Resource-Oriented Architecture for Business Processes

Xiwei Xu, Liming Zhu, Yan Liu, Mark Staples
NICTA, Australian Technology Park, Eveleigh, Australia
School of Computer Science and Engineering, University of New South Wales, Australia

Abstract

REpresentational State Transfer (REST) [3] is the set of design principles behind the World Wide Web (WWW). REST treats all entities in the world as link-connected resources, and supports a Resource-Oriented Architecture (ROA) for the design of applications. REST and ROA are responsible for many of the desirable quality attributes achieved in the WWW, such as loose-coupling (better adaptability) and interoperability. However, many exiting Web-based or service-oriented applications (WSDL/SOAP-based) only use WWW/HTTP as a tunneling protocol or abuse URL and POX (Plain Old XML) by encoding method semantics in them. These applications use fine-grained Remote Procedure Calls (RPC), breaking REST/ROA principles. We observe two kinds of challenges: 1) conceptually modelling process-intensive applications using a ROA promoted by the REST principles; and 2) practically decomposing a workflow-based business process into distributed, dynamic and RESTful process fragments. In this paper, we propose a ROA for business processes following the RESTful principles. We evaluate our approach by comparing it with current SOAP/WSDL/BPEL-driven approaches in terms of feasibility, process visibility, interoperability, and adaptability.

1. Introduction

The World Wide Web (WWW) started as a means for transporting data representations using URL-identifiable and inter-connected resources. Its design follows the REpresentational State Transfer (REST) principles [3]. REST principles include: stateless and context-free requests, standardized and unified interfaces, and URL identifiable and inter-linked resources. All interesting information is exposed as “abstract” resources in a Resource-Oriented Architecture (ROA) which is identifiable through URLs and may have multiple representations. A system can manipulate these resources via a uniform interface (e.g. standardized HTTP verbs) and exchange representations of resources. REST is directly reflected in the HTTP 1.1 protocol design and is largely responsible for many of the good characteristics of the Web, such as adaptability, data visibility and interoperability [3].

In addition to data-intensive applications, the WWW has recently been widely used for process-intensive business applications. The design practices behind these process-intensive applications have evolved from early ad-hoc constructions to the recent standardized Web services approaches using WSDL/SOAP/BPEL [7]. This style of web service arose from the history of Remote Procedure Calls (RPC) and Object-Oriented (OO) programming, and is different paradigms to REST and ROA.

It is not obvious how a process-intensive application can fit with a ROA view and follow the REST principles. This has caused a number of problems:

- Fine-grained RPC method parameters are often adopted due to the inertia of traditional RPC/OO-based programming models. They are directly encoded into either URL or ad-hoc XML formats, immediately violating the REST principles [8].
- WSDL/SOAP standards use HTTP as a black-box transport protocol and do not use basic web principles. This has caused misalignment confusions and has not used WWW infrastructure such as caching, proxy and metadata management. Many WS-* standards are being invented to re-achieve these quality characteristics.
- A separate and centralized workflow-based language (such as BPEL [7]) is needed to model the semantics of process coordination. There are no mechanisms within a service to inform invocation sequences in relationship with other services at design time and run time.

These problems have been reflected in the constant debate on REST versus SOAP in building distributed systems [5]. Increasingly, developers are taking advantage of the inherent nature of the Web by building RESTful Web services [8] instead of using WSDL/SOAP/BPEL based services [6].
However, many of the successful RESTful web applications are still limited to data exposure and manipulation [1, 2]. We observe that the conceptual gap between REST/ROA and “processes” is a major barrier for applying RESTful principles to wider problem domains, especially process-intensive ones.

In this paper, we propose a solution for modeling business process concepts [9, 10] using REST/ROA. In particular, we provide solutions for:

1) How to model business processes, instances, tasks and states as URL identifiable resources
2) How to model control flows and state transitions as links between connected resources
3) How to use micro-formats and URL templates for dynamic process coordination

A process coordination engine has been implemented to support the concepts mentioned above. This engine is built on top of an open source REST framework – RESTlet [21].

The proposed approach aligns a process-intensive application with the basic Web principles and promotes many of the desired quality attributes:

• Firstly, a RESTful business process exposes all interesting information as URL-identifiable resources, including process instances, states and tasks. The methods for manipulating these are standardized HTTP verbs which increase process visibility and interoperability.
• Secondly, micro-formats (using XHTML/XML attribute extension mechanisms) are used for dynamically communicating possible next-step tasks (control flows and state transitions), current states and other types of information for an executing process instance. This increases the adaptability and the loose-coupling of process coordination. Rather than having a pre-determined static process, dynamic process fragments are communicated to direct and change processes during execution.
• Thirdly, all communications are stateless. Every request from a client carries all necessary information and does not rely on the server to maintain application state [8] for each client. This promotes the scalability of the system through the feasibility of caching and reduced workload on the server.
• Finally, all resources in RESTful business processes are connected through links. Links could represent control/data flows for possible next-step actions or other information. A client can advance from one state to another by following these links.

The contribution of this paper is in the proposal of a new process application modelling and development paradigm using REST/ROA and an analysis of its benefits for a range of quality attributes.

2. ROA for business Processes

A business process usually has the following concepts [9]: process, case, task, state, and routing. The challenges in developing an ROA for business processes are to: 1) map these concepts into corresponding resources; 2) devise a set of request-response message formats for each type of resources; and 3) designate the semantics for standard HTTP verbs (e.g. GET, PUT) invoked on these resources.

We use a simplified job application example (Figure 1) to illustrate our approach. We choose a state diagram here to emphasize that the details of what and how tasks are performed could be ever-changing and dynamic. The process fragments (as indicated) can change without affecting the state diagram.

Figure 1. Job application process state

Process: A process defines the order of the relevant tasks that should be performed. A process can be considered to be a case factory resource, which is referenced by a URL. A POST request to the case factory results in the creation of a new case (Table 1). If the creation of the job application case is successful, the response message will contain a unique identifier for the new case. The response will also contain possible next-step tasks, links to their corresponding task resources, and the permitted methods.

Table 1. Generate a case

Request
POST /jobapplication HTTP/1.1
Host: http://www.Restfulbp.com
Accept: application/xml

Response
HTTP/1.1 201 CREATED
Date: May 6 2008 13:32:08 GMT
Location: http://www.Restfulbp.com/application/1234
Content-Type: application/xml
The extensible message format is designed using XHTML micro-formats by taking advantage of tags like “rel” to annotate additional semantics. Micro-formats are mechanisms for adding simple semantic mark-up to XHTML/XML-based formats [4].

**Case:** A case is an instance of a business process, which has a unique identifier. A case can be updated or deleted by issuing a PUT or DELETE request. A case, as a resource, has a life cycle with a number of explicit states. A GET request to a case resource will return different responses depending on the current state of the case.

**Task:** A task is a logical unit of work in a business process. It is also modeled as a resource. For example the abstract task “reject by recruiter” could be modeled as a rejection resource with a unique URL: /jobapplication/rejection. An instance of the abstract task in a particular case will also have a unique URL. A task verb is usually noun-ified to make it compliant with a resource-oriented view. Depending on the result of the task initiated or performed, status codes, state changes, and possible next-step actions will be communicated through the return message, synchronously or asynchronously.

**State:** A state is a resource representing the current state of a case. A task may advance a case from one state to another. In workflow-based business process languages, states are often internal shared data rather than exposed external addressable resources. In our approach, the exposure of these states is useful for clients to have better visibility into the running process.

States can be exposed in different ways. As described earlier, they could be described in the response message of a request to a case resource or a task resource. One could also model the abstract state itself as a resource. For example, a GET request to the following abstract state resource URL (/jobapplication/rejected) will return all rejected applications (applications in the rejected state).

**Routing:** A routing determines the path that a particular case could take in a process. Traditionally, these are modeled in a workflow-based diagram using control flows. A process modeled in this way is fairly static and restrictive as all possible routings have to be captured during design time in a complex and interconnected workflow. Such control flows often needlessly over-specify the routing aspect of the process in a cumbersome way.

In our approach, we model routings as specific URL links or general URL templates. For links, as illustrated earlier, possible next-step actions (routing) could be included in the response representation. One can follow them to a new task resource. For URL templates, one can infer new URLs by following the conventions in a URL template.

It is important that possible next-steps tasks be able to be changed at run time to support flexibility. Clients can choose among the possible next-step actions and make a local decision on how to proceed rather than relying on a centralized process model to be told what to do next. In our approach, the whole routing aspect of a process is intentionally under-specified during design-time and can be created and changed during run-time. For each task resource, multiple routing possibilities exist, stored as unlimited number of separate process fragments.

Figure 2 shows the overall architecture of our solution. This new architecture does not prevent the usage of existing services regardless whether they are RESTful services or not, even human intensive services. A resource-oriented “layer” can be added to existing services and capture the additional REST/ROA semantics and micro-formats based messages.
3. Case Study

3.1 The Basic Job application

The job application process of a company is a long running process which requires adaptability to accommodate various scenarios and exceptions. The process also requires interactions among different roles, including recruiters, candidates, interviewers, managers and administrators, forming a complex process with a mixture of human-intensive and automated tasks. This is very typical for real-world business processes, and motivates our selection of the job application example for our case study. We first built the basic job application example to assess the feasibility of the approach including the modeling aspects, the communication formats/mechanisms and the run-time composition of distributed RESTful business processes. We then introduced two change scenarios to the basic example to evaluate process visibility, interoperability and adaptability.

In our RESTful business process, only basic recruiting-related tasks are predefined and exposed through resources. Control flows (i.e. what to do next) are defined as chunks of process fragments for each task. Multiple process fragments can exist for a task, and new fragments can be added if required. These process fragments will be communicated using micro-formats at run time. As long as each party understands the semantics of each task and the micro-formats used, they will be able to cooperate to dynamically choose appropriate control flows.

Figure 3 shows one allowed and implied process in a traditional workflow-based process diagram for illustrative purposes.

First, a candidate creates a job application by POSTing to a well-known URL (Table 1). The server responds with a representation of the job application resource. The format of the response can be negotiated through HTTP’s content negotiation mechanisms. In this case, a XML message is returned. The process has also advanced to the initial state – Created.

Consider the scenario where the candidate finds the wrong name was supplied. To see how information can be updated, a HTTP OPTIONS request is issued to the case URL. The response shown in Table 2 indicates that it supports and currently allows PUT for updating information.

Table 2. method option

| Date: May 6 2008 13:32:08 GMT |
| Location: http://www.Restfulbp.com/application/1234 |
| Allow: GET, PUT |

The candidates now can update their personal information by PUTting an updated application to the URL. Until now the state of the job application has been Created. The submission task is completed by PUTting the job application materials (e.g. resumes and referee letters) to the URL following the instructions in <nextsteps>. Then the server’s response includes updated information about the application (e.g. state and time stamp) and new next-step actions.

Assuming now the process has proceeded to the Reviewed state. An Interviewer can go to the Reviewed state URL (Table 3): /application/reviewed to get a list of candidates that can be interviewed.

Table 3. Check state

```xml
<JobApplications xmlns="urn:restful-business-process">
  <JobApplication>
    <candidate>Sherry</candidate>
    <applicationID>1234</applicationID>
    <state>Reviewed</state>
    <nextsteps><task><role>Interviewer</role><link rel="interview" class="PUT" type="application/xml" href="/job/application/interview/1234"></task><nextsteps>
  </JobApplication>
  <JobApplication>
    <candidate>Harry</candidate>
    <applicationID>2345</applicationID>
    <state>Reviewed</state>
    <nextsteps><task><role>Interviewer</role><link rel="interview" class="PUT" type="application/xml" href="/job/application/interview/1234"></task><nextsteps>
  </JobApplication>
</JobApplications>
```

Figure 3. Job application process
The interviewer can complete this task and collect feedback by correctly populating the following URL templates separately:

```
/application/interview/{applicationID};
/application/feedback/{applicationID};
```

A task could be human-intensive and involve lengthy off-line work, ultimately leading to data being entered into the system through following the URL.

Different roles would be able to complete different tasks. The process will finally reach the Rejected or the Offered state.

### 3.2 Change Scenarios

A major drawback of current process modeling languages is that they require detailed definition of control flows at design time. The only way to accommodate change is to modify the design-time process definitions. On the other hand, RESTful business processes only require tasks and minimal control flows to be defined at design time. Links enable dynamic process coordination at run time. We will use some change scenarios to demonstrate how RESTful business processes handle change.

A company’s recruitment process can vary according to position or circumstances. For example, for technical personnel, a written exam might occur before the interview, to pre-qualify a candidate’s technical knowledge. But for sales positions, a candidate might interview before a written exam, or perhaps have no written exam at all. Additionally, depending on factors such as the number and the quality of the candidates, the company might change the number of interview rounds. Using traditional process modeling language, many process models may need to be created, and it can be hard to switch between them at run time.

Assume this time the company decides to recruit customer service personnel. The customer service recruitment process is similar to the previous process but reverses the order of the written exam task and the interview task, as shown in Figure 3.

The tasks and their associated process fragments link to form a RESTful business process. To accommodate the proposed change, we need only modify the tasks and associated next-step process fragments. From a workflow point of view, this effectively breaks the process chain, reversing the two sub-processes. But rather than manipulating a workflow diagram, these changes are made by modifying `<nextsteps>` tags in response messages, and will be communicated to clients at run time.

However, consider the scenario where after the process is deployed and operational, the company notices that the number of candidates is smaller than expected. The company decides to remove the written exam phase this time. To accommodate this change, the developer need only modify the `<nextsteps>` of the collection feedback resource. This can be done without interrupting the running system, as the client can be instructed to ignore the written exam task in the modified process fragment.

We have considered two examples where the process of customer service recruitment has been changed. The client hasn’t needed to make any changes, assuming it has known how to handle all the atomic tasks.

### 3.3 Evaluation

For evaluation, we re-modeled the job application case in BPEL and WSDL using Axis1.4 [22] and Active BPEL3.1 [14]. We created WSDL interfaces for all tasks and the recruitment BPEL service. The task services are passively called and have no local decisions to make. Comparing the RESTful process implementation with the WSDL/BPEL implementation, we are able to evaluate the feasibility, interoperability and adaptability of our approach.

#### 3.3.1. Feasibility

The job application case demonstrated the feasibility of using a ROA for business processes. The application would have been very different in a traditional SOA.

In SOA, a process advances through different states by maintaining an internal conversation state machine (application state), or be relying on external centralized workflow engines such as a BPEL engine. In ROA for business processes, this is achieved through context-free requests. A process is advanced by communicating link-based process fragments.

In SOA, a service endpoint is coarse-grained and may contain several operations. These operations and related state and process information are not exposed. In ROA for business processes, all process-related concepts are addressable through their own URLs.

In SOA, fine-grained remote operations must be invented. Cooperating systems have to learn each other’s ad-hoc RPC-styled operations. In ROA for processes, we use the uniform interface (standard verbs) that HTTP defines to manipulate all resources.
3.3.2. Process Visibility. A BPEL-based web service orchestration is itself an encapsulated black box web service. This black-box view of web services can be powerful at times. Such services are highly modular and reusable, and any implementation supporting the same interface could be substituted for the original. However, BPEL does not provide good support for interrogating aspects of the process that do not appear in WSDL interfaces [19]. It is difficult to discuss the state of a case based on a BPEL process itself. Instead, we have to depend on web service management tools to examine individual business transactions [20].

In contrast, RESTful business processes permit complete visibility into the internals of a process. The status is part of the general information for every case, task and state. Status is updated automatically when a particular task is completed.

As well as visibility into process content, the visibility of other information provided by HTTP headers can also be important. These can be used for analysis of some non-functional requirements. For example, it would be possible to count the time elapsed for one task by monitoring the Date header.

3.3.3. Process Interoperability. Business process collaboration is being used in increasingly complex systems, and is being developed in many programming paradigms/platforms. Interoperability in such heterogeneous systems is a fundamental problem in the business process domain [13].

Standardization is a key to achieve interoperability. BPEL, as a standard business process execution language, is an increasingly popular language facilitating the orchestration of web services. An earlier case study [16] has argued that BPEL supports better interoperability than traditional methods of business process collaboration, because it has a relatively small set of instructions specifically selected for process definition. However, it still has severe limitations when developing real-world enterprise scale applications [17]. A standard process coordination language only solves the “coordination” part of the problem – specifying interoperation in a standard way. The other part of the interoperability problem is to establish intrinsic interoperability between services – whether existing services can be coordinated at all. This is defined in [18] at different levels.

In our approach, we do not have a separate centralized workflow engine for coordinating process interoperation. Thus, we choose to assess interoperability by following this three-level model.

- **Signature Level**: At this level, component interoperability is based on the signature profile of a component’s operations [18]. In our case study, every resource supports the uniform HTTP methods while fine-grained new operations have to be defined for every WSDL/SOAP-based web service. Syntax differences did arise regarding the parameters of an operation (the representation formats in our case), but this is an issue for uniform or non-uniform operations. Using a uniform interface for verbs with strong HTTP metadata facilities improves signature level interoperability in our case.

- **Protocol Level**: Interoperability at this level means the order in which a component expects its methods to be called, and the order in which it invokes other’s methods [18]. First, every HTTP request in our case is stateless and context-free. This simplifies the execution of the job application protocol. Second, invocation orders are also communicated through <nextsteps> tags. This allows more flexible protocols – (e.g. variants of the job application process in our case study). Both increase interoperability at the protocol level in our case.

- **Semantic Level**: At this level, the requesters and providers have a common understanding of the meaning of the requested services and data [18]. First, we mapped existing HTTP semantics (verbs and status codes) to business process domains, and evaluated this in the job application example. This semantic alignment between HTTP semantics and business process semantics helps indirectly reduce the potential semantic mismatches between two services. Incorrect HTTP verbs have often been used for service/process semantics. We were able to avoid mismatches by following our approach in HTTP verb mapping.

In the job application example, every task could be provided by different business partners or departments, where their systems are running on heterogeneous platforms. RESTful business processes ignore all these difference, exposing an extra layer of semantics with the correct mapping to existing services.

3.3.4. Process Adaptability. Adaptability is often evaluated through change scenarios. We introduced two change scenarios in our case study. It is important to demonstrate that the change scenarios do represent different types of process change.

**Classification of changes**

Our process change classification is based on [12] which classifies the change of a business processes along three dimensions: Time, Scope and Perspective. Time represents design time and run time. Within this paper, we have only cover the time perspective, and introduced two scenarios that cover both design time and run time. Both of the variants change the control
flow of the initial process by exchanging two groups of tasks or by removing a special group of tasks.

For evaluating adaptability, we also re-model the change scenarios in BPEL and WSDL.

**Design time adaptability**

As described earlier, the modification of RESTful business processes involves changing a task resource by introducing/modifying a process fragment variant.

For the WSDL+BPEL approach, more changes have to be made. To exchange the written exam and the interview in the first modification, the wsd1 file for the process does not need to be modified, because the conversational relationships between the recruitment service and interacted services are still the same. All the changes are in the BPEL file. Four <link> tags inside the <links> constraining the order of two tasks, must be changed, the order of the <invoke> of these related tasks must be swapped correspondingly, with the modification of their <target> of a link, <source> of a link and <transactioncondition> for decision. For the second change of removing the written exam task, the aspects mentioned above in BPEL file must be modified, and also the <partnerlinktype> in the process WSDL file must be changed by removing the written exam related partner link types. Effectively, a new BPEL process is defined and needs to be run separately from the old process. By contrast, in RESTful business processes, two process variants can run together and new variants can be added at run-time.

**Run time adaptability**

Business organizations can get more flexibility in their business processes by not only dynamically switching among services which have the same interface (as can be done with dynamic BPEL) but also by changing process topology. For maximum adaptability this should not be a centralized change which has already been captured in a BPEL process. Instead, it should be distributed and local. Our approach only requires a part of the process to be predefined, and uses hypermedia to enable dynamic process composition at run time.

In our case study, we divided our job application process into several process fragments and individual tasks. The whole process can be dynamically generated at run time by connected hyperlinks. The rules for deciding what process fragments to return are stored separately. This has significantly helped us to enable run-time adaptability.

WSDL and BPEL do not support this type of adaptability very well because the different coordinating parts have to understand the protocol beforehand. Under the REST principles, each service gets their process instructions at run-time, rather than just receiving data.

4. Related Work

The current WSDL/SOAP research [15] still assumes the underlying protocols will be used as black-box tunnels. The assumption is valid as it is desirable for Web services to be transportation-protocol independent. On the other hand, if the underlying protocol, such as HTTP, can accommodate not only data intensive applications but also process-intensive applications, then it may be sensible to explicitly use the underlying protocol and its existing infrastructure, such as caching, proxy, and indexing.

Although REST [2,3] and ROA [8] principles are well established, it is still not clear how to apply them to process-intensive systems. CREST [1] is an extension to REST to model computation as a resource. The representation being communicated is mobile code rather than higher-level abstractions such as process fragments. CREST also extends HTTP to allow the transfer of computing expressions. Computing resources in CREST resemble a task resource in RESTful business processes and some CREST principles, such as data and computing bundling, could be applied to RESTful processes.

A WSDL-based service does not have a means to communicate the invocation sequences of all the operations is provides [2]. These invocation sequences are communicated informally, implied in code or embodied in centralized workflow languages such as BPEL or WS-CDL. Not being able to capture process information on the service interface at design time or communicate them at run time significantly reduces the flexibility and visibility of the service. BPEL-like solutions are separated from the services themselves and promote a centralized process coordination strategy. SSDL [10] is a new service description language which captures design time process logic on the service interface. But these are not changed or communicated at run time.

There are also attempts to use workflow languages specifically designed for the Web, such as WebDSL [11]. These do not model process concepts using REST and ROA principles but rather abstract the underlying Web frameworks to form a new Domain Specific Language (DSL). They do not fundamentally change the way process-intensive applications are built.

5. Conclusion and Future Work

RESTful business processes put “Web” principles into process-intensive systems. We have proposed an approach for modelling process-related concepts using REST and ROA principles. We have also developed an infrastructure to support RESTful business
processes. A new modelling tool is being developed to allow annotation of resource-related information on existing process models.

Our approach differs significantly from the existing service design paradigm for process driven systems:

- Unlike existing Web service designs that define dedicated “verbs” for performing various tasks, RESTful business processes use standardized HTTP verbs while representing all concepts (including tasks) as resources with unique identifiers.
- Unlike existing process coordination approaches which rely on static process models and pre-determined control flows, RESTful business processes expose all information as resources and promote context-free, stateless and high visibility communications.

We have conducted initial evaluations of our proposed approach, and have demonstrated its feasibility and improved visibility, interoperability and adaptability. However, there are a number of potential limitations of this work:

- Exposing substantially more process information and interaction through the web demands a strong security model. Such security models need fine-grained access control and adherence to REST principles without sacrificing performance.
- The criteria for deciding what information is to be exposed as resources and what information is to be contained in a representation are still not fully established. The maintenance of consistent information across all locations could also be a substantial challenge.

References