International Journal of u- and e- Service, Science and Technology
Vol.8, No.1 (2015), pp.173-188
http://dx.doi.org/10.14257/ijunesst.2015.8.1.16

# Business-Driven Process Fragment Selections in RESTful Business Processes

Qinghua Lu[1,2], Xiwei Xu[2], Weishan Zhang[1], Liming Zhu[2] and Shanshan Li[1]

[1]*College of Computer and Communication Engineering, China University of Petroleum, Qingdao, China*
[2]*Software Systems Research Group, NICTA, Sydney, Australia*
*dr.qinghua.lu@gmail.com*

***Abstract***

*Past work showed that runtime adaptability of business processes can be improved by applying Representational State Transfer (REST) to design and implementation of business processes. However, the existing solutions for RESTful business processes (RESTfulBP) were focused on manual selection of process fragments to be composed at runtime. In this paper, we propose solutions that enable semi-automatic selection of process fragment at each decision-making point of RESTfulBP. The new built-in middleware MiniZnMASC can provide user based process fragment advice to knowledge workers in ways that achieve better overall business value while satisfying all existing constraints. In addition, we redesign the architecture of RESTfulBP in order to allow the business-driven decision-making solutions. The solutions are evaluated for feasibility, functional correctness, business benefits, and performance.*

***Keywords:*** *Business-driven IT management,adaptation, RESTful, business process management, decision-making, autonomic computing*

## 1. Introduction

One of the most desired quality attributes in business processes is runtime adaptability [6] that is the ability of a software system to change its functionality during runtime. Our RESTful business processes (RESTfulBP) [7] is an architecture style which utilises REpresentational State Transfer (REST) techniques [4] in business process design, implementation and execution to improve adaptability of business processes. RESTfulBP has two significant characteristics compared to flow-based business processes (e.g. using BPEL). First, in RESTfulBP, business processes are modelled in a declarative way and reusable process fragments are bundled, re-bundled and unbundled rapidly and flexibly. Second, RESTfulBP utilises the HTTP protocol to provide a lightweight infrastructure for flexible business process execution, monitoring, and adaptation.

In RESTfulBP, decision-making points are where process fragments can be bundled, unbundled, and re-bundled. When more than one process fragments can be used in a decision making point of RESTfulBP, it is necessary to determine which of the business processes to be bundled. In previous design of RESTfulBP, process fragment is selected by a human called "knowledge worker", who is authorized to manage business process instances. However, knowledge workers can hardly have full knowledge about the RESTfulBP runtime states and understand various interdependencies between diverse metrics and components. It is also difficult for knowledge workers to manually calculate business value of each alternative process fragment and determine which

ISSN: 2005-4246 IJUNESST

process fragment to select taking into account business strategies of the business process owner organisation. Therefore, business-driven decision-making support solutions are needed in order to assist knowledge workers to select process fragments at runtime.

Our middleware MiniZnMASC [4] is an autonomic business-driven decision-making middleware for adaptation of business processes, which is an extension of our earlier middleware MiniMASC [5]. MiniZnMASC much better handles situations when multiple running business process instances have to be adapted at the same time. In such situations, MiniMASC examines adaptation of each business process instance in the same way and has much less powerful support for considering various constraints (e.g., resource limitations). MiniZinc [10] is a powerful constraint programming specification language, which has a free efficient constraint solving software. MiniZnMASC integrates the MiniZinc solver into MiniMASC and implements decision making algorithms that can concurrently make different adaptation decisions for different classes of business process instances in a way that achieves maximum overall business value while satisfying all given constraints. A class of instance is a group of business process instances that share a combination of characteristics that warrants adaptation in the same way. The most important among these characteristics are: the implement business process type, the current position within the running business process, and the class of user. These decision-making algorithms use information specified as policy assertions in our WS-Policy4MASC language [8].

In our past publication [11], we presented an adapted version of MiniZnMASC, a built-in decision-making middleware designed for RESTfulBP, which can provide runtime decision-making support to knowledge workers at each decision-making point of the RESTfulBP. However, we have not presented our decision-making algorithms implemented in the built-in MiniZnMASC and architecture of RESTfulBP with built-in MiniZnMASC which can show the extensibility of MiniZnMASC. In this paper, we systematically present our overall decision-making problem model, decision-making algorithms and architecture of RESTfulBP with built-in MiniZnMASC. The built-in MiniZnMASC first selects different groups of process fragments for different classes of users, in ways that satisfy the business value constraints and other constraints (e.g., cost and resource constraints). The knowledge worker then examines each group of process fragments advised by MiniZnMASC and finally selects a process fragment for each class of user. We analyze challenges of building MiniZnMASC into RESTfulBP and designed the architecture of the new integrated system in order to solve the challenges.

The rest of this paper is organized as follows. Section II introduces a motivating example. Section III examines the main related work. Section IV discusses in detail the proposed solutions. Section V evaluates RESTfulBP with the built-in MiniZnMASC. The last section concludes the paper.

## 2. Motivating Example

In this section, we use the mortgage loan application process using LIXI (Lending Industry XML Initiative) standards [18)] as a real-world scenario to illustrate the need for making different adaptation decisions concurrently for different classes of RESTfulBP system users. In this scenario, the main process actor is the lending institution L that interacts with additional process fragment providers to create the actual process at runtime. Credit bureau B collects information from various sources and provides the credit information on individual consumers. Valuation firm V

 

estimates the property value through inspection. Mortgage insurer M provides the insurance service aimed at compensating lenders for losses due to loan defaults. Settlement agent S contacts the loan requester's conveyancer G to complete necessary government paperwork after the loan is approved. The end user is consumer C (person applying for the loan or her/his representative, such as loan broker).

It is possible that the lending institution L classifies its users into 3 classes according to their credit history and previous loan records: gold, silver, and bronze. Table 1 shows some example loan application process' service levels and prices that the lending institution L offers to its customers. The table lists guaranteed technical quality of service – QoS (availability and response time), prices per invocation, penalties if the guarantees are not met, and historical probabilities to meet the guarantees.

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 limitations, we will only discuss the decision making of the "Check property value" sub-process shown in Figure 1. 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 additional fee request (triggering the "Negotiate fee" process) could be sent before or after the inspection. (For example, the "Negotiate fee" process is often completed before the actual "Inspect property" process for users with relatively low credits and after the "Inspect property" process for users with higher credits.) The extra fee could be also removed if it is not big compared to the total valuation fee. The flexible "Negotiate fee" mechanism may reduce the risk of the lending institution L as well as increase its customers' satisfaction. These process fragment options can be varied due to changes in the business or technical operating environment. Therefore, decision making for business process adaptation is required in this kind of business processes.

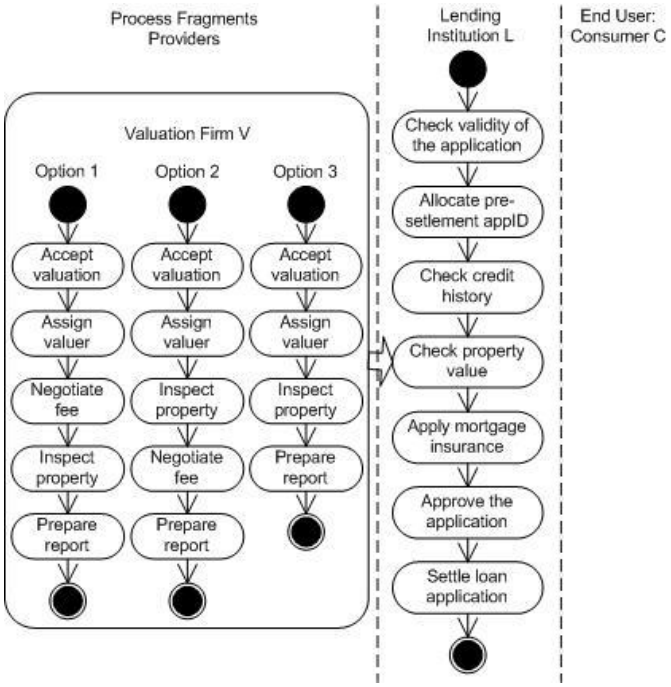


**Figure 1. Part of RESTfulBP for a LIXI Loan Application Process**

One single selection of a process fragment for all users is not appropriate from the business viewpoint. In this example, the "Negotiate fee" process takes some time to be complete and this time depends on the specific users. Therefore, there is a possibility that the completion time guarantee cannot be achieved and penalties will have to be paid to users. Gold users have previous loan records and good credit history in the lending institution L. They are "big" customers as they may bring high profit to the lending institution. It is usually appropriate to remove the extra fee for them. So, different process fragments are needed for different classes of users. Fig. 1 shows the available process fragments for the "Check property value" sub-process. In this context, various business tradeoffs are involved in the adaptation decision making. To determine how to adapt, business value and cost of each option have to be calculated according to some policies. The selected adaptation options have to satisfy various constraints, such as resource constraints and cost limit constraints.

**Table 1. The Loan Application System**

| Class | Availability guarantee | Response time guarantee | Completion time guarantee | Price per invocation | Penalty when guarantees are not met | Probability to meet guarantees |
|---|---|---|---|---|---|---|
| Gold | 98% | 10 ms | 24 hours | $300 | $500 | 98% |
| Silver | 96% | 20 ms | 48 hours | $200 | $200 | 94% |
| Bronze | 94% | 40 ms | 96 hours | $150 | $50 | 90% |

**Table 2. Business Value (V) and Cost (C) of Different Adaptation Alternatives for Different Classes of User in One Example Situation**

| **Class of user** | 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** |

Table 2 shows one example of a runtime situation where there are 3 classes of users, different number of currently running loan application process instances (1 instance per loan application) in each class, as well as calculated business value (V) and cost (C) in AU$ of each adaptation option for each class of user. The overall cost limit is AU$15000. For this example, the adaptation decisions best for lending institution's business are bolded and underlined in Table 2.

In this paper, we will show how our MiniZnMASC middleware makes such decisions based on policies specified in our WS-Policy4MASC language. We have actually used the described example for testing and evaluation of our solution (see Section V).

 

## 3. Related Work

Although REST [8] and Resource-Oriented Architecture [5] principles are well established, their application to business process systems has not been well understood. Several proposals have emerged from both the industry and research community in the last decade, aiming to bring REST to business processes. Most of them extend SOA standards with RESTful interfaces [12] or impose selected REST constraints on business process implementation [13-14, 19-20]. However, there are a number of limitations: Firstly, methods for introducing additional constraints focus only on two constraints: uniform interfaces, and "hypermedia as the engine of application state". They ignore other useful mechanisms of the web infrastructure, such as content negotiation, rich metadata and transfer of process fragments to enable truly distributed and localized process execution. There are very few style implementation guidelines and associated methodologies. Secondly, there is a lack of full support of workflow patterns. Finally, confusion arises when different and sometimes conflicting additional constraints are proposed without a clear definition of the new style and associated methodology for implementing it.

In terms of adaptation work, there are many publications on particular types of adaptation of business processes. In this section we will only point out the most relevant issues and position some representative related works. We provided a classification and a relatively detailed analysis of many additional related works in [9]. While in industry practice adaptation decision-making is still mainly done by human administrators, the vast majority of research focuses on adaptation with minimal help from humans. However, the past adaptation decision-making algorithms predominantly choose adaptation that maximizes technical metrics [1-3, 15-16], while maximization of business metrics is still an open research area. While [17] is not directly on adaptation of business processes, some of its solutions could be reused in our context. It presented a system for maximization of business metrics that schedules the triggered management policies by minimizing the penalty specified in service level agreements (SLAs), but it did not examine resolution for conflicting policies, which is a critical problem in policy-based management.

Our work is different from traditional solutions of optimization problems in business process. We focus on the runtime decision-making support for concurrent adaptation of multiple RESTful business process instances with consideration of business process owner's business strategies.

## 4. RESTfulBP with Built-in MiniZnMASC

In order to provide runtime decision-making support to knowledge workers for the RESTfulBP from the business viewpoint, we integrated an adapted version of our autonomic middleware MiniZnMASC with the RESTfulBP architecture style. The novel integration enables semi-automatic business-driven adaptations in complex, multi-user RESTfulBP with minimal help from human knowledge workers. table 3 outlines the important characteristics of RESTfulBP, MiniZnMASC, and RESTfulBP with built-in MiniZnMASC.

When there are more than one available process fragment alternatives in a decision making point of RESTfulBP, decisions are needed to determine which process fragments to execute. Since the affected business process users can have different characteristics, one single selection of a process fragment for all users can rarely achieve overall maximum business value. It is often necessary to concurrently examine

how to select process fragments for all affected users. In past work [7], this decision was made by a human called "knowledge worker", who is authorized to manage business process instances. However, it is hardly for human knowledge workers to have all required knowledge about RESTfulBP. It is also difficult for knowledge workers to

**Table 3. Comparison of Important Characteristics of RESTfulBP, MiniZnMASC and RESTfulBP with built-in MiniZnMASC**

| | RESTfulBP | MiniZnMASC | RESTfulBP with built-in MiniZnMASC |
|---|---|---|---|
| **Design motivation** | Improve the adaptability of the business processes | Provide autonomic business-driven decision making for adaptation of the business processes | Provide semi-automatic decision making support at a decision making point of the RESTfulBP |
| **Scope of decision making** | Instance-specific | Instance-specific | Instance-specific |
| **Decision maker** | Human knowledge worker | MiniZnMASC | Human knowledge worker with decision making support from MiniZnMASC |
| **Efficiency of decision making** | Situation-related | Highly efficient and situation-independent | Highly efficient, situation-independent, and with minimal human guidance |
| **Goal of decision making** | Not specified | Maximise overall business value | Achieve better overall business value |

manually calculate business value each process fragment alternative leads to and decide which process fragment(s) to execute taking into account business strategies of the business process owner organization. Therefore, we extended MiniZnMASC with new autonomic business-driven decision algorithms determining which of the process fragment alternatives can be executed for particular users in RESTfulBP systems. The main difference is in the used constraint programming model. Different classes of user are adapted concurrently in a way that satisfies various constraints including business value constraints (*e.g.*, a process fragment should be discarded because the business value it leads to is less than the expected business value), resource constraints, and cost constraints. The chosen adaptations depend on business metrics and business strategies, plus operational conditions (*e.g.* current number of users in each class).

There were four challenges to build MiniZnMASC into RESTfulBP: 1) tailor the previous automatic decision making algorithms to semi-automatic decision making algorithm and the previous BDIM modelling in MiniZnMASC to describe the selection of process fragments in RESTfulBP; 2) provide external monitoring mechanism in the RESTfulBP (not previously provided by the RESTfulBP runtime engine), which can send real-time data in form of XML documents to MiniZnMASC; 3) externally execute the adaptation actions that are invoked by sending messages (or XML documents); 4) design the overall architecture of the application. The solution to the first challenge is

 

explained in Section 4.1 and Section 4.2. The solutions to the last three challenges are discussed in Section 4.2.

### 4.1. Constraint Programming Model

In this section, we represent our decision-making model of the built-in MiniZnMASC of RESTfulBP in constraint programming. The decision-making model discussed in this section is to select a group of process fragment options for each "class of user", which satisfy the given business value constraint and other constraints, for the human knowledge worker to make final selections. We introduced several new types of constraints in the decision-making model discussed in this section.

In RESTfulBP, MiniZnMASC concurrently selects a group of process fragment options (*i.e.*, 1 WS-Policy4MASC action policy assertion) for each class of user, from a set of process fragment options at each decision making point. Table IV summarises the notations for the variables and parameters in the problem model.

**Table 4. Variables and Parameters in the Model**

| Notation | Meaning |
|---|---|
| N | There are N classes of user |
| Xn | The current number of instances in class n |
| Mn | The number of process fragment options for a user in class n |
| An,i | The ith process fragment option for class n. |
| K | The number of business value types that can be reasoned about |
| Vk,n,i | The summary business value of type k for process fragment option An,i |
| Wk | The weight of business value type k |
| Bn,i | $B_{n,i}=\sum_{k\in UsedBVTs}(W_k V_{k,n,i})$, (n=1,…,N; i=1,…,Mn), represents the summary business value of all used business value types for process fragment option An,i. The set UsedBVTs contains all business value types that are deemed relevant for comparing business worth of process fragment options. It is specified in the applied meta-policy, which enables that it can differ across change situations. |
| Jn | The selected process fragment for class n |

The problem of finding the globally optimal set of process fragments can be represented in constraint programming as the task to find the set of N groups of process fragments Jn (n=1,…,N), $J_n \in \{A_{n,i},\ldots,A_{n,M_n}\}$, to satisfy all given constraints. The most important constraint here is business value constraints, which describe business situations when an adaptation option should be discarded because the business value it leads to is less than the expected business value. The business value constraints are modelled as $S=\sum_{i=1}^{N}(X_n B_{n,j_n}) \geq$ BusinessValue. The additional constraints can be of different types, but we emphasize cost limits, time limits, and resource limits, as important in practice. Cost limits describe common business situations when an adaptation option should be discarded because its short-term costs are higher than

available funds. They are modelled similarly to: $|\sum_{n=1}^{N}(X_n \sum_{k \in CostBVTs} W_k V_{k,n,j_n})| \leq CostLimit$. Here, the set CostBVTs contains all business value types that represent relevant costs. It is specified in the applied meta-policy. Another important type of constraints is resource limits. Resources (e.g., memory, processor time, bandwidth, energy, etc.) are often limited and this is one of the main reasons why decisions of concurrent instances should be considered together instead of separately. WS-Policy4MASC enables definition of non-financial business value types that represent resource usage and use appropriate units. For example, it is possible to define that all non-financial costs for which attribute Cause= "Memory" represent memory usage and have unit "GB". Then, a memory limit can be modelled similarly to: $|\sum_{n=1}^{N}(X_n \sum_{k \in MemoryBVTs} W_k V_{k,n,j_n})| \leq MemoryLimit$. The set MemoryBVTs contains all business value types that represent memory usage. Time is often limited in business processes. For example, some certain business process instances must finish a certain task Yp (p=1,…,P) in a business process within 48 hours due to some emergency. Here, time limits are defined as a variant as they might be different depending on the business process instances. So, a time limit can be modelled similarly to: TimeLength(Yp, jn)≤TimeLimit(Yp, jn).

### 4.2. Decision-Making Algorithms

As shown in Figure 2, the decision-making algorithm first loops through all alternative process fragments which are specified as action policy assertions (APAs) in each class of instance n (n=1, ..., N). Business value metrics (BVMs) for particular process fragments are specified within utility policy assertions (UPAs). UPAs correspond to consequences of executing APAs. For each APAi,n in class n, the algorithm finds the related UPAn,i. If use of probabilities are disabled, the algorithm sums all the BVMs in UPAn,i and returns the sum business value BVMSUMn,i. If use of probabilities are enabled, the algorithm loops through all the BVMs in UPAn,i. PPAk,n,i represents the probability that BVMk,n,i is a correct estimate. For each BVMk,n,i, the algorithm finds PPAk,n,i, multiplies BVMk,n,i with PPAk,n,i , and sums them up. The algorithm multiplies the running instance number with the summed value and adds the result to the list. The algorithm checks whether APAn,i satisfies all existing constraints. The action policy assertions (APAs) that satisfy all constraints are added to the list, which are advised to the knowledge worker for final determination.

The complexity of the overall decision-making algorithm shown in Figure 2 is analysed as below. There are a number of interdependent factors which complicate complexity analysis.

- N is the maximal number of classes. For each class, there is loop from the start of the algorithm. Therefore, the algorithm is linear: O(N).
- Mn represents the maximal number of alternative process fragments (i.e., action policy assertions- APAs) in class n (1≤n≤N). For given n and Mn, the algorithm is linear: $O(M_n)$. However, total number of APAs is basically $\sum$ . This is $O(N*M_{max})$ where $M_{max}$ is $M_n$ that is the greatest. The algorithm is product. For more detailed analysis, it needs to know $M_n=f(n)$, which is unrealistic.
- The total number of UPAs by assumptions is the same as the total number of APAs. There, the algorithm is $O(N*M_{max})$.
- For the time taken to find UPAs for APAs, the best case is straightforward to find UPAs matching APA then it is $O(N*M_{max})$. The worst case is if UPA has be compared

 

with all APAs, then this is linear search and in the space of $O(N*M_{max})$. Then for total number of UPAs, it is $O(N*M_{max})*O(N*M_{max})=O(N2*M_{max}^2)$. If UPA search is logarithmic $O(\log2(N*M_{max}))$ $=O(\log_2N+\log_2M)=O(\log_2max(N, M_{max})$. Then for total number of UPAs, it is $O(N*M_{max})*O(\log2(N*M_{max}))=O(N*M_{max}+\log_2max(N, M_{max}))$.

- BVM (business value metrics) has three factors: k, n, i. K is the maximum number of BVM types per UPA. In $BVM_{k,n,i}$ (k=1,...K; n=1,..., N; i=1, ...,$M_n$), k represents type k business value metric. Summation is linear for given n. For given n and m, this is summation. Therefore, it is linear: O(K). For total number of BVMs, it is $O(N*M_{max}*K)$.
- By assumption, there are correspondence 1-to-1 or 0-to-1 between PPAs (probability policy assertions) and UPAs. This means we again have $O(N*M_{max}*K)$.
- J is maximal number of j. For given n and $M_n$, it is O(J). Since it is linear, then for total number of LongTermBV: $O(n*M_{max}*J)$.
- $BVSUM_{n,i}$ depends on $O(N*M_{max})$.

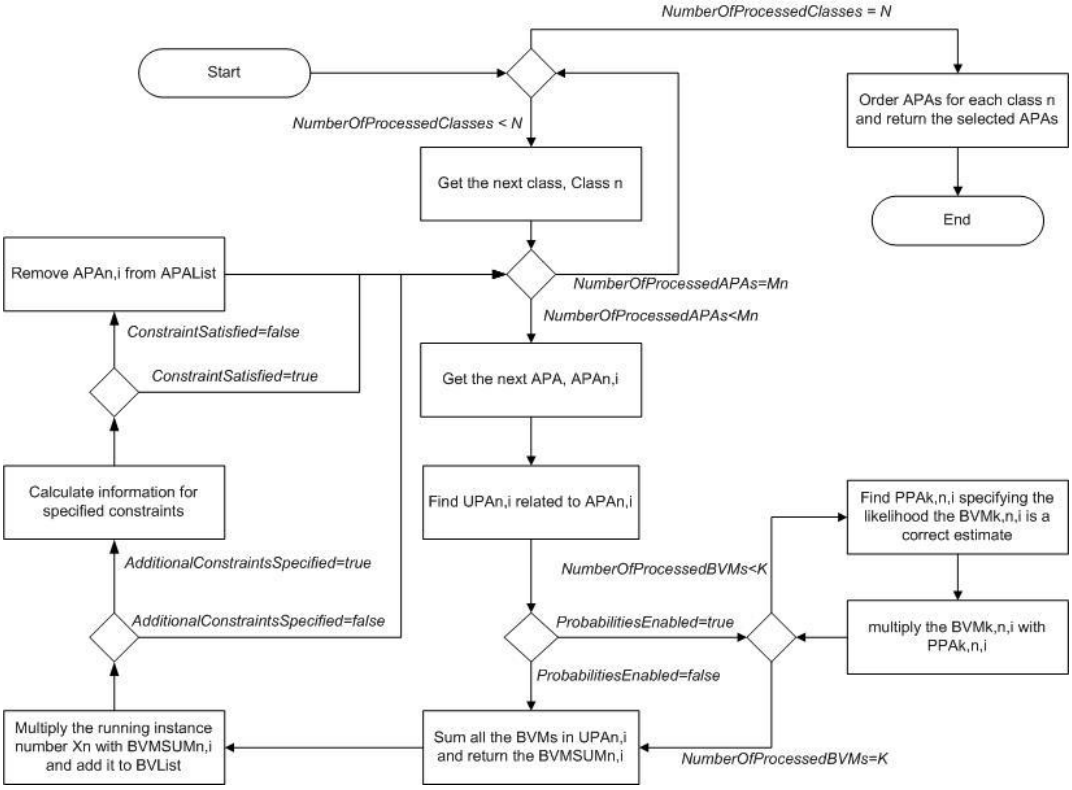


**Figure 2. Decision-making Algorithms in the Built-in MiniZnMASC of RESTfulBP**

### 4.3. Architecture of RESTfulBP with Built-in MiniZnMASC

Figure 3 shows the overall architecture of the new RESTfulBP with built-in MiniZnMASC. There is no change to the architecture of MiniZnMASC, which shows good extensibility of MiniZnMASC. The business process is defined by business analyst(s) using BPMN and RESTfulBP related annotations. The purpose of modelling is to identify the tasks in the business process at different levels of abstraction, the roles of the process participants, the control dependencies between the tasks (*e.g.*, sequencing,

parallel), and existing constraints (*e.g.*, resource and time limitation). RESTfulBP modelling tool is on the basis of Eclipse platform and several existing Eclipse projects, such as BPMN modeller and EMF. BPMN modeller is an Eclipse-based business process diagram editor designed for business analysts. It realizes Business Process Modelling Notation (BPMN) specification. An annotation plug-in is implemented on top of the BPMN modeller by utilizing the extension mechanism of Eclipse. It extends several extension points provided by the BPMN modeller and Eclipse platform itself. The annotation plug-in allows the developer to annotate various elements of the BPMN diagram with RESTfulBP information, which is essential for the code generation engine to generate code. The code generation engine can extract useful information from the diagram and generate process skeletons and constraints of certain tasks or sub-process. It consists of four modules: workflow pattern detector, constraints detector, code generation wizard, and code generation template. The workflow pattern detector can recognize which workflow pattern a certain task belongs to referring to its pre-and-post tasks. The identified workflow pattern is used by code generation template to create the construct of tasks and process fragments. Developers can configure the generated project using code generation wizard, *e.g.* choosing the path of the dependent library. Then the code generation engine combines the information from both the template and wizard to create the executable process. The process fragments extracted from the normal process are the normal process fragments. The process fragments extracted from the compensate process are the compensate process fragments, which are used for the exception handling. The information is stored in an XML file as an input of the process fragment processor. The output of the process fragment processor is the information needed by the MiniZinc models, e.g. the number of available process fragments for each class of user. The tree viewer organizes the tasks and corresponding process fragments into a tree construct. All the tasks are the children of the root node. A task can contain arbitrary numbers of process fragments as its children. The business analyst creates MiniZinc models and WS-Policy4MASC policies according to all relevant RESTfulBP system models and information, *e.g.*, process fragments, constraints, business strategies and operational conditions.

 

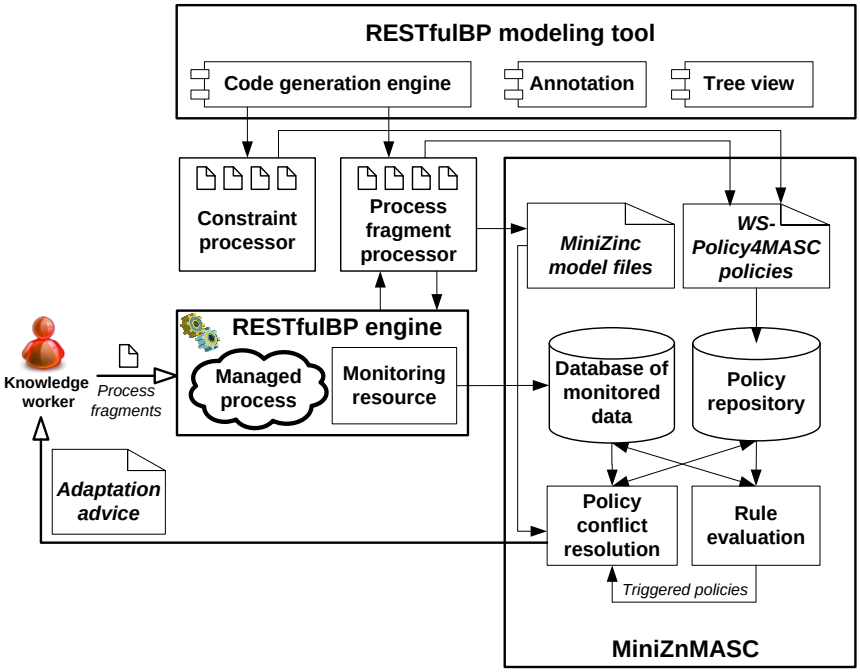


**Figure 3. Architecture of RESTfulBP with Built-in MiniZnMASC**

At runtime, the RESTfulBP engine is used to execute the RESTfulBP-based business processes. RESTfulBP is designed for the human-intensive processes. Thus, a knowledge worker is required to drive the executed process, and the MiniZnMASC is used to assist the knowledge worker to make more suitable decision among a group of possible process fragments. Regarding process execution, RESTfulBP provides a process execution runtime environment that allows process fragments being exchanged between process participants. The process is driven by a knowledge worker, who is responsible to select a suitable process fragment to execute among a group of process fragment alternatives based on his/her personal experience and domain-specific knowledge. MiniZnMASC can assist the knowledge worker in terms of selecting different groups of process fragment alternatives for different classes of user, which satisfy all the existing constraints. The selections made by MiniZnMASC are based on monitored data about the run-time execution and according to policies in WSPolicy4MASC. If changes happen in the operational environment, IT managers may update the constraints and relevant policies in the Policy Repository module at runtime. MiniZnMASC is used to not only assist the knowledge worker to make the decision in normal process execution, but also provide a mechanism for the exception handling.

For process monitoring, the monitoring module of RESTfulBP uses a set of monitoring resources to provide the monitoring functionality. The monitoring resource on the top-level is addressed by a URI, *e.g.* "valuationprocess/monitoring/". It has a set of sub-resources; each represents a kind of runtime data required to be monitored, *e.g.*, the resource identified by the URI "valuationprocess/monitoring/responsetime" represents the response time metric. Similarly, the resource with the URI "valuationprocess/monitoring/availability" represents the availability metric. These quality of service (QoS) metrics can be used to measure different levels of process entities, which are also represented by the sub-resources of the QoS metric resources, *e.g.*, the response time of the process instance can be achieved by accessing the

resource with the URI "valuationprocess/monitoring/responsetime /{instanceId}". Similarly, the response time of a certain task of the same process instance is represented by the resource identified by "valuationprocess/monitoring/ responsetime/{instanceId}/{taskId}". The monitoring resource records all the possible runtime data automatically at runtime. The monitored runtime data is stored in the Database of Monitored Data. The monitored data includes not only technical metrics (e.g., measured response time, calculated availability), but also business metrics (e.g., paid prices and penalties).

For process adaptation, based on the recent monitored data and historical information stored in the Database of Monitored Data, the Rule Evaluation module determines which WS-Policy4MASC policy assertions should be executed. If more than one action policy assertions (each representing an adaptation option, *e.g.* a process fragment option) are triggered at the same time, the Policy Conflict Resolution module decides the optimal adaptation. The most important part of MiniZnMASC is the Policy Conflict Resolution module, which implements the adaptation decision algorithm that decides which of the adaptation options should be executed. The MiniZinc solver located in the Policy Conflict Resolution module solves the MiniZinc constraint programming model that is populated with the calculated business metric results. The solution is the set of groups of adaptation alternatives that satisfy all given constraints, including business value constraints, cost limit constraints, resource limit constraints, etc. For example, the selected process fragment alternative should achieve at least $1000 business value for RESTfulBP owner organisation. The selected process fragments are included in adaptation advice file sent to RESTfulBP engine. RESTfulBP engine inherently supports the adaptation execution for two reasons: first, the execution of RESTfulBP process is different from the execution of traditional flow-based process in that the whole process is generated dynamically at runtime by a set of process fragments rather than statically defined at design-time; second, the adaptation decision advice made by MiniZnMASC is to select different groups of process fragments from all available process fragments for different classes of users in ways that achieve better business value and satisfy various constraints. The adaptation action is invoked by sending a HTTP request (a PUT request).

## 5. Evaluation

We evaluated the new RESTfulBP with the built-in MiniZnMASC and the developed algorithms for business-driven decision-making in five aspects: feasibility, functional correctness, business benefits, performance overhead, and scalability. We evaluated feasibility through implementation of a proof-of-concept prototype. This prototype is implemented in Java and using the Postgre SQL database. We found no problems with feasibility. The new RESTfulBP with built-in MiniZnMASC and the algorithms meet the expected needs of the decision making by using the prototype.

We implemented our motivating example introduced in Section 2 and evaluated the functional correctness of the decision-making algorithm by comparing the results calculated by the built-in MiniZnMASC and by hand. The results show that the decision-making algorithm was built correctly. Based on the business value and cost of different adaptation alternatives shown in Table II, we calculated the business value (V) and cost (C) for different possible single decisions and multiple decisions made by MiniZnMASC shown in Table 5. The results show that the benefits of the decision-

 

making algorithm in the built-in MiniZnMASC is the most compared to other single selections.

**Table 5. Business Value (V) and Cost (C) of Different Decision Making**

| Decision | Business benefits |
|---|---|
| Single selection of Option 1 | V: 3200; C:10000 |
| Single selection of Option 2 | V: 33000; C: 12000 |
| Single Selection of Option 3 | V: 31000; C: 14000 |
| Selections by MiniZnMASC | **V: 35000; C: 10000** |

For the performance overhead and scalability evaluation, we used a Hewlett-Packard laptop model HP EliteBook6930p with Intel Core 2 Duo CPU T900 2.53GHz processor and 4.00 GB of RAM memory, running 32-bit Windows Vista operating system.

We increased the number of conflicting APAs. We started with 3 action policy assertions, then increased to 10, and then to 100. Using the Java "System.currentTimeMillis()" method, we measured the execution time of the overall decision-making (DM) algorithm and the execution time of the confliction resolution algorithm (CRA) part. We repeated 100s of tests at different times of a day and averaged their results.

Table 6 shows the measurement results in distribution of the range (min, max) and their averages of the execution time. The overall execution time of decision-making in MiniZnMASC rises because the execution time of the summation of business value for each conflicting action policy assertion increases with increasing number of conflict action policy assertions and business value metrics. The last test case with 100 conflicting action policy assertions and 64 business value metrics is not realistic. Thus, 12.3 sec is not an issue. It is important to note that in realistic application scenarios of MiniZnMASC to RESTfulBP the number of conflicting action policy assertions will be low (around 2) while overall number of action policy assertions can be huge. Therefore, we did not perform detailed additional tests with much higher number of conflicting action policy assertions. We also checked that the number of additional non-conflicting action policy assertions in policy process (100 action policy assertions) has practically no effect on performance.

**Table 6. Performance Measurement Results with Increasing Number of Conflicting Action Policy Assertions (APAs) and Business Value Metrics (BVMs)**

| Test case | Execution time of DM | Execution time of CRA |
|---|---|---|
| 3 conflicting APAs, 2 BVMs each | Average: 250 ms<br>Range: 234-266 ms | Average: 16 ms<br>Range: 15-32 ms |
| 10 conflicting APAs, 32 BVMs each | Average: 905 ms<br>Range: 889-936 ms | Average: 31 ms<br>Range: 16-32 ms |
| 100 conflicting APAs, 64 BVMs each | Average: 12309 ms<br>Range: 12262-12449 ms | Average: 140 ms<br>Range: 140-156 ms |

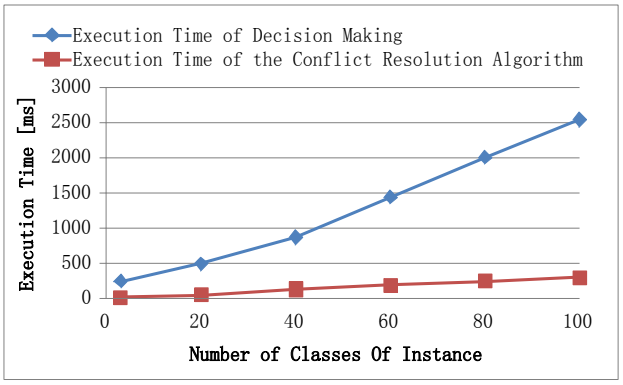


**Figure 4. Performance Measurement Results with Increasing Number of Classes of Instance**

Figure 4 shows the performance measurement results of application of MiniZnMASC to RESTfulBP with increasing number of classes of instance. Both of the measurement results are linear. In application of MiniZnMASC to RESTfulBP, the decision-making component first decides the action policy assertions triggered by the events and sets the utility policy assertions and probability policy assertions for the relevant action policy assertions. Then, if the number of action policy assertions is more than one, application of MiniZnMASC to RESTfulBP runs the Conflict Resolution Algorithm to select among the conflicting action policy assertions. This result shows the majority of the execution time of decision-making of the built-in MiniZnMASC is spent on deciding triggered action policy assertions and setting utility policy assertions and probability policy assertions of them.

## 6. Conclusion

In the previous design of RESTfulBP, process fragment is selected by knowledge workers. It is difficult for knowledge workers to manually calculate business value of each alternative process fragment and determine which process fragment to select taking into account business strategies of the business process owner organisation.

This paper presented a built-in decision-making support tool, MiniZnMASC, for RESTful business processes (RESTfulBP). The novel integration of the RESTfulBP architectural style and the management middleware enables semi-automatic business-driven adaptations in complex, multi-user business process systems with minimal help from human knowledge workers. Particularly, the new decision-making algorithms in the built-in MiniZnMASC concurrently provide different adaptation decision advice for different classes of user at runtime depending on different business strategies and operational circumstances, in ways that achieve better overall business value and satisfy various constraints. In commercial business process systems, such as the LIXI loan application process, our solution focuses on the business value metrics. However, the technical QoS metrics can be also considered in our work, because of the generality of WS-Policy4MASC and MiniZnMASC modules. This could be useful for life-critical or safety-critical business processes.

To achieve the application of MiniZnMASC to RESTfulBP, we developed new business-driven decision algorithms for determining which of the process fragment alternatives can be executed in the managed RESTfulBP. These algorithms required several changes to the constraint programming models and the MiniZnMASC modules.

 

We also extended the tooling for RESTfulBP. In particular, we introduced several kinds of monitoring resources into the RESTfulBP runtime engine to automatically record values of different technical and business metrics. The monitored runtime data is passed to MiniZnMASC in an XML document, which is used in the adaptation decision-making. The prototype for the RESTfulBP with the built-in MiniZnMASC was implemented using Java, PostgreSQL database, and the MiniZinc solver. Experiments with the prototype implementation showed that our solutions are feasible, functionally correct, business beneficial, with relatively low performance overhead, and with satisfactory scalability.

## Acknowledgements

This project is supported by National Natural Science Foundation of China (Grant No. 61402533), "the Fundamental Research Funds for the Central Universities" (No. 14CX02140A) and "the Scientific Research Foundation of China University of Petroleum" (No. Y1307021).

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.

## References

[1] G. Guo, F. Yu, Z. Chen and D. Xie, "A method for semantic web service selection based on QoS ontology," Journal of Computers, vol. 6, no. 2, **(2011)**.
[2] C. Liu and D. Liu, "QoS-oriented web service framework b missed programming techniques," *Journal of Computers*, vol. 8, no. 7**(2013)**.
[3] Z. Yao and A. Sen, "Implementation of the Auto Complete Feature of the Textbox Based Ajax and Web Service," *Journal of Computers*, vol. 8, no. 9, **(2013)**.
[4] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine **(2000)**.
[5] L. Richardson and S. Ruby, "*RESTful Web Services*", O'Reilly Media **(2007)**.
[6] R. Taylor,"Architectural Styles for Runtime Software Adaptation", Proceedings of 2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture IEEE, **(2009),** pp. 171-180.
[7] X. Xu, L. Zhu, U. Kannengiesser and Y. Liu, "An Architectural Style for Process-Intensive Web Information Systems", Proceedings of Web Information Systems Engineering, Springer**, (2010),** pp. 534-547**.**
[8] Q. Lu and V. Tosic, "Support for concurrent adaptation of multiple Web service compositions to maximize business metrics", Proceedings of 2011 IEEE/IFIP International Symposium of Integrated Management, Dublin, Ireland, **(2011)**.
[9] Q. Lu and V. Tosic, "MiniMASC: A Framework for Diverse Autonomic Adaptations of Web Service Compositions", Proceedings of the First International Workshop on Autonomic Networks and Services (ANS2010), **(2010).**
[10] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck and G. Tack, " MiniZinc: Towards a standard CP modelling language", Proceedings of Constraint Programming 2007, Springer, **(2007)**, pp. 520-543.
[11] Q. Lu, X. Xu, V. Tosic and L. Zhu, "Application of business-driven decision making to RESTful Business Processes", Proceedings of 2012 International Conference on Service Oriented Computing, Shanghai, China, **(2012)**.
[12] H. Overdick, "Towards Resource-Oriented BPEL", Emerging Web Services Technology, Springer, vol. 2, **(2008),** pp. 129-140**.**
[13] C. Pautasso, "BPEL for REST", Proc. of BPM 2008, Springer, LNCS, **(2008),** pp. 278-293.
[14] M. zur Muehlen, J. Nickerson and K. Swenson, "Developing Web Services Choreography Standards--The case of REST vs. SOAP" Decision Support Systems, Elsevier, vol. 40, **(2005)**, pp. 9-29.
[15] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal and B. Srivastava, "Adaptation in Web Service Composition and Execution", IEEE, Proc. ICWS, **(2006),** pp. 549-557.
[16] H. Tong and S. Zhang, "A Fuzzy Multi-attribute Decision making Algorithm for Web Services Selection Based on QoS", IEEE, Proce. of APSCC, **(2006)**, pp. 51-57.

 

[17] I. Aib and R. Boutaba, "Business-Driven Optimization of Policy-Based Management Solutions", Proc. of IM, IEEE**, (2007),** pp. 254-263.
[18] LIXI (Lending Industry XML Initiative). Retrieved Jannuary 10, 2014, from http://www.lixi.org.au/.
[19] J. Webber, S. Parastatidis and I. Robinson, "How to GET a Cup of Coffee", Retrieved on 10 January 2014, from: http://www.infoq.com/articles/webber-rest-workflow.
[20] Rest-client, Retrieved on January 10 2014, from http://github.com/caelum/rest-client.

## Authors

**Qinghua Lu**, is a lecturer at the College of Computer and Communication Engineering, China University of Petroleum, Qingdao, China. She received her PhD from University of New South Wales (UNSW) in 2013. Her research interests include software architecture, dependability of distributed systems, and business-driven IT management.

**Xiwei Xu**, received her BEng degree in Software Engineering from Nankai University (NKU), in Tianjin, China, in 2007. She received her Ph.D. degree in the School of Computer Science and Engineering at University of New South Wales (UNSW), in Sydney, Australia, in 2011. She is also with National ICT Australia (NICTA). Her research interests are in software architecture, business process and cloud computing.

**Weishan Zhang**, he is a full professor, deputy head for research of Department of Software Engineering, College of Computer and Communication Engineering, China University of Petroleum. He is the founding director of the Autonomous Service Systems Lab. His current research interests are big data platforms, pervasive cloud computing, and service oriented computing. Weishan has published over 50 papers and his current h-index according to google schoolar is 10.

**Liming Zhu**, he is a senior researcher at NICTA (National ICT Australia). He holds conjoint academic positions and teaches software architecture courses at both University of New South Wales (UNSW) and The University of Sydney. His research interests include software architecture, service engineering and system development methodologies.

**Shanshan Li**, she is a postgraduate student at College of Computer and Communication Engineering, China University of Petroleum. Her research interests include software architecture, API and cloud computing.

 