Integration of RESTfulBP with BDIM Decision Making

Qinghua Lu[2,1], Xiwei Xu[2,1], Vladimir Tosic[1,2], Jacky W. Keung[1,2], Liming Zhu[1,2]
[1]Business Adaptation and Interoperation Project, Managing Complexity Theme, NICTA, Sydney, Australia
[2]School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
{Qinghua.Lu, Xiwei.Xu, Vladimir.Tosic, Jacky.Keung, Liming.Zhu}@nicta.com.au

ABSTRACT
Software runtime adaptability is one of the desired quality attributes in modern business process systems. It helps satisfy a variety of users' needs and accommodate diverse business and technical changes, both in the running software and its operating environment. In this paper, we present and demonstrate the application of our adaptation middleware MiniMASC+MiniZinc to business processes designed and implemented using the REpresentational State Transfer (REST) architectural style. We extended MiniMASC+MiniZinc with new autonomic business-driven decision making algorithms to determine which process fragment to execute in a decision point of a RESTful business process. Our new decision making algorithms enable different adaptation decisions for different classes of consumer at runtime depending on business strategies, in a way that achieves maximum overall business value while satisfying all given constraints. We demonstrate the new decision making algorithms in a LIXI (Lending Industry XML Initiative)-compliant loan application process system that was implemented using the REST principles.

Categories and Subject Descriptors
H.1.0 [**Information Systems**]: Models and Principles – *General;* H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services*.

General Terms
Algorithms, Management

Keywords
Business-driven IT management, software adaptation, REST, business process management, middleware, autonomic computing

1. INTRODUCTION AND MOTIVATION
Software runtime adaptability [1] as the ability of a software system to change its functionality during runtime is one of the desired quality attributes in modern business process systems. To improve runtime adaptability of software implementing business processes, we previously adapted the REpresentational State Transfer (REST) principles [2] (used in the World Wide Web – WWW) to business process design and execution. The new architectural style – RESTfulBP [3] introduced several architectural constraints to both process orchestrations and process endpoints. Its distinguishing characteristics are:

- Using the limited set of standard HTTP (HyperText Transport Protocol) operations (GET/PUT/POST/DELETE) on nouns used to represent domain-specific actions for both business process and process endpoints.
- Exposing all process entities and status as URI (Uniform Resource Identification) identifiable resources. These externalized process entities and states can ease the monitoring of the running process instances.
- Describing business processes in a declarative rather than imperative manner and exposing the coordination of activities at design-time using the notion of process fragments. Process fragments are primitive, reusable workflow patterns, representing possible next steps or sub-processes. They are communicated among business process participants to dynamically coordinate their activities.

In contrast to the traditional workflow systems (e.g., using the Business Process Execution Language – BPEL), RESTfulBP models business processes in a declarative way and allows reusable process fragments to be bundled, unbundled and re-bundled flexibly and rapidly. Thus, RESTfulBP provides an adaptable infrastructure support for business process execution, monitoring, and adaptation.

Hereafter, a "RESTfulBP system" means a business process implemented following the RESTfulBP architectural style. When several process fragments can be used in a decision making point of a RESTfulBP system, decision making is required to determine which process fragment to execute in this point. In the work presented in this paper, our software autonomously makes such decisions using a business-driven approach.

The business-driven IT management (BDIM) [4] research area studies models, algorithms, and architectures for mapping business and technical IT metrics and making IT management decisions best from the business viewpoint. Decisions in BDIM are made based on such mappings and are performed to maximize overall business value for the enterprise. Autonomic computing [5] is an approach towards reducing complexity in IT system/service management, where IT systems self-manage themselves using configurable policies with minimal human intervention. Autonomic BDIM is the intersection area of autonomic computing and BDIM– it adds processing of business metrics to decision making components of autonomic systems. [6, 7] discussed that there are many unsolved research challenges in autonomic BDIM, and some of them are relevant for business process management.

Middleware'10 Posters and Demos Track, December 1, 2010, Bangalore, India.

MiniMASC+MiniZinc is an extension of our earlier middleware MiniMASC [8] that implements autonomic BDIM decision making algorithms. The new feature of MiniMASC+MiniZinc is that it finds globally optimal concurrent adaptation of multiple business process instances (old MiniMASC only examined adaptation of each business process instance separately). To optimize concurrent adaptation of multiple business process instances, MiniMASC+MiniZinc integrates the solver for powerful constraint programming language MiniZinc [9].

In this paper, we present and demonstrate the application of our middleware MiniMASC+MiniZinc to RESTfulBP systems, which implements new autonomic business-driven decision algorithms that decide which process fragments to execute in a RESTfulBP system. Our new decision making algorithms can make different adaptation decisions for different classes of consumer at runtime depending on business strategies, in a way that achieves maximum overall business value while satisfying all given constraints. MiniMASC+MiniZinc prototype was implemented using Java, the PostgreSQL database, and the MiniZinc solver.

In the following 2 sections, we explain the architecture of MiniMASC+MiniZinc and our demonstration example of applying MiniMASC+MiniZinc to RESTfulBP systems. The last section summarizes conclusions and outlines some future work items.

2. MINIMASC+MINIZINC ARCHITECTURE

Fig. 1 shows the architecture of our autonomic management system for RESTfulBP systems. There are three groups of components and artifacts: MiniMASC+MiniZinc, file creation components and RESTfulBP systems.

The centre of Fig. 1 represents MiniMASC+MiniZinc – the main part of our solution. It determines how to optimally (i.e., best for business metrics maximization) adapt the managed RESTfulBP system, based on monitored data about the managed system's runtime execution and according to policies in WSPolicy4MASC [7] [10]. WS-Policy4MASC policy language is our extension of the WS-Policy industry standard and defines five new types of WS-Policy policy assertions: goal, action, utility, probability, and meta-policy assertions. Goal policy assertions specify conditions to meet (e.g., response time limits) in correct operation, while action policy assertions specify sets of adaptation actions. Probability policy assertions are used to describe uncertainty. The main WS-Policy4MASC support for business-driven decision making is in utility and meta-policy assertions. Utility policy assertions are used to specify diverse financial and non-financial business metrics, such as cost and customer satisfaction, associated with different situations (states and events) that can occur in the managed system. Meta-policy assertions are used to specify business strategies that determine how to choose the adaptation to be performed when several options (represented with action policy assertions) exist. WS-PolicyAttachment part of WS-Policy defines how to associate a policy (a collection of policy assertions) with subjects to which it applies.

The most important part of MiniMASC+MiniZinc is the Policy Conflict Resolution module, which implements the adaptation decision algorithm that decide which of the adaptation options should be executed. The Calculation of Business Metrics sub-module of the Policy Conflict Resolution module calculates all relevant business metrics associated with triggered adaptation options and passes the results to the Selection among Alternatives sub-module. The MiniZinc solver located in the Selection among Alternative sub-module solves the MiniZinc constraint programming model (this model corresponds to the WS-Policy4MASC policies) that is populated with the business metric results from the Calculation of Business Metrics sub-module. The solution is the globally optimal (in the sense that it maximizes overall business value) set of adaptation decisions (1 decision for each affected class of consumer). While the calculation of business metrics could have also been done using MiniZinc instead of the sub-module Calculation of Business Metrics (implemented in Java), our architectural solution is faster. It combines the speed of calculation in Java with the MiniZinc's ability to handle complex decision cases.

We now describe the other modules in MiniMASC+MiniZinc. WS-Policy4MASC policies are first parsed by the Policy Parsing module and then stored in the Policy Repository module. The Database of Monitored Data stores runtime data about monitored technical metrics (e.g., measured response time, calculated availability), business metrics (e.g., paid prices and penalties), and events coming from the RESTfulBP monitoring modules, which are processed by the Processing of Monitored Data module. Based on the recent monitored data (particularly events) and historical information stored in the Database of Monitored Data, the Determining Triggered Policies module determines which WS-Policy4MASC policy assertions should be executed. If more than 1 action policy assertion (each representing an adaptation option) are triggered at the same time, the Policy Conflict Resolution module decides the optimal adaptation, as discussed above.

On the left-hand side of Fig. 2 are optional components and artifacts in our system. Their main purpose is to help users in creation of WS-Policy4MASC files, WS-PolicyAttachment files, and MiniZinc model files. We described them in appropriate detail in [8].

On the right-hand side of Fig. 2 are components of managed RESTfulBP systems. The RESTfulBP runtime engine [3] is used for execution of business processes with the selected process fragments. It provides an environment that allows runtime exchange of process fragments between process participants. In RESTfulBP, process fragments to be executed by a particular participant are flexibly selected based on local information and business strategy. In our integration of RESTfulBP with BDIM, these selections are made by MiniMASC+MiniZinc to maximize overall business value for the particular participant (for which the decision is made). RESTfulBP systems expose all relevant process information as URI-identifiable resources. In this way, there is inherent visibility in the status of process instances (incl. class of consumer that uses it and the current position in process execution). For monitoring of technical quality of service (QoS) metrics (e.g., response time, availability), additional Monitoring Tools can be used.

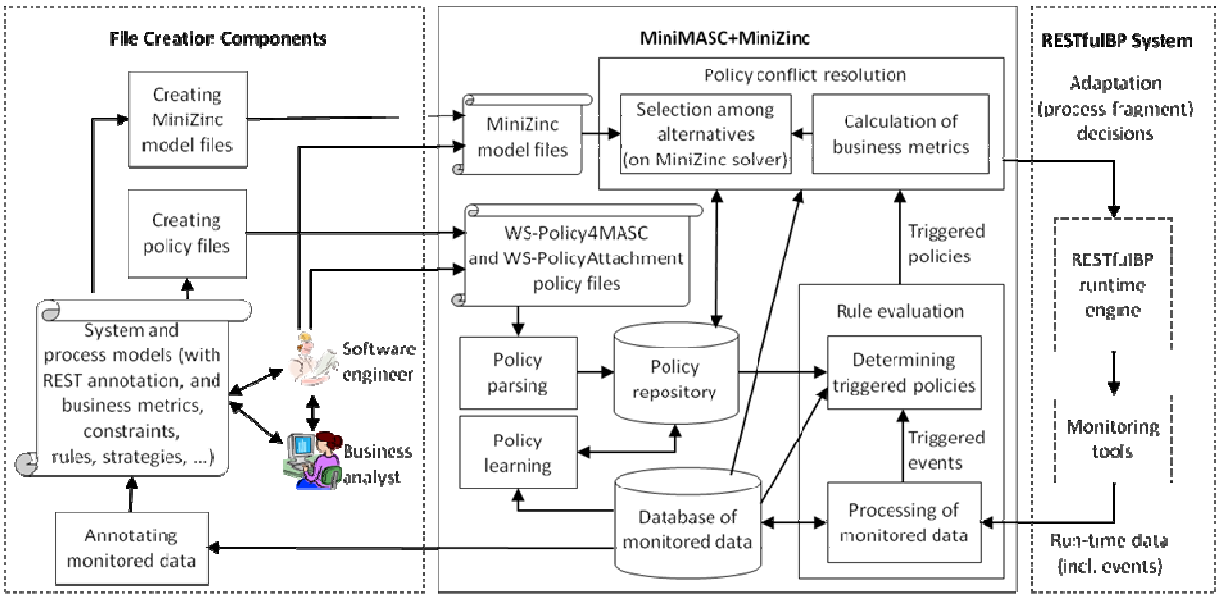

Fig. 1. Architecture of our management system for RESTful BP systems.

3. DEMONSTRATION EXAMPLE

We use the loan application process using LIXI (Lending Industry XML Initiative) standards [11] as an example to demonstrate our solutions. The main process actor is the lending institution I. However, it interacts with additional process fragment providers to create the actual process at runtime. Credit bureau collects information from various sources and provides the credit information on individual consumers. Valuation firm estimates the property value through inspection. Mortgage insurer provides the insurance service aimed at compensating lenders for losses due to loan defaults. Settlement agent contacts the loan requestor's conveyancer to complete necessary government paperwork after the loan is approved.

There are many points in LIXI loan application processes where different process fragments can be used and advanced decision making is necessary. However, due to the space limitation, this short demonstration paper only discusses the decision making of the "Check property value" sub-process shown in Fig. 2. During the valuation process, an extra travel fee due to a remote location or a complex property type may be requested by the valuer. The valuation firm V provides three process options to deal with the fee negotiation. The fee negotiation could be completed either before or after the inspection process. That means the additional fee request could be sent before or after the inspection (For customers with relatively low credits, the fee negotiation may be completed before the actual inspection process). The extra fee could be also removed if it is not big compared to the total valuation fee. These process fragment options can be varied due to changes in business or technical operating environment. Therefore, decision making for business process adaptation is required in this kind of business processes. For the "Check property value" activity, different executable process fragments can be selected for different classes of customers to achieve maximum overall business value (for the lending institution I) while satisfying various constraints (e.g., cost limit constraints).

Lending institution I classifies its customers into three classes according their credit records: gold, silver, and bronze. Table 1 shows the loan application system's guaranteed technical QoS (availability and response time), prices per invocation, penalties if the guarantees are not met, and historical probabilities to meet the guarantees.

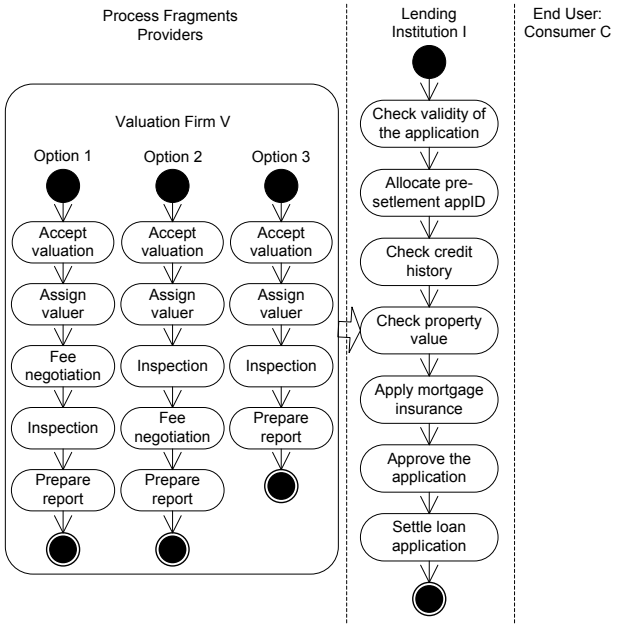

Fig. 2. Part of the RESTfulBP system for a LIXI loan application process

Table 1. The loan application system

Class	Availability guarantee	Response time guarantee	Price per invocation	Penalty when guarantees are not met	Probability to meet guarantees
Gold	98%	10 ms	$300	$500	98%
Silver	96%	20 ms	$200	$200	94%
Bronze	94%	40 ms	$150	$50	90%

We now illustrate how MiniMASC+MiniZinc decides process fragments in the decision making point of the loan application RESTfulBP system shown in Fig. 2. Each process fragment can be viewed as an adaptation option and represented as a WS-Policy4MASC action policy assertion (APA). To determine how to adapt, MiniMASC+MiniZinc calculates business value and cost of each adaptation option according to the policies in WS-Policy4MASC. For the above example, Table 2 shows classes of customers in the system, example number of currently running instances in each class, as well as calculated business value (V) and cost (C) in US$ of each adaptation option for each class of customer. The overall cost limit is US$15000. The problem is modeled as a constraint satisfaction problem and the MiniZinc solver finds the optimal solution that maximizes overall business value. For the above example, the decisions made by MiniMASC+MiniZinc are bolded and underlined in Table 2.

Table 2. Business value (V) and cost (C) of different adaptation alternatives for different classes of service

Class of customer	Gold	Silver	Bronze
No. of instances	10	20	50
Option 1	V: 7000 C:1000	V: 11000 C: 4000	**V: 14000** **C: 5000**
Option 2	V: 8000 C: 1500	**V: 12000** **C: 3000**	V: 13000 C: 75000
Option 3	**V: 9000** **C: 2000**	V: 10000 C: 2000	V: 12000 C: 10000
Selected Adaptation	**Option 3**	**Option 2**	**Option 1**

The software demo will first show how to parse WS-Policy4MASC policies and events into the database Then it will show the decision result. The software demo will run on our own laptop.

4. CONCLUSIONS AND FUTURE WORK

This paper presents and demonstrates the integration of RESTfulBP systems with our new middleware MiniMASC+MiniZinc, which implements new autonomic business-driven decision algorithms for determining which process fragments to execute in the managed RESTfulBP systems. Both the use of the RESTfulBP architectural style and the management middleware MiniMASC+MiniZinc increase software runtime adaptability. Their novel integration enables autonomic business-driven adaptations in complex, multi-customer operational circumstances. Particularly, the decision making algorithms in MiniMASC+MiniZinc concurrently provide adaptation decisions for different classes of customer at runtime depending on different business strategies and operational circumstances, in a way that achieves maximal (globally optimal) overall business value while satisfying all given constraints. MiniMASC+MiniZinc prototype was implemented using Java, PostgreSQL database, and the MiniZinc solver. We seek industry partners for industrial evaluation and application of our proposed technology, for business processes and other complex distributed systems.

5. ACKNOWLEDGMENTS

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.

6. REFERENCES

[1] R. Taylor, *et al.*, "Architectural Styles for Runtime Software Adaptation," in *Proc. of ECSA 2009*, IEEE, 2009, pp. 171-180.

[2] R. Fielding, "*Architectural Styles and the Design of Network-based Software Architectures*," University of California Irvine, USA, 2000.

[3] X. Xu, *et al.*, "An Architectural Style for Process-Intensive Web Information Systems," in *Proc. of WISE 2010*, Hong Kong, China, Springer, 2010.

[4] J. Sauve, *et al.*, "An Introductory Overview and Survey of Business-Driven IT Management," in *Proc. of BDIM 2006 workshop at NOMS 2006*, IEEE, 2006, pp. 1-10.

[5] J. O. Kephart and D. M. Chess, "The Vision of Autonomic Computing," *Computer,* vol. 36, no. 1, pp. 41-50, 2003.

[6] C. Bartolini, *et al.*, "Proceedings of the Second IEEE/IFIP Workshop on Business-Driven IT Management," IM 2007 workshops, IEEE, 2007.

[7] V. Tosic, "Autonomic business-driven dynamic adaptation of service-oriented systems and the WS-Policy4MASC support for such adaptation," *Intl. J. of Systems and Service-Oriented Eng. (IJSSOE),* vol. 1, no. 1, pp. 79-95, 2010.

[8] Q. Lu and V. Tosic, "MiniMASC: A Framework for Diverse Autonomic Adaptations of Web Service Compositions," in *Proc. of ANS2010 workshop at ATC 2010*, IEEE-CS, 2010.

[9] N. Nethercote, *et al.*, "MiniZinc: Towards a starndard CP modelling language," in *Proc. of CP 2007*, Springer, 2007, pp. 529-543.

[10] V. Tosic, *et al.*, "WS-Policy4MASC - A WS-Policy Extension Used in the Manageable and Adaptable Service Compositions (MASC) Middleware," in *Proc. of SCC 2007*, IEEE-CS, 2007, pp. 458-465.