Resource-Oriented Business Process Modeling for Ultra-Large-Scale Systems
Xiwei Xu, Liming Zhu, Yan Liu, Mark Staples
NICTA (National ICT Australia)
School of CSE, University of New South Wales

ABSTRACT
REpresentational State Transfer (REST) [3] and the resource-oriented viewpoint are considered to be the guiding principles behind the WWW ULS ecosystem. RESTful principles are responsible for many of the desirable ULS quality attributes achieved, such as loose-coupling, reliability, data visibility 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 are designed as fine-grained distributed Remote Procedure Calls (RPC), breaking many of the REST principles, and are subsequently harmful to the overall ULS system health. The debate on REST versus SOAP-based “Big” Web services has been raging in the industry. We observe that the main problems lie in two areas: 1) conceptually modeling process-centric business applications using a “resource-oriented” viewpoint promoted by the REST principles; and 2) decentralizing a workflow-based business process (e.g. BPEL) into distributed and dynamic process fragments. In this paper, we propose a solution to these two problems. Our approach aligns process-intensive applications with the basic Web principles and promotes dynamic and distributed process coordination.

1. Introduction
The World Wide Web (WWW) has been widely used for process intensive business applications from the early ad-hoc constructions of Web applications to the recent standardized WSDL/SOAP/BPEL [8] based Web services. Arguably, all these applications effectively form a ULS system [6]. However, the original WWW was not explicitly designed for process-centric business applications. Instead, it was designed for transporting data representations using URL-identifiable resources and hypertexts. Its design follows REpresentational State Transfer (REST) principles [3], which are a set of architectural constraints on top of the basic client-server architecture style. REST principles include stateless and context free requests, standardized and unified interfaces, and URL identifiable and inter-linked resources. Every piece of interesting information is exposed as an “abstract” resource which is identifiable through a URL and may have multiple representations (e.g. HTML/XML/PDF). The components of the ecosystem can manipulate these resources via a uniform interface (standardized HTTP verbs) and exchange the representations. REST is directly reflected in the HTTP 1.1 protocol design and is largely responsible for many of the good characteristics of the ULS Web, such as scalability, loose-coupling and interoperability [3].

However, it is not obvious how a process-centric application fits with a resource-oriented view and follows 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 basic design of the WWW [9]. URLs are not identifiers of any meaningful resources. XML payloads are not used as meaningful resource representations. Process state transfers are not performed through links and explicit resource connectedness.
- The rise of WSDL/SOAP standards does not address the problem as they simply use HTTP as a black-box transport protocol. None of the basic web principles are necessarily being followed or aligned in WSDL/SOAP-based service application designs. Many WS-* new standards are being re-invented to achieve desired ULS characteristics.
- The need of a separate and centralized workflow-based language (such as BPEL [8]) to model the semantics of process coordination while the basic Web has already provided the infrastructures and semantics to achieve many of the goals.

Categories and Subject Descriptors
D.2.10 [Software]: Software Engineering – Design; D.2.11 [Software]: Software Engineering – Software Architectures; D.2.2 [Software]: Software Engineering – Design Tools and Techniques

General Terms
Design, Theory

Keywords
Business Processes, ULS, Service Oriented Architecture, REST, Workflow

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2008 ACM 978-1-60558-026-5/08/05...$5.00.
These problems have been reflected in the constant debate on REST versus SOAP in building ULS systems [5]. Increasingly, developers now simply take advantage of the good characteristics of Web by directly building RESTful Web services [9] instead of using WSDL/SOAP/BPEL based services. Major e-commerce ecosystems (e.g. Amazon) offer web service interfaces in both the RESTful style and the SOAP/WSDL style with a surprisingly higher take-up for the RESTful style [7].

However, many of the successful RESTful web applications are limited to data exposure/manipulation [1, 2]. We think the conceptual gap between REST/resource-oriented architectures and “business processes” is a major barrier for applying RESTful web services to wider problem domains.

In this paper, we propose an initial solution for modeling business process concepts [10, 11] using REST principles and a resource-oriented viewpoint. In particular, we provide a modeling solution as follows:

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

A process coordination engine is implemented to support the concepts mentioned above. This engine is built on the top of an open source REST framework - RESTlet. A tool is also being built to annotate existing business process models (e.g. in BPMN or BEPL) with resource-oriented concepts and dynamic process distribution semantics so a resource-oriented architecture with code skeletons can be generated from an annotated business process.

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

- Firstly, a RESTful business process exposes all interesting information as URL identifiable resources including process instances, states and activities. The methods for manipulating these resources are standardized HTTP verbs rather than self-invented fine-grained RPC methods. This increases process visibility and interoperability.
- Secondly, XHTML/XML-based micro-formats are used for dynamically communicating possible next-step activities, current states and other type of information for an executing process instance. This increases the dynamic and the loose-coupling characteristics of the process coordination. Rather than having a pre-determined static process, a dynamic process fragment is communicated during a process 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” for each client. Representations of a resource capture “resource state” which is exposed and easily accessible. This promotes the scalability of the ULS system through the feasibility of caching and reduced workload on the server.
- Finally, all resources in RESTful business processes are connected to each other through links. That means most of the representations of resources are hypermedia: documents that contain both data and links to other resources. Most links represent control/data flows for possible next-step actions. The client can advanced the process state by following these links.

2. Resource Oriented Business Process Architecture for Ultra-Large-Scale Systems

Business processes usually have the following concepts [10]: process, case, task, state, routing. The challenge is to map these concepts into corresponding resources, devise a set of request-response message formats for each type of resources and designate the semantics for standard HTTP operations (e.g. GET, PUT, POST, DELETE, PATCH) invoked on these resources. We propose the following solution to these challenges and use a job application case (Figure 1) to illustrate.

![Job application life cycle and some process fragments](image)

**Process:** A process defines the order of the relevant tasks that should be done. When a process is instantiated, it becomes a case. A process can be considered as a case factory resource, which is referenced by a URL. A POST request to this process factory results in the creation of a new case. i.e. you can create a new job application by POSTing your request in a pre-agreed format to a business “process” URL:

http://www.example.com/job/application

The response message will contain the unique identifier of the newly created job application. The response may also contain possible next-step actions to be taken (possibly by different roles) and their corresponding links:

```
200 OK
Location: http://www.example.com/job/application?1234
Content-Type: application/xml

<application xmlns="urn:restful-business-process">
...<application xmlns="urn:restful-business-process">
...<link rel="review" href="/job/application/review?1234" type="application/xml">
...<link rel="update" href="/job/application/update?1234" type="application/xml">
</link>
</application>
```

The extensible message format is designed using HTML micro-formats by taking advantage of tags like “rel” to annotate...
additional semantics. Microformats are mechanisms for adding simple semantic mark-ups to XHTML/XML-based formats [4]. A GET request to process resource will return a list of existing cases with corresponding URLs.

**Case:** A case is an instance of a business process. As illustrated earlier, each case has a unique identifier. A case can be further updated or deleted by issuing a PUT or DELETE request. A case, as a resource, has a life cycle through a number of explicit states. A GET request to the same case resource will get different responses depending on the current state of the source. For example, the response format illustrated early will only be returned if the state is “submitted”.

**Task:** A task is a logical unit of work in a business process. It is also modeled as a resource. For example the task “reject application” could be modeled as rejection resource and has its unique URL: http://www.example.com/job/application/rejection. Usually, a task verb is noun-ified to make it compliant with a resource-oriented view. The implementation of a task can be human-intensive or use only computing resources [1]. Depending on the result of the task performed, status codes, state changes, and possible next-step actions will be communicated through the return message.

**State:** A state is a resource representing the current state of a case. A task may move a case from one state to another. A process can be said to contain all of the states which could be reached by any of its tasks. In workflow based business process languages, states are often internal shared data rather than exposed external accessible resources. In our RESTful business process architecture, we expose all states to increase process visibility. For example, a GET request to the following state resource (http://www.example.com/job/application/rejection) will return all rejected applications. All possible next states can also be returned in the response message to enable further analysis.

**Routing:** A routing determines the path that a particular case could take in a process. It can be modelled as links between resources or URL templates. For links, as illustrated earlier, they can be included in the response resource representation. One can follow them to a new resource and manipulate them. For URL templates, one can infer new URLs by following the conventions in a URL template directly. For example, one can infer the URL to issue a review request by following a template like http://www.example.com/job/application/review/?(application_id)

Figure 2 shows the overall architecture of our solution. As business processes often exist in the form of workflow-based models (e.g. BPMN and BPEL), we provide a resource life cycle design tool for annotating a business processes with states and resource-related information. One such example for a procurement process is shown Figure 3. A state indicating a state change can be annotated on a control flow following the execution of a task. Such a tool can provide some basic analysis features such as identifying obvious conflicts. Through a model driven development paradigm, a resource-oriented architecture and code skeletons can be generated from the annotated model. A new programming model for supporting a resource-oriented view of business process concepts has been developed. The programming model will have associated design time APIs and runtime engines for supporting the programming and running of a RESTful business process. The server and client side will communicate through pre-agreed but extensible micro-formats.

This new architecture does not prevent the usage of existing services regardless whether they are RESTful services or not, even human intensive services.

![Figure 2. Architecture of RESTful Business Process](image)

![Figure 3. Procurement process annotated by states](image)

### 3. Discussion and Conclusion

We have applied part of our method in an industry-wide data dissemination standardization project, which has demonstrated the feasibility of the approach and has achieved some positive initial results [12, 13]. For example, the resource-oriented process allows more flexibility in distributing different process fragments (permisssible next-step tasks) to different consumers of data. However, the processes involved in data dissemination are not highly complex – we are currently applying the method to other more control-flow intensive processes.

The solution we have proposed differs significantly from existing service design paradigm for ULS 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 which relies on a static process models and pre-determined control flows, RESTful business processes use micro-formats to dynamically communicate process “fragments” that define all possible next-step actions during execution.

• Unlike existing business process engines which maintain internal state and exposing limited information about a business process, RESTful business processes expose all information as resources and promote context-free, stateless and high visibility communications.

All these design decisions strictly follow the RESTful principles which have made the WWW ULS ecosystem successful in terms of data transportation. We bring these proven benefits to process-intensive applications and harmonize the two types (process and data) of applications by using the same resource-oriented view.

However, there are a number of potential limitations of this solution.

• Exposing substantially more process information and interaction through the web demands a strong security model. Such security models need fine-grained access and control capabilities and adherence to REST principles without significantly sacrificing performance and scalability.

• The criteria for deciding what information is to be exposed as one or more resources and what information is to be exclusively contained in a representation of a resource is still not clear and may often be an application-specific decision. The maintenance of consistent information across all places could be a substantial challenge.

We are currently implementing the resource life cycle annotation tool and improving our runtime engine.

4. Acknowledgements

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.

5. References


