Integration of UML Modeling and Policy-Driven Management of Web Service Systems

Basem Suleiman¹, ², Vladimir Tosić¹, ², ³

¹ Managing Complexity Research Group – Sydney (ATP), NICTA*, Australia
² School of Computer Science and Engineering, University of New South Wales, Australia
³ Department of Computer Science, University of Western Ontario, Canada

Basem.Suleiman@nicta.com.au, vladat@computer.org

Abstract

We address the problems that Web service management policies are not defined during Web service design and that management information collected during run-time is not directly annotated on designs to guide their improvements. We present novel mechanisms for: (1) generation of WS-Policy4MASC policies (for run-time Web service management) from the corresponding UML profiles, (2) feedback of information monitored during run-time by MASC middleware into another set of UML diagram annotations. The latter annotations show which design elements have not performed well in technical or financial metrics and service designers can use this information for design analysis and re-engineering decisions. We validated our mechanisms on prototypes using Eclipse UML tools and XSLT processing, applied to a stock trading Web service composition example.

1. Introduction

Information technology (IT) system management is the process of monitoring and control to discover and fix problems, maximize quality of service (QoS), enforce security, and account for used resources. It is necessary to achieve dependable IT systems. One frequent-approach to IT system management is based on policies. A policy formally specifies a collection of high-level, implementation-independent, operation and management goals and/or rules in a human-readable form. Policies are enacted during run-time by middleware that collects measures or calculates monitored information and executes control actions.

While it has been recognized that manageability must be an important concern during all lifecycle phases of IT systems, this is not yet a common practice. For example, policies are usually defined only during deployment or run time, and not during design. Furthermore, the current support for using information monitored during run-time for improvement of system design has limitations. More precisely, there is a lot of ongoing research on limited-scope system reconfiguration during run-time [1], but not enough work on annotating designs to guide more substantial off-line improvements by humans.

Another limitation of almost all current IT system management solutions is that they maximize technical metrics, while users actually need maximization of business metrics. For example, higher IT system availability or lower response time need not lead to increases in profits. Unfortunately, the past practice has shown that mapping between business and technical models and metrics is difficult [2, 3]. The goal of business-driven IT management (BDIM) [2] is to determine such mappings and leverage them to make run-time IT system management decisions that maximize business value metrics (i.e., measures of business worth, such as profit or customer satisfaction).

A similar limitation of current software engineering methodologies is that business value metrics information is not explicitly captured in software engineering artifacts (e.g., design models) and used for decision making during software development. Value-based software engineering (VBSE) [3] explicitly considers value issues during software development, in order to make the resulting systems more useful.

In [4], we discussed that integration of achievements on bridging the business-technology gap during design-

* NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.
time (VBSE) and run-time (BDIM) would lead to improvement of business-IT alignment across the whole system lifecycle. We pointed out the need for further results on bi-directional transfer of business value metrics and other management information between design-time and run-time artifacts and on using this information in novel closed-control loops. While these challenging topics are relevant for all IT systems, they are particularly important for modern software systems that can be automatically created or modified during run-time, such as Web service compositions. Therefore, in [4] we also presented UML profiles for WS-Policy4MASC, a policy language [5, 6] for management of Web service compositions that is used in the Manageable and Adaptable Service Compositions (MASC) middleware [7].

In this paper, we present novel mechanisms for:

1. generation of WS-Policy4MASC policies from the corresponding UML profiles,
2. feedback of information monitored by the MASC middleware during run-time into another set of UML diagram annotations.

The former mechanisms reduce effort and time required from system designers to specify and maintain Web service policies, particularly when system design changes. The latter mechanisms provide information that system designers can use for system optimization or re-design. Together, these mechanisms enable bi-directional transfer of Web service management information between design models and run-time management middleware.

In the next section, we present a Web service composition example that illustrates the objectives of our research. Section 3 summarizes the key characteristics of WS-Policy4MASC and the corresponding UML profiles. Section 4 describes in detail our mechanisms and prototype tools. In Section 5, we briefly review the main related work. The last section summarizes conclusions and future work.

2. Motivating example

To illustrate the objectives of our research, we present how an example Web service composition for financial stock trading could benefit from specification of policies and run-time management information in UML models. In the following sections, we will discuss how we achieved these objectives and used the stock trading Web service composition for validation of feasibility and usefulness of our solutions.

The stock trading business process is implemented as a composition of several Web services. As shown in Figure 1, it is modeled as a UML activity diagram (at this time, disregard the stereotypes and notes) where each Web service has its own swimlane. The complete description of all parties and activities in the process is available on our Web site [8]. In this paper, we only point out that the party ShareRegistry executes the TransferOwnership activity that transfers ownership of stocks from the seller through the StockMarket party to the buyer, i.e., the Investor party.

To ensure that this Web service composition performs appropriately during run-time, it can be accompanied with various policies, composed of fine-grained policy assertions. An example policy assertion is that TransferOwnership should be completed within 3 work hours. Such policy assertions (e.g., in WS-Policy4MASC) guide monitoring and control actions by a management middleware (e.g., MASC), while the Web service composition is executing.

In current practice, management policies are usually specified at deployment or run-time. However, policies, or at least policy templates such as ‘activity duration should be less than X hours’, are often known during design-time. To enable their design-time specification, we developed UML profiles for WS-Policy4MASC [4]. A UML profile is a set of stereotypes, tagged values, and constraints. In Figure 1, two stereotypes are applied to the TransferOwnership activity in the stock trading process. For visualization purposes, the details of tagged values of these stereotypes are shown in the note under this activity (at this time, disregard the upper note). The stereotype <<Goal_PA>> contains data for the policy assertion that limits activity duration to 3 work hours and states that this condition should be checked by FundManager when TransferOwnership ends. The stereotype <<Utility_PA>> describes the policy assertion that states that when TransferOwnership ends and the above condition is satisfied, then FundManager pays ShareRegistry AU$1.00. Other policy assertions (not shown in detail in Figure 1, only <<Action_PA>> is written with the activity name) can specify actions to be taken and monetary penalties to be paid when TransferOwnership is not completed on time. Note that most modern policy languages are verbose (for Web services they are in the Extensible Markup Language – XML) and direct use of their syntax in UML tagged values would lead to overcrowded diagrams. One of the challenges in designing a format for representing policies in UML diagrams is to achieve a 1-to-1 correspondence with the run-time policy language while consuming minimal diagram space.

There are at least 3 benefits of specifying policies in UML models. First, this helps think about management-related issues during design and guides their subsequent implementation and testing. Second, it supports
design-time analysis. For example, it is in some cases possible to make predictions of overall process duration based on information about duration of individual activities. Third, this information can be translated into a policy language (e.g., WS-Policy4MASC) used in run-time management middleware (e.g., MASC). While this translation can be largely automated, some human involvement is needed, e.g., because not all information relevant for run-time is known at design time. One of the objectives for our current research is definition of rules for semi-automatic generation of policy files from extended UML models. Such automatic generation requires a 1-to-1 mapping between the UML tagged values and the policy language elements.

Even bigger challenge is feedback of run-time management information into UML models. At run-time, various events and changes occur. For example, the stock trading process executes and management middleware measures that duration of TransferOwnership is 2.1 work hours, evaluates that the specified 3 work hours time limit is met, and accounts payment of AU$1.00 to be made. Figure 1 shows the information about the activity duration and payments within the upper note attached to TransferOwnership. One of the challenges is again achieving a balance of providing much information on limited space. The benefit of showing such information, from one process instance or averages/summaries across multiple process instances, in UML diagrams is to help humans who redesign the system (e.g., software engineers, business analysts) make better decisions. It can be also used for tracking behavior of currently executing Web services and their compositions at a level of granularity and abstraction that is different from the current management dashboards. Such feedback to annotate UML diagrams was not achieved by past works (including [4]), so it is one of our objectives.

3. WS-Policy4MASC and the corresponding UML profiles

WS-Policy4MASC [5, 6] is an XML-based language for formal specification of management policies for Web service systems. During run-time, MASC middleware uses these policies for its Web service composition monitoring and control actions.

WS-Policy4MASC extends the Web Services Policy Framework (WS-Policy) [9]. In WS-Policy, a policy is 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 it applies, such as the Web Services Description Language (WSDL) or the Web Services Business Process Execution Language (WS-BPEL) constructs.

WS-Policy4MASC extends WS-Policy by defining 4 new types of policy assertions [6]:

Figure 1. Extended UML activity diagram for the stock trading Web service composition example

Authorized licensed use limited to: UNSW Library. Downloaded on July 5, 2009 at 22:05 from IEEE Xplore. Restrictions apply.
1. **Goal policy assertions** specify requirements and guarantees to be met in desired operation (e.g., response time of an activity has to be less than 3 seconds). They guide MASC monitoring activities.

2. **Action policy assertions** specify diverse actions to be taken if certain conditions are met (e.g., some goal policy assertions were not satisfied). Examples of actions are removal, addition, replacement, skipping, and retrying of a sub-process or an activity. They guide control (adaptation) activities in MASC.

3. **Utility policy assertions** specify expressions for calculating business value metrics (e.g., profit, cost) assigned to particular situations (e.g., non-satisfaction of some goal policy assertions). They guide accounting activities in MASC.

4. **Meta-policy assertions** specify which action policy assertions are alternatives and which business value metrics driven conflict resolution strategies should be used. They guide BDIM activities in MASC.

The two most important characteristics are the rich specification of diverse (both financial and non-financial) business value metrics in utility policy assertions and the specification of connections of business value metrics and IT system management decisions in meta-policy assertions. The precise definition and meaning of all new policy assertion types were elaborated in our earlier publications (see [6]). These new policy assertion types can be used as any other policy assertion in WS-Policy, together with WS-PolicyAttachment. Apart from them, WS-Policy4MASC defines a number of supporting constructs [6], such as definitions of when policy assertions should be applied and references to external ontologies defining semantics (e.g., of used QoS metrics).

These supporting constructs specify details necessary for actual run-time Web service management activities and overcome some of the limitations of WS-Policy, such as imprecise semantics of policy assertions’ effects on policy subjects. The recent WS-Policy4MASC version 0.9 [5] also adds **probability policy assertions** that specify probabilities that particular situations will occur. They can be used for specification of trust in various parties and risks.

Our UML profiles [4, 8] have 1-to-1 correspondence with WS-Policy4MASC (version 0.8 [6], not yet the recent version 0.9 [5]), but they use a textual format that saves space in UML diagrams. There are several related UML profiles, but the most important one models policy assertions. Every policy assertion type has a corresponding UML stereotype: <<Goal_PA>>, <<Action_PA>>, <<Utility_PA>>, and <<Meta_PA>>. In these stereotypes, UML tagged values correspond to XML attributes and sub-elements defined in WS-Policy4MASC. Constraints in our UML profile correspond to XML Schema constraints (e.g., maxOccurs). Common attributes and elements of all policy assertion types are placed into the abstract UML stereotype <<MASC_PA>> from which all 4 previously mentioned stereotypes inherit. This stereotype extends the <<Metaclass>> element in UML, which enables applying our stereotypes to any UML element (e.g., class, attribute, operation, activity, package, or other).

In practice, we implemented our UML profile for WS-Policy4MASC policy assertions using OMONDO Eclipse 3.3 plug-in [10]. Detailed documentation about this implementation can be found on our Web site [8]. We chose OMONDO Eclipse 3.3 plug-in for many reasons. It has a free edition that is rich in features and can
be plugged into the open-source Eclipse Modeling Framework (EMF). It supports two perspectives relevant for modeling of various UML diagrams and addition of UML profiles: a graphical (Java) perspective and a resource perspective with additional modeling capabilities. OMONDO enables import and export of UML models, including all extensions and annotations, in the XML Meta-data Interchange (XMI) format. XMI is a standard exchange format for UML models and is supported by many UML tools. Unfortunately, we had to deal with the issue that different UML modeling tools generate somewhat different versions of XMI. A short comparative analysis of OMONDO and several other tools for UML modeling is available at [8].

4. Integration of design-time and run-time

Our solution for integrating design-time UML profiles and run-time WS-Policy4MASC and MASC aspects of Web service engineering and policy-driven management consists of two main phases: (a) generation of WS-Policy4MASC files from extended UML models, i.e., forward translation, and (b) annotation of run-time information into UML models, i.e., feedback annotation. In principle, we could have also provided: (c) generation of UML stereotype annotations from WS-Policy4MASC files as backward translation. This is inverse from the forward translation. However, we found that there is currently little practical benefit from backward translation because there are only a few WS-Policy4MASC files written without UML profiles. Thus, this is left for future work.

Since the feedback annotation is a unique and previously unpublished competitive advantage of our research, this paper puts more emphasis on it. However, we also explain our novel mechanisms for the forward translation. The whole process with the phases (a) and (b) listed above is illustrated in Figure 2. It consists of 9 steps (steps 1-4 are for forward translation, steps 5-9 are for feedback annotation):

1. Various UML diagrams (e.g., activity, sequence, class, package) model different features and components of a particular Web service composition at runtime. Our UML profiles capturing the essence of WS-Policy4MASC policies were explained in [4] and are posted at [8]. They are applied to UML diagram elements. (As explained in [4], the profiles can, in principle, be applied to any UML element.) An example was shown in Figure 1 discussed in Section 2. Due to limited space on UML diagrams, some lower-level information, such as namespaces and URIs of Web services implementing particular activities necessary for generation of WS-Policy4MASC and, particularly, WS-PolicyAttachment files is specified in additional XML configuration files.

2. This UML model is exported to XMI that contains a representation of the model and the applied policy assertions. The exported XMI for the activity diagram of the stock trading example can be found at [8]. Exported XMI files and configuration files can be stored in a model repository (e.g., a database).

3. Extensible Stylesheet Language Transformations (XSLT) rules are used to extract policy assertions from the XMI file and details from configuration files and to fill in our template files with the extracted information to produce WS-Policy4MASC and WS-PolicyAttachment files. These XSLT rules can generate almost all details in WS-Policy4MASC and WS-PolicyAttachment files, but in some cases human intervention is needed. For example, while the current rules can process some simple expressions, more complicated expressions require human intervention. (This is being addressed in ongoing work.) In addition to implementation-specific XSLT rules, we developed generalized algorithms and mapping tables for generation of WS-Policy4MASC files from extended UML models. Due to the limited space in this paper, the high-level summary of our forward translation and the details of the algorithms, mapping tables, and XSLT rules are posted at [8].

Our choice of XSLT requires explanation. XSLT is an XML-based language for specification of rules for transforming XML documents into other document formats, such as XML, text, and others. Many XSLT tools are available – we used Oxygen XML Editor 9.1 that supports different XSLT processors. Unfortunately, writing and maintaining XSLT rules is not an easy task. Thus, many authors nowadays use more powerful model-driven development (MDD) frameworks, such as AndroMDA [11], to write model transformations. While we looked at some alternatives, we chose XSLT because we believed that for our project its benefits outweighed its weaknesses. First, XSLT is still widely used in both industry and academia, an indication of its capability of performing some useful transformations. Second, there are many XSLT tools, user groups, and forums, which is important for troubleshooting during development. Third, XSLT is simpler to learn than alternative frameworks, e.g., AndroMDA. Fourth, although the verbose nature of WS-Policy4MASC requires many XSLT rules, the 1-to-1 mappings between UML profiles and WS-Policy4MASC make our forward and feedback transformation rules simpler.

4. The generated WS-Policy4MASC files contain management policies that can be used by MASC. The MASC middleware contains a number of modules, only...
a few of which are shown in Figure 2. While further details are available in [7], the MASC policy parse processes WS-Policy4MASC files and stores their C# implementation into a policy repository.

(5) The database within MASC stores the run-time information collected by MASC, including all run-time events and run-time values of QoS and business value metrics. An example of stored information is: “At 2008-06-04T11:34:44.156Z, FundManager measured duration of activity TransferOwnership for process instance X to be 2.1 work hours”. This information can be from particular process instances or summary information (e.g., average activity duration) across a number of instances. Such information is the key input into our feedback annotation mechanisms.

(6) The MASC run-time monitoring database is queried to retrieve run-time information relevant for a particular UML diagram or its part. This is a ‘pull’ operational mode. Our solution also supports a ‘push’ operational mode in which notification events are sent from the run-time database. The requested information is returned in an XML-based format.

(7) For this retrieved run-time monitoring information, the XMI file of the corresponding UML model is determined and, if necessary, retrieved from the model repository. Usually, this XMI document contains the (graphical) representation of whole project package including UML models. The XML run-time information file and the XMI file are then used along with another set of XSLT transformation rules (available at [8]) as input to an XSLT processor. These XSLT rules match run-time monitoring information with corresponding policy assertions in the extended UML model and format it in a condensed textual format. This representation of run-time information is then added into a UML note annotated to the corresponding UML model element (e.g., activity, object, or other) to which the related policy assertions were applied. An example was shown in Figure 1 discussed in Section 2. The result of this transformation is an XMI file that contains the original UML model, design-time policy assertions and run-time monitoring information corresponding to these policy assertions. Such XMI file is automatically generated for the stock trading example is available at [8]. We also developed generalized algorithms and mapping tables for generation of UML models annotated with run-time monitored information collected by MASC. While their details are at [8], Figure 3 gives a high-level summary.

(8) The generated XMI of the new UML model is then stored in the model repository. It can be used for different purposes, such as visualization of design-time policies and run-time information and/or various analyses, such as business value metrics performance analysis or technical QoS predictions. Some of these analyses can be automated, but an important feature of UML annotations is that humans can use them for more complex analyses and system re-designs.

(9) The generated XMI file can be imported into a UML tool to visually see the feedback information. The upper note in Figure 1 is an example of this visualization. The re-designed UML models can contain changes of policies. They can be again exported into XMI and translated, through steps 1 to 4, into new WS-Policy4MASC files for use by MASC. This closes the engineering and management circle.

We implemented a proof-of-concept prototype based on OMONDO Eclipse 3.3 plug-in and Oxygen XML Editor 9.1. The prototype includes definition of the UML profiles for WS-Policy4MASC, XSLT rules for transformation from XMI and configuration files into WS-Policy4MASC/WS-PolicyAttachment and XSLT rules for transformation from the XML format of run-time information into XMI. We applied these solutions primarily to UML class and activity diagrams. Analogous XSLT transformation rules can be developed for the other types of UML diagrams. This is subject of our ongoing work. To demonstrate both feasibility and usefulness, we used the weather report Web service example described in [4] and the stock trading Web service composition described in Section 2. Fig-
Figure 1 is a result of the evaluation of our prototype. While we used our solutions only in laboratory settings (not yet in industry), there are strong indications about their usefulness in practice. Details about our prototype and its evaluation are at [8].

5. Related work

Our past work on WS-Policy4MASC and the MASC middleware was influenced by works in the policy-driven management and BDIM communities. The discussion of competitive advantages and original contributions of our past research compared with the other policy languages and Web service management middleware, such as [12], is outside the scope of this paper, but can be found in [4-7].

Several competing UML profiles for WSDL and/or WSBPEL concepts were published, e.g., [13, 14]. Further, several UML profiles, such as [13], support modeling of WS-Policy concepts. The goal of our research is not another profile for WSDL, WSBPEL, or WS-Policy, but a profile with unique WS-Policy4MASC concepts (notably, the new types of policy assertions) compatible with and additional to these past works. In addition, there are now many UML profiles, e.g., [14], for specification of various extra-functional properties (mainly QoS) of Web service systems, in the way that can be used for automatic generation of monitoring code. However, most of the past UML profiles are not detailed enough for run-time QoS monitoring, because they do not describe precisely when, where, and/or by which party it is performed. WS-Policy4MASC and our UML profiles provide detailed modeling of not only QoS, but also business value metrics and business strategies for BDIM. Thus, our work goes significantly beyond the previous UML profiles.

There is also a huge body of work in the broad area of model-driven development (MDD). Some MDD frameworks, such as [15], incorporate business aspects into UML artefacts or business process models to automatically generate business specifications in various system models. However, they did not research linking design-time and run-time artifacts for Web service engineering and management, particularly not for BDIM. Other specific MDD approaches [13, 16] enable automatic transformation of UML models into run-time executables. The approach in [13] enables Web service composition and execution using UML activity diagrams and XSLT transformation rules. Web service compositions are modeled in UML activity diagrams and transformed into two executable languages: BPEL4WS (an early version of WS-BPEL) and WorkSco. Contrary, our forward engineering mechanisms transform extended UML diagrams with Web service policy assertions into WS-Policy4MASC files that guide run-time Web service monitoring and control activities executed by the MASC middleware. Furthermore, our feedback mechanisms transform run-time monitoring information and annotate them to corresponding UML model elements to support human analysis and re-design activities of Web service systems. The UML Model Transformation (UMT) tool [16] has capability to do UML-to-text transformation (forward translation) and text/code-to-UML transformations (backward translation). UMT enables several transformations such as from UML to J2EE and XDoclet, WSDL-to-UML and UML-to-WSDL, UML-to-BPEL4WS. However, it cannot transform our extended UML models into WS-policy4MASC monitoring files and also cannot transform MASC run-time monitoring information into UML annotated models. Since we do not transform UML into WSDL and WS-BPEL, there is compatibility between our work and the past results from [13, 16], so they could be used together.

Workflow Management Systems (WfMS) [17] and WSBPEL frameworks, such as [18], that have visualization of status of executing workflows or Web service compositions are also close works to our research. Compared to such WfMS and WSBPEL frameworks, our work has several distinctive characteristics. We elaborate the two main ones. First, it contains forward translation from extended UML models into run-time system management artifacts. UML is the most widely used modeling language. Business processes modeling notations used by WfMS and WSBPEL frameworks have some advantages over UML activity diagrams, but UML also contains various other diagram types (e.g., class, state) that are not supported by WfMS and WSBPEL frameworks and that, taken together, more completely model an IT system. As mentioned above, our UML profiles can be applied to any UML element. Second, our work has a unique support for feedback of run-time information into UML models, different in level of abstraction and granularity from what the current management dashboards provide. Our solution can associate run-time information with not only deployment nodes and process activities as in WfMS and WSBPEL, but also any other UML element (e.g., a class). This enables additional analysis.

6. Conclusions and future work

To facilitate management of IT (e.g., Web service) systems, management issues must be considered and supported in all lifecycle stages, from design-time to run-time and back to re-design time. In particular, bi-
directional transfer of management information between design-time and run-time models is needed for better IT system engineering, management, and analysis and re-engineering based on feedback from run-time systems to design-time models.

In our previous work on the WS-Policy4MASC language [6] and the MASC middleware [7], we provided original contributions on business-driven management of Web service systems during run-time. Our UML profiles for WS-Policy4MASC [4] enabled specification of relevant management information during design-time, in UML models. This paper presented the mechanisms for semi-automatic generation of WS-Policy4MASC policies from our UML profiles, as well as the mechanisms for feedback of information monitored during run-time by MASC middleware into another set of UML diagram annotations. For example, these annotations can show measured activity duration, whether activity duration limits were met and what prices/penalties have to be paid. Our implementation of these mechanisms is based on XSLT processing. Our prototype uses OMONDO Eclipse 3.3 plug-in and Oxygen XML Editor 9.1 and adds definition of the UML profiles for WS-Policy4MASC, XSLT rules for transformation from XMI of extended UML models to WS-Policy4MASC and XSLT rules for transformation from the XML format of run-time information to XMI of annotated UML models. Our application of this prototype on examples, such as the stock trading process in Figure 1, demonstrated feasibility of our solutions and indicated their usefulness for practice. The feedback information can be used (by software or humans) for design analysis and re-design decisions that improve manageability, business value metrics and technical QoS. In addition, this information can be used for visualization of run-time behavior of the modeled Web service systems.

Our ongoing and near-future work proceeds on addressing several issues. First, we are working on making our UML profiles and XSLT rules compatible with the new WS-Policy4MASC version 0.9 [5]. Second, our current XSLT rules are focused on UML class and activity diagrams. Development of precise XSLT rules for the other UML diagram types is analogous to our past achievements and we plan to complete it soon. Third, adding several annotations with feedback information into a UML diagram can easily make the diagram overcrowded. To address this, we plan to enable that these annotations can be shown minimized in the diagram. When a modeling tool user clicks on a particular minimized UML note, it will be maximized to show all details. Fourth, we plan to work on (XSLT-based) mechanisms for generation of UML stereotype annotations from existing WS-Policy4MASC files (i.e., the backward translation).

7. References