Policy-Driven Middleware for Manageable and Adaptive Web Services Compositions

Abdelkarim Erradi[1], Piyush Maheshwari[2], Vladimir Tosic[3]

[1] School of Computer Science and Eng., University of New South Wales, Sydney, Australia
[2] IBM India Research Lab, New Delhi, India
[3*] National ICT of Australia (NICTA), Sydney, Australia
aerradi@cse.unsw.edu.au, pimahesh@in.ibm.com, vladat@computer.org

Abstract. MASC (Manageable and Adaptive Service Compositions)[1*] is a policy-based middleware for monitoring of Web service compositions and their dynamic adaptation to various runtime changes. The monitorable requirements and the adaptation actions are specified in the WS-Policy4MASC language which extends WS-Policy by defining new types of policy assertions. In this paper, we present an overview of MASC architecture and implementation. Compared with recent related works, MASC has several distinctive characteristics: (1) support for synchronous and asynchronous monitoring and coordination of adaptation both at the SOAP messaging layer and the process orchestration layer, (2) greater diversity of monitoring and control constructs, for example, a sub-process (or an activity) can be added, removed, replaced, skipped, or retried, (3) the externalization of monitoring and adaptation actions from definitions of business processes, (4) use of both technical and business metrics for adaptation decisions, and (5) extend the power and flexibility of the new Microsoft .NET 3.0 platform. We have implemented a MASC proof-of-concept prototype and evaluated it on monitoring and adaptation scenarios from a stock trading case study.

1. Introduction and Motivation

Organizations are increasingly using composite Web services to automate business processes, via dynamically selecting and assembling a set of autonomous and loosely-coupled Web services, possibly from different service providers. However, Web services based integration builds a web of interdependencies between collaborating systems and introduces various challenging interoperability and management issues as participating services may change or behave in unpredictable ways. Hence, composite Web services execution has to be continuously monitored to check compliance to runtime policies in order to detect and adapt to business exceptions and faults.

However, monitoring and runtime adaptability is not yet adequately supported by dominant Web service composition languages, such as WS-BPEL. Additionally the exception handling mechanisms offered by the process orchestration engines do not provide sufficient support for monitoring to detect and handle the broad range of

[1*] MASC project (http://masc.web.cse.unsw.edu.au/) is sponsored by the Australian Research Council (ARC) and Microsoft Australia. We also thank A/Prof. Boualem Benatallah for his comments.
[3*] NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council (ARC).

business exceptions or faults that may occur during the process execution. Moreover, the monitoring and adaptation logic is often scattered across different modules and tangled with the functional specification and implementation of the normal process flow. This negatively impacts maintainability and increases design complexity and development costs. To address the requirements for manageable and adaptive composite Web services, we propose a policy-based approach to runtime monitoring and adaptation to detect and handle business exceptions and faults. The central part of the approach is our lightweight Web service management middleware MASC (Manageable and Adaptive Service Compositions) that performs runtime monitoring and adaptation. For formal specification of policies MASC uses WS-Policy4MASC [3], which is our novel extension of the Web Services Policy (WS-Policy) Framework [5]. This externalization of monitoring and adaptation aspects yields higher degree of flexibility promotes reusability and contributes to keep the specification of the base process simpler and easier to maintain. Another distinctive characteristic of MASC is that it leverages and extends Microsoft .NET 3.0 [4]. The details of MASC approach have been reported in a number of publications, such as [1-3]. This paper presents a summary of MASC architecture and its implementation. It also summarizes our motivating scenarios used to evaluate and demo MASC middleware capabilities.

2. MASC Middleware Architecture and Implementation

The conceptual architecture of MASC, capturing key components and their relationships, is shown in Figure 1.

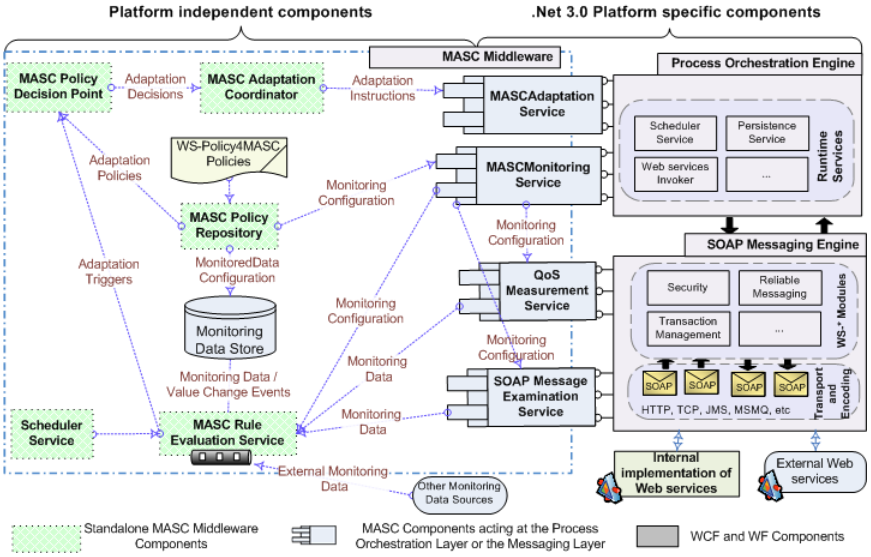

Figure 1- Architecture of MASC middleware and its key Components

We have been implementing its prototype using the newly released .Net 3.0. MASC extends a SOAP messaging engine (the WCF in .NET 3.0) and a process or-

chestration engine (the WF in .NET 3.0). Monitoring and adaptation policy assertions are stored in an in-memory policy repository, which is a collection of instances of policy classes. Dynamic adaptation is started when the MASCMonitoringService module raises an event that for a particular process instance it detected adaptation preconditions specified in monitoring policies. The raised events are handled by MASCPolicyDecisionPoint, which determines adaptation policy assertions to be applied to the process instance and sends an event to MASCAdaptationCoordinator, which calls modules that execute particular adaptation actions. Most often, adaptation is performed by the MASCAdaptationService that is responsible for executing adaptation at the process layer and supports various adaptation actions, such as add activity block, remove activity block, replace activity block, skip, and retry. We have been evaluating the MASC support for policy-based monitoring and adaptation on Stock Trading scenarios implemented with MASC, .NET 3.0, and C#. These evaluations indicate improved reliability of Web service compositions; at the cost of relatively low overheads (e.g., increase in response time).

3. Monitoring and adaptation scenarios

In this demo, the monitoring and dynamic adaptation capabilities of MASC will be illustrated using parts of the Stock Trading case study that we have used to determine requirements for our work in this area and evaluate our solutions.

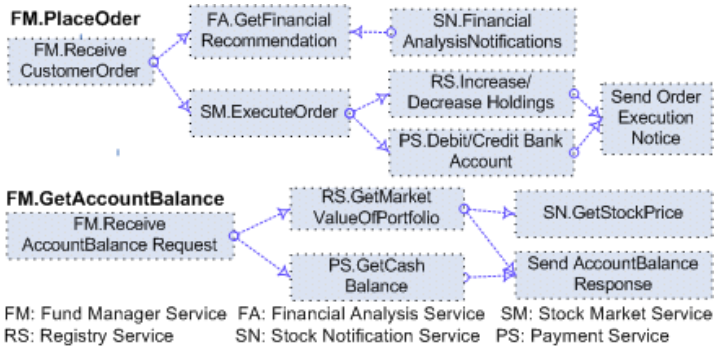

Figure 2: Example Web services interactions in the Stock Trading case study

The base Trading Process, shown in Figure 2, is initiated when a human investor places an investment or redemption order with their FundManagerService. The latter, after verifying the order, invokes the FinancialAnalysisService to get a recommendation to enable an informed investment/redemption decision. The FinancialAnalysis-Service gets periodic notifications from the Stock-NotificationService about the current stock values and real-time market surveillance, announcements, quotes, and other information. Based on this information, historical records, and predictive models built into the service (for our prototype, we used very simple models), the FinancialAnalysisService informs the FundManager-Service about how well certain stocks are performing. The FundManagerService makes a decision which stock to buy/sell for the monetary amount requested by the investor. Then, the FundManagerService sends the buying/selling request to the StockMarketService. The latter performs a simple trade

matching between the buy orders and the sell orders. When a trade match is formed, the StockMarketService invokes in parallel the StockRegistryService to transfer the stock share ownership and the PaymentService to transfer funds.

Several scenarios will be used to illustrate the mechanisms used by MASC for monitoring of composite Web services to trigger timely dynamic adaptation to handle business exceptions and faults. Examples of such scenarios are:

- Monitoring the GetAccountBalance process to examine incoming AccountBalance messages and in case of a foreign portfolio an adaptation trigger can be raised to dynamically add a CurrencyConversion Web service to convert prices of foreign stocks to a local currency.
- Monitoring the invocation of the PaymentService to check the payment amount against the trade amount. When underpayment is detected (i.e., the payment received is less than the trade amount) then a process adaptation could be triggered to calculate the amount still owing and issue a residue invoice to the customer. Whereas in case overpayment the process adaptation could dynamically add a sub-process to notify the customer, calculate the over-paid amount and refund it.
- Monitoring missing events such as the Trade Payment not received within a particular timeframe (e.g., NoPaymentAfter30Days event).
- Monitoring that the response time of the GetStockPrice operation is less than 10 seconds otherwise the returned price should not be considered. In case of service unavailability or timeout, an example reaction could be to cancel the request and to submit a new one to an equivalent service.

4. Conclusions

In this paper, we presented MASC – a policy-based middleware for monitoring and adaptation of Web services compositions. The underlying design principle of our approach is the separation of concerns between the functional process definition and the monitoring and control. The benefits of our approach are of twofold:
(1) A novel language, WS-Policy4MASC, is used to declaratively specify monitoring and adaptation policies for composite Web services (2) The new MASC middleware architecture has been designed and implemented to autonomously make and coordinate enforcement of runtime adaptation decisions across both the business process orchestration layer and the SOAP messaging layer.

References

[1] Erradi, A., Maheshwari, P. and Tosic, V. 2006, 'Policy-Driven Middleware for Self-Adaptive Web Services Composition', in *Middleware'06*, Melbourne, Australia.
[2] Erradi, A., Maheshwari, P. and Tosic, V. 2006, 'Recovery Policies for Enhancing Web Services Reliability', in *IEEE Int. Conference on Web Services 2006 (ICWS'06)*, Chicago, USA.
[3] Erradi, A., Tosic, V. and Maheshwari, P. 2007, 'MASC – .NET-Based Middleware for Adaptive Composite Web Services', in *IEEE International Conference on Web Services 2007 (ICWS'07)*, Utah, USA.
[4] Microsoft 2007, *.NET Framework 3.0*. http://www.netfx3.com/.
[5] W3C Web Services Policy Working Group 2006, *Web Services Policy (WS-Policy) 1.5, November 2006*. http://www.w3.org/TR/ws-policy/.