [22].

3. Overview of WS-Policy4MASC

WS-Policy4MASC extends WS-Policy by defining XML schemas with new types of WS-Policy policy assertions. (These are not domain-specific policy assertions because WS-Policy4MASC constructs can be used for representing functional constraints, QoS, adaptation, security, prices, and other information.) There are no changes to WS-Policy constructs (e.g., <*wsp:Policy*>, <*wsp:All*>, and <*wsp:ExactlyOne*>), so these constructs can be used with WS-Policy4MASC in exactly the same way as for any other WS-Policy policy assertions. WS-Policy4MASC policy assertions can be attached to WSDL constructs (e.g., endpoint, operation, message) and WSBPEL or XAML constructs (e.g., process, sub-process, activity).

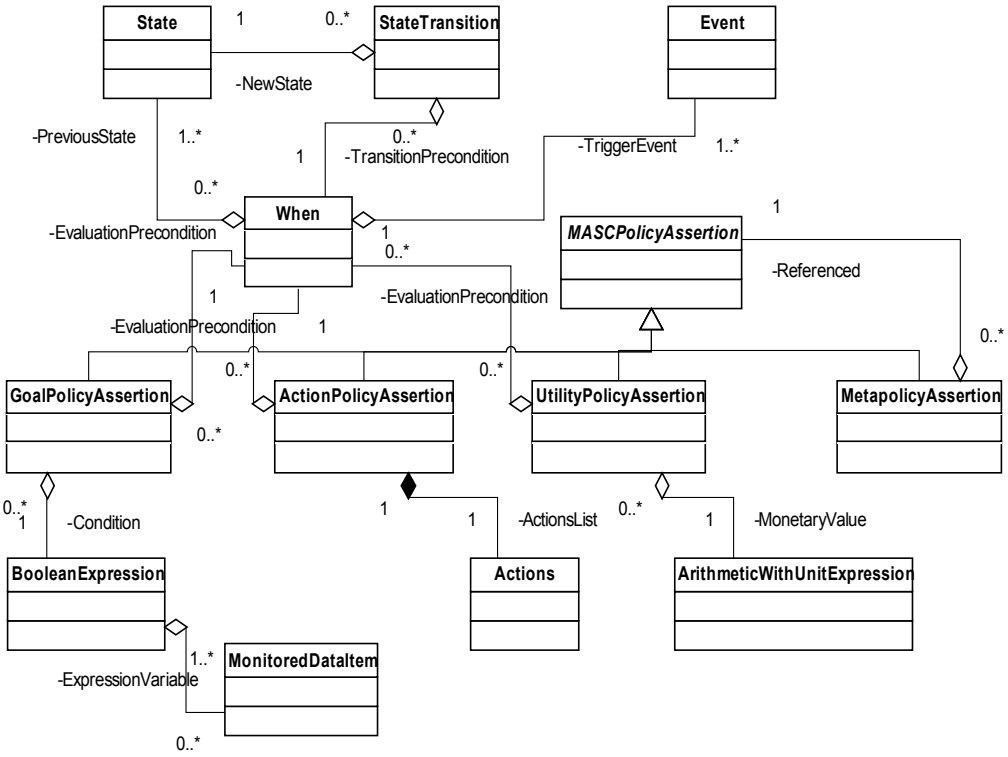

Figure 1. **Some relationships between the main WS-Policy4MASC constructs**

3.1. Main Constructs in WS-Policy4MASC

WS-Policy4MASC constructs can be classified into "real" policy assertions and other constructs. There are 4 types of "real" policy assertions:
(1) Goal policy assertions specify requirements and guarantees to be met in desired normal operation (e.g., response time of a particular activity has to be less than 1 second). In MASC, they guide monitoring activities.
(2) Action policy assertions specify actions to be taken if certain conditions are met (e.g., some guarantees were not satisfied). For example, these actions can be removal, addition, replacement, skipping, or retrying of a sub-process (or individual activity) or process termination. In MASC, they guide adaptation and other control actions (and a few aspects of monitoring, such as monitored data exchange).
(3) Utility policy assertions specify monetary values assigned to particular situations (e.g., execution of some action). They can be used by MASC for billing and for selection between alternative action policy assertions.
(4) Meta-policy assertions can be used to specify which action policy assertions are alternative and which conflict resolution strategy should be used.

It should be noted that the WS-Policy4MASC use of the terms “goal policy assertion” and “utility policy assertion” is somewhat different from the terms “goal policy” and “utility policy” used by [2]. A WS-Policy4MASC goal policy assertion specifies a condition (requirement or guarantee) to be achieved/met/satisfied, while a goal policy in [2] specifies a desired state (situation) of the system. While a complete desired state of the system is not denoted by an individual WS-Policy4MASC goal policy assertion, it is determined by a combination of all WS-Policy4MASC goal policy assertions valid at the same time. This difference could be viewed as consistent with the WS-Policy distinction between the terms “policy” and “policy assertion”. Analogously, an individual WS-Policy4MASC utility policy assertion does not denote a complete utility value associated to a particular system, but a combination of all WS-Policy4MASC utility policy assertions valid at the same time determines a complete utility value. A utility policy in [2] specifies utility value associated to a particular state (situation) of the system, while in WS-Policy4MASC a utility can be associated not only to staying in a state for some time, but also to state transitions, execution/non-execution of action policy assertions, satisfaction/non-satisfaction of goal policy assertions, and other events. However, the meaning of the WS-Policy4MASC term “action policy assertion” can be considered the same as the meaning of the [2] term “action policy”. They both represent a set of

actions to perform if particular conditions are met.

In addition to these 4 new types of "real" policy assertions, WS-Policy4MASC enables specification of additional information that is necessary for run-time policy-driven management. Some of this information (e.g., which party performs evaluation/execution of a policy assertion, which party is responsible for meeting a goal) is specified in attributes of the above-mentioned "real" policy assertions. Much more information is specified in additional WS-Policy4MASC constructs, specifying monitored data items, states, state transitions, schedules, events, scopes, and various expressions (Boolean, arithmetic, arithmetic with units, string, date/time/duration). These "other" constructs are also implemented as WS-Policy policy assertions from the syntax viewpoint (although they are not policy assertions from the semantic viewpoint), in order to support reusability and ease automatic code generation (this will be elaborated below). Probably the most important of them is the <*When*> construct that specifies when something (e.g., evaluation of a goal policy assertion or execution of actions in an action policy assertion) should take place. It contains information about one or more states in which this occurs, one or more events (e.g., Web service operation executed) that can (each individually) trigger this occurrence, and an optional additional Boolean condition to be satisfied (it can be used for filtering). The emphasis in this paper will be on explaining the 4 types of "real" policy assertions, while details of the other WS-Policy4MASC constructs can be found in the on-line language documentation [10].

Note that simplicity of automatic code generation from XML schemas into C# classes and reusability of specification elements had significant influences on the design of our language. For example, we avoided the <*choice*> element in XML schemas for WS-Policy4MASC because it resulted in C# classes that were difficult to understand and handle. A drawback of our approach is that many WS-Policy4MASC supporting constructs (e.g., event definitions) are specified as WS-Policy policy assertions, leading to specifications that are somewhat verbose. However, we hope that, in the future, WS-Policy4MASC files will be generated by graphical tools, so this verbosity of WS-Policy4MASC files will not be a strain for humans.

Some relationships between the main WS-Policy4MASC constructs are shown in the UML diagram in Figure 2. Due to the space constraints, the figure does not show all existing constructs and relationships. Goal policy assertions, action policy assertions, utility policy assertions and meta-policy assertions are subtypes of the abstract policy assertion construct *MASCPolicyAssertion*. It defines common attributes, such as policy assertion ID and party that performs execution/evaluation. The 4 policy assertion types have additional attributes and elements. The former 3 types of policy assertions reference a <*When*> element describing in which state(s) and on occurrence of which event(s) a policy assertion should be processed. A goal policy references a Boolean expression with the condition to be evaluated – only if the given expression evaluates to “true” the goal was achieved/met/satisfied. An action policy assertion references a group of actions that are to be executed. A utility policy assertion contains an arithmetic with (currency) unit expression that determines associated monetary value. A meta-policy assertion does not reference a <*When*> element, but a set of mutually conflicting policy assertions. It contains information about a conflict resolution strategy (this will be explained in Subsection 3.4).

3.2. WS-Policy4MASC Examples

Figure 2 illustrates some of the WS-Policy4MASC constructs on a simple example. A weather report Web service has one operation: *Integer weatherTemperature(String postalCode)*. Figure 2 shows how WS-Policy4MASC can be used to specify the post-condition that the result represents temperature in Celsius degrees and that it should be between -70C and 50C. WS-Policy4MASC policy assertions and other constructs are specified within a WS-Policy element (its attributes defining namespaces are omitted in Figure 2 for brevity). First, the <*MonitoredDataItem*>, <*MonitoredDataItemCollection*>, and <*ActionGroup*> constructs specify that the message part *"weatherTemperature"* is monitored and expressed in Celsius degrees. (For the above-mentioned reasons, this specification is verbose.) Definitions of states and event follows, but they are omitted from Figure 1. Then, a <*When*> construct referring to the state *"Executing"* and the event *"MessageToBeSent"* is defined. The subsequent action policy assertion specifies that when this event happens in this state, monitoring of the message part *"weatherTemperature"* is performed by the provider Web service. This action policy assertion is used to configure MASC monitoring modules. Definition of the Boolean expression *"LimitsOfValidWeatherTemperature"* is omitted from Figure 2 for brevity. This is a complex expression that specifies that values of the monitored data item (the message part *"weatherTemperature"*) must be between -70C and 50C. The above-mentioned <*When*> construct and Boolean expression are referenced in the definition of the subsequent goal policy assertion, which is also used to configure MASC monitoring modules. This goal policy assertion specifies that when the event

"MessageToBeSent" occurs in the state *"Executing"*, then the provider Web service should evaluate the mentioned Boolean expression. It also states that the provider is responsible for meeting this goal. In a separate file, a WS-Policy policy attachment element defines that this defined policy is applied to the reply message of the current weather report Web service. This information is also omitted from Figure 2 for brevity.

```
<!-- WS-Policy policy element (namespaces are omitted) -->
<wsp:Policy wsu:Id="CanadianWhetherReport-Output" ...>
<!-- Definition of monitored data items -->
 <masc-gp:MonitoredDataItem MASCID="Temperature-Weather-
InCelsius"  MessagePartName="weatherTemperature" ValueData
Type="xs:integer " Unit="ontology1:Celsius"/>
 <!-- Definition of monitoring data collection actions -->
 <masc-ap:MonitoredDataItemCollection MASCID="MonitoringOf
Temperature-Weather-InCelsius">
  <masc-gp:MonitoredDataItemRef To="tns:Temperature-Weather
-InCelsius"/>
 </masc-ap:MonitoredDataItemCollection>
 <!-- Definition of action groups -->
 <masc-ap:ActionGroup MASCID="Monitoring">
  <masc-ap:MonitoredDataCollectionRef To="tns:MonitoringOfTe
mperature-Weather-InCelsius"/>
 </masc-ap:ActionGroup>
 <!-- Definitions of states and events (omitted) -->
 ...
 <!-- Definition of When constructs -->
 <masc-se:When MASCID="Executing-MessageToBeSent">
  <masc-se:AllowedStates>
   <masc-se:StateRef To="tns:Executing"/>
  </masc-se:AllowedStates>
  <masc-se:PossibleTriggerEvents>
   <masc-se:EventRef To="tns:MessageToBeSent"/>
  </masc-se:PossibleTriggerEvents>
 </masc-se:When>
 <!--Definition of action policy assertions configuring monitoring-->
 <masc-ap:ActionPolicyAssertion MASCID="MonitorResultValue"
ManagementParty="MASC_WSPROVIDER">
  <masc-se:WhenRef To="tns:Executing-MessageToBeSent"/>
  <masc-ap:ActionGroupRef To="tns:Monitoring"/>
 </masc-ap:ActionPolicyAssertion>
 <!-- Definition of Boolean expressions (omitted) -->
 ...
 <!-- Definition of goal policy assertions -->
 <masc-gp:GoalPolicyAssertion MASCID="ValidWeatherTempera
ture" ResponsibleParty="MASC_WSPROVIDER" ManagementPa
rty="MASC_WSPROVIDER">
  <masc-se:WhenRef To="tns:Executing-MessageToBeSent"/>
  <masc-ex:BooleanExpressionRef To="tns:LimitsOfValidWeather
Temperature"/>
 </masc-gp:GoalPolicyAssertion>
</wsp:Policy>
```

Figure 2. **Examples of WS-Policy4MASC constructs**

3.3. Supported Adaptation Actions

Specification of Web service requirements/guarantees for monitoring activities was enabled by a number of past languages, such as WSLA, WSML, WSOL, and WS-CoL. In this area, WS-Policy4MASC offers only minor advantages. However, one of the distinctive characteristics of WS-Policy4MASC is built-in support for a diverse range of common Web service composition adaptation actions. We focused on actions that are both useful and practical in handling frequently occurring adaptation needs, such customization (including versioning) and corrective adaptation for fault management. Most of these adaptation actions are executed by the process orchestration layer of MASC, although some of them are executed by the SOAP messaging layer of MASC (without an intervention by the process orchestration engine), as discussed in [13].

Actions supported by the WS-Policy4MASC language and the MASC middleware can be grouped into: 1) monitored data collection (using logging, measurement, calculation); 2) monitored data transfer (using

special push or pull operations); 3) cancellation of previously scheduled actions or events; 4) middleware configuration adaptation (e.g., changing parameters of used WS-* protocols); 5) messaging adaptation (data flow adaptation at the SOAP messaging layer, e.g., message transformation); 6) process instance structure adaptation (e.g., replacing a sub-process); 7) process instance execution adaptation (e.g., process termination); 8) activity instance execution adaptation (e.g., skipping an activity); 9) policy assertion adaptation (e.g., deactivating a policy assertion).

We have also considered supporting some other actions, but left them for future work. Due to the space constraints, we will list only the actions for the last 4 groups. Details about how some of these actions are implemented in the MASC middleware are discussed in [14].

The support for process instance structural adaptation includes a number of actions. The first is removal of a specified block of activities (or one activity) from the base (adapted) process. The second is addition of an external known process at a specified point of the base process. The third is addition of a single call to a known external Web service at a specified point of the base process. The fourth is searching a specified Web service directory with specified parameters for an external process or Web service operation that, when found, is added at a specified point in the base process. The fifth is a replacement of a specified block of activities with an external known process. The sixth is a replacement of a specified block of activities with a single call to a known external Web service. The seventh is searching a specified Web service directory with specified parameters for an external process or Web service operation that, when found, replaces a block of activities in the base process. While replacement could be relatively easily modeled as a removal plus an addition, we decided to model it explicitly because it is a common case and because its meaning is hidden when it is modeled with two separate constructs.

The support for process instance execution adaptation includes actions for process termination, process suspension, and process resumption.

The support for activity instance execution adaptation also includes many actions. The first is cancellation of the currently executing activity. The second is skipping the next activity that was supposed to be executed. The third is skipping a block of activities from the next activity to a specified activity. The fourth is rescheduling of the next activity for some (specified) later point in the process execution. The fifth is compensation of the last executed activity (assuming that its compensation operation is known). The sixth is process-level retrial of the last activity (e.g., if it was not completed). The seventh is suspension of the currently executing activity (this suspends its thread, but not the other threads in the process). The eight is resumption of a specified previously suspended activity.

The support for policy assertion adaptation contains actions for deactivation and activation of individual policy assertions. We plan to add priorities to policy assertions, so we will also add an action to change these priorities during run-time.

3.4. WS-Policy4MASC Support for Business-Driven Web Service Management

A particular novelty of WS-Policy4MASC are utility policy assertions and meta-policy assertions, so we will describe them in more detail here. (Some aspects were also discussed in [11].) Utility policy assertions enable providing monetary amounts (specified as general expressions involving numbers with currency units) to <When> constructs that contain information about allowed states, trigger events, and optional filtering conditions. For example, it is possible to specify amounts paid when goal policy assertions are met (or not met) or when action policy assertions are executed. Since all real monetary transactions require two parties, two attributes enable specification of the beneficiary party and the paying party. A positive monetary amount means that the beneficiary party receives payment from the paying party, while a negative amount denotes that the beneficiary party has to pay the paying party. In the former case, the beneficiary party has a benefit, while in the latter it has a cost. This is inverse for the paying party. The sum of (positive) benefits and (negative) costs is a profit. While it is mandatory to specify a beneficiary party, specification of a paying party is optional. If it is missing, this means that the specified monetary amount is not a real scheduled payment, but an estimate of some possible future business value from one or several paying parties. An additional Boolean attribute specifies whether the given amount is tangible (i.e., a real monetary amount paid) or intangible (i.e., a monetary estimate of some other business value, such as customer satisfaction). In a future version of WS-Policy4MASC, it will be possible to also specify schedules of future payments (currently, only one payment is specified per utility policy assertion), probability that an estimated future business value will be realized, and confidence in the precision of monetary estimates of intangible business values. It is also possible that a separation between various intangible business values will also be supported later.

A meta-policy assertion lists several conflicting action policy assertions and a conflict resolution strategy when some of them are triggered simultaneously. At this time, we have focused on strategies that choose only the best (as defined under some criteria) among the listed alternatives. In the future, we will also research strategies that allow conditional execution of more than one alternative. The WS-Policy4MASC language is extensible, so one can add new strategies. We have defined several strategies using various maximizations of business values. Further, we have designed MASC middleware support (e.g., algorithms and data structures) for these strategies and started implementing them in our MASC prototype (the MASCPolicyDecisionMaker module). The strategies are classified along 3 mutually orthogonal dimensions:

1. 'Only immediate' vs. 'long-term': For all strategies, it is possible to specify an optional ending event until which monetary values are added. If it is not specified, only immediate monetary implications of actions specified in the listed action policy are added. However, if such an ending event is specified, then a discrete event simulation is started and monetary implications of actions that are triggered by executing previously simulated actions are also calculated and added, until the specified ending event (or some specified limit of the number of counted values) is reached. Note that in the current MASC prototype, such discrete event simulations are not yet implemented, but they are planned for near future work.

2. 'Both agreed payments and estimated future business values' vs. 'only agreed payments': One group of strategies adds all agreed payments and estimated future business value and compares the results between alternative action policy assertions to choose only the best one. Another group of strategies adds only agreed payments, but if difference between two (or more) such sums is less or equal some specified amount, then the strategy separately adds estimated future business values and uses this as a tiebreaker. In the future, we will also develop strategies that will take into consideration probability of estimated future business values.

3. 'Both tangible and intangible' vs. 'only tangible' vs. 'only intangible': One group of strategies adds all tangible and intangible business values and compares the results between alternative action policy assertions to choose only the best one. Another group of strategies adds only tangible business values, but if difference between two (or more) such sums is less or equal some specified amount, then the strategy separately adds intangible business values and uses this as a tiebreaker. Conversely, yet another group of strategies adds only intangible business values, but if difference between two or more such sums is less or equal some specified amount, then the strategy separately adds tangible business values and uses this as a tiebreaker. The latter group of strategies is used when market share, customer retention, customer satisfaction and/or other intangible business metrics are more important than immediate profit. In the future, we will also develop strategies that will take into consideration confidence in the precision of monetary estimates of intangible business values.

Since these 3 dimensions are mutually orthogonal, a strategy specifies behavior along each dimension, e.g., 'only immediate' (dimension 1), 'only agreed payments' (dimension 2), and 'only tangible' (dimension 3). This produces 2*2*3=12 combinations. For the 4 combinations with 'only agreed payments' along dimension 2 and either 'only tangible' or 'only intangible' along dimension 3, it is also necessary to specify which dimension is used as the first tiebreaker, which produces 2 variations per combination. This means that, currently, the total number of strategies that we have defined is 12+4=16.

We will also consider adding the fourth dimension: 'benefits and costs' vs. 'cost limit'. Here, benefits, costs, and profit are from the viewpoint of the beneficiary party. Currently, all our strategies are in the former group because we add all positive and negative monetary values and choose an alternative with the highest total profit. However, in some cases alternatives with too high costs are not acceptable (e.g., due to a lack of current funds) even if they bring higher long-term profit, so the 'cost limit' group of strategies seems useful.

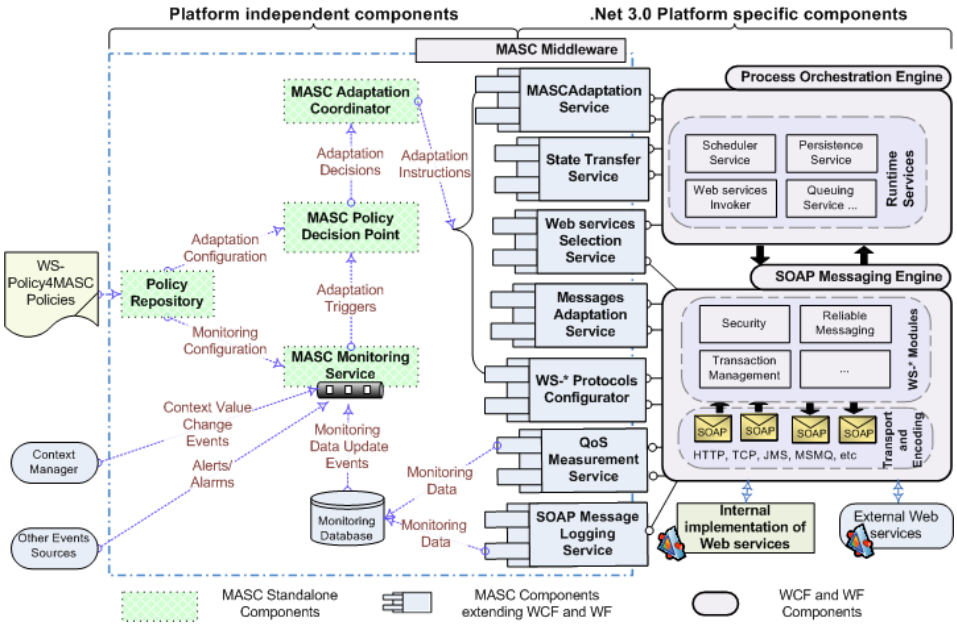

Figure 3. **Conceptual architecture of MASC**

4. Use of WS-Policy4MASC in the MASC Middleware

This section presents the architecture of the MASC middleware, with an emphasis on modules that facilitate policy-driven management of Web service compositions representing business processes (workflows). Additional details can be found in [12, 14]. The MASC architecture, shown in Figure 3 incorporates both platform-independent components and platform-specific ones. The former can be leveraged regardless of the SOAP messaging engine and the process orchestration engine used. The latter execute adaptation actions through transforming WS-Policy4MASC policy assertions into either .NET 3.0 platform-specific commands (i.e., API calls to the WF orchestration engine, the WCF SOAP messaging engine, or one of their custom extensions) or actions over messages exchanged between the composed Web services.

Within the MASC middleware, WS-Policy4MASC policy assertions are stored in an in-memory policy repository. It is a collection of instances of policy classes generated automatically from the WS-Policy4MASC schema, using an XML-schema-to-classes generator (in our .NET 3.0 and C#-based prototype of MASC, we used the XSD tool from .NET 3.0). When MASC starts, our *MASCPolicyParser* within it imports WS-Policy4MASC files, creates instances of corresponding policy classes, and stores these instances in the policy repository.

The *SOAPMessageLoggingService* and the *QoSMeasurementService* monitor the messages exchanges between the composed Web services and collect monitoring information, such as values of message parts and measured QoS metrics. This information is stored in the Monitoring Database. Using this information, the *MASCMonitoringService* evaluates goal monitoring policies to detect adaptation triggers and events of interest. When they happen, the *MASCMonitoringService* generates an event (with all necessary information, e.g., process instance ID and monitored data values) to the *MASCPolicyDecionPoint*. The *MASCPolicyDecionPoint* determines adaptation action policy assertions to be applied and submits them to the *MASCAdaptationCoordinator* for execution. If there are several alternatives, it chooses the one to execute, based on WS-Policy4MASC meta-policy and utility policy assertions and built-in strategies for maximization of business value. In a future MASC version, *MASCPolicyDecionPoint* will be extended with evaluation of pre-conditions and constraints associated with adaptation actions to ensure correctness of both the adaptation actions and the state of the adapted process.

The *MASCAdaptationCoordinator* coordinates the execution of adaptation actions between the SOAP messaging layer and the process orchestration layer. Management actions can adapt a single process instance, several instances of the same process schema, or instances of different process schemas. The adaptation at the process orchestration layer is managed by the *MASCAdaptationService*. It is implemented as a WF runtime service and exposes a set of management interfaces that abstract interactions between the MASC's decision making components and the WF runtime. The *MASCAdaptationService*'s management

interfaces correspond to the adaptation actions supported by WS-Policy4MASC. Their implementation maps these actions into WF extension commands and API calls to .NET 3.0 libraries.

We have evaluated feasibility of the WS-Policy4MASC solutions by implementing a policy repository and other modules in MASC. Further, we have examined their expressiveness, effectiveness, and usefulness on a set of realistic stock trading scenarios, described in more detail in [12, 14]. We have written WS-Policy4MASC files for some of the scenarios in order to check whether and how various adaptation needs can be expressed in our language. In addition, we have implemented and studied some of the scenarios with our prototype of the MASC middleware. For example, in some experiments MASC dynamically adapted a base business process for national stock trading with support for international stock trading. In other experiments, we periodically injected random exception events at various stages of the stock trading process to study behavior of the system (with and without MASC) in response to faults or QoS changes of constituent services. In some experiments, we measured the overhead introduced by MASC on overall response time. The conducted experiments were completed successfully and demonstrated feasibility and usefulness of the MASC approach in adding dynamic adaptation capabilities to existing Web services compositions, guided by declarative policies specified in WS-Policy4MASC. MASC has provided a solution for policy-driven static and dynamic adaptation without any changes to the base process definition, implementations of the used Web services, or the implementation of .NET 3.0 technologies. All that is needed for adaptation is a WS-Policy4MASC document describing policy assertions to be enforced. When a WS-Policy4MASC document changes, these changes are automatically enforced the next time adaptation is needed, with no need to restart any software component.

5. Conclusions and Future Work

We argue that an extended WS-Policy can play a key unifying role in annotating WSDL and WSBPEL Web service descriptions with various rules and support for Web service management (monitoring and control), such as service customization/versioning, fault management, and QoS management. Our WS-Policy extensions for the specification of goal policy assertions, action policy assertions, and utility policy assertions address the same needs as WSLA, WSOL, WS-Agreement, and WS-CoL. Thus, our work enables that an XML Web service composition can be comprehensively described using only WSDL, WSBPEL, and our WS-Policy4MASC. However, WS-Policy4MASC provides a number of solutions that are not present in related works, such as specification of information for various types of adaptation of Web service compositions and for selection between alternative control actions based on various strategies for maximization of monetary and intangible business values. We have defined and started implementing 16 such strategies.

We have completed definition of the main XML schemas for WS-Policy4MASC. Our initial focus was on supporting monitoring and dynamic adaptation through WS-Policy4MASC goal policy assertions, action policy assertions, and related constructs (e.g., describing when something occurs). Once this was completed, our focus shifted to supporting business-driven management of Web services through using WS-Policy4MASC utility policy assertions and meta-policy assertions to select between alternative action policy assertions. The main item for our ongoing work is further development of the proof-of-concept prototype implementation of the MASC middleware that uses WS-Policy4MASC policy assertions. While we already have a working prototype [12, 14], we use an iterative development process to add new features into it (and, sometimes, the MASC architecture) and evaluate them on case studies. In some cases, changes to the MASC architecture require changes to the WS-Policy4MASC schemas (language grammar), so our language will continue to evolve.

Acknowledgments. This work is a part of the research project “Building Policy-Driven Middleware for QoS-Aware and Adaptive Web Services Composition” sponsored by the Australian Research Council (ARC) and Microsoft Australia through the ARC Linkage Project LP0453880. We also thank A/Prof. Boualem Benattallah for insightful discussions and his comments.

References

[1] Sloman, M. 1994, 'Policy driven management for distributed systems', J. of Network and Systems Management, Plenum, Vol. 2, No. 4, pp. 333-360.

[2] Kephart, J.O. and Walsh, W.E. 2004, 'An Artificial Intelligence Perspective on Autonomic Computing Policies', in Policy 2004, IEEE, pp. 3-12.

[3] A. Keller, and H. Ludwig 2003, 'The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services', J. of Network and Systems Management, Plenum, Vol. 11, No 1, pp. 57-81.

[4] Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., and Casati, F. 'Automated SLA Monitoring for Web Services', in DSOM 2002, LNCS, Springer, No. 2506, pp. 28-41

[5] Tosic, V., Pagurek, B., Patel, K., Esfandiari, B., and Ma, W. 2005, Management Applications of the Web Service Offerings Language (WSOL), Information Systems, Elsevier, Vol. 30, No. 7, pp. 564-586.

[6] Ludwig, H., Dan, A. and Kearney, R. 2004, 'Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements', in ICSOC'04, ACM, pp. 65-74.

[7] W3C Web Services Policy Working Group 2006, Web Services Policy (WS-Policy) 1.5, Nov. 2006, www.w3.org/TR/ws-policy/

[8] Schlimmer, J. (ed.) 2004, 'Web Services Policy Framework (WS-Policy)', version: Sept. 2004, www6.software.ibm.com/software/developer/library/ws-policy.pdf

[9] Baresi, L., Guinea, S. and Plebani, P. 2005, 'WS-Policy for Service Monitoring', in TES 2005, Norway, LNCS, Springer, Vol. 3811, pp. 72-83.

[10] A. Erradi (ed.) 2006, 'Manageable and Adaptable Service Compositions (MASC)', masc.web.cse.unsw.edu.au/

[11] Tosic, V., Erradi, A., Maheshwari, P. 2007, ' On Extending WS-Policy with Specification of XML Web Service Semantics, Monitoring, and Control Driven by Business Value', submitted for publication.

[12] Erradi, A., Maheshwari, P. and Tosic, V. 2006, 'Policy-Driven Middleware for Self-Adaptive Web Services Composition', in Middleware 2006, LNCS, Springer, Vol. 4290, pp. 62-80.

[13] Erradi, A., Maheshwari, P. and Tosic, V. 2006, 'Recovery Policies for Enhancing Web Services Reliability', in ICWS'06, IEEE.

[14] Erradi, A., Tosic, V. and Maheshwari, P. 2007, 'MASC – Middleware for Adaptive .NET 3.0 Composite Web Services', submitted for peer-review.

[15] Microsoft 2006, Microsoft .NET Framework 3.0 Community (NetFx3), www.netfx3.com

[16] Anderson, A.H. 2004, 'An Introduction to the Web Services Policy Language(WSPL)', in Policy 2004, IEEE, pp. 189-192.

[17] Pautasso, C., Heinis, T. and Alonso, G. 2005, 'Autonomic Execution of Service Compositions', in ICWS'05, IEEE.

[18] Verma, K., Doshi, P., Gomadam, K., Miller, J. A. and Sheth, A. P. 2006, 'Optimal Adaptation in Web Processes with Coordination Constraints', in ICWS 2006, IEEE.

[19] IBM 2006, 'Policy Management for Autonomic Computing', www.alphaworks.ibm.com/tech/pmac

[20] Uszok, A., Bradshaw, J. M., Johnson, M., Jeffers, R., Tate, A., Dalton, J. and Aitken, S. 2004, 'KAoS policy management for semantic Web services', IEEE Intelligent Systems and Their Applications, IEEE, Vol. 19, No. 4, pp. 32-41.

[21] Tai, S., Mikalsen, T., Wohlstadter, E., Desai, N. and Rouvellou, I. 2004, 'Transaction policies for service-oriented computing', Data & Knowledge Engineering, Vol. 51, No. 1, pp. 59-79.

[22] Agrawal, D., Giles, J., Lee, K.-W. and Lobo, J. 2005, 'Policy Ratification', in POLICY 2005, pp. 223-232.