An Architecting Method for Distributed Process-Intensive Systems

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

Abstract

This paper introduces an architecting method for distributed process-intensive systems. Traditional methods (e.g. object-orientation, structured analysis or component/service-based designs) decompose a process-intensive system into entities with attached domain-specific operations (process constituents). This results in fine-grained Remote Procedure Calls in distributed systems which are often detrimental to quality attributes such as performance, loose-coupling, adaptability and interoperability. Our method tailors the REpresentational State Transfer (REST) principles used for hypermedia data transfer to process-intensive systems by making process constituents into resources, and attaching a set of standard operations. Distributed processes interoperate by adhering to these operations and exchanging process information. In our method, process information exchange contains not only typical meta-information about a process, but also process fragments that indicate possible next-steps and interconnectedness for process coordination purposes. We have implemented our method in a Web environment and conducted a case study providing initial validation of its benefits.

1. Introduction

Many software architecture design methods are action-oriented in nature, since the computer is basically an action-oriented device [11]. A typical action-oriented design strategy is functional decomposition, where a process is realized by decomposition into a collection of domain-specific operations/procedures. Many business process modeling approaches, such as BPMN [3] and BPEL [7] follow this approach. Alternatively, when architecting a process in the OO paradigm, architects identify and decompose entities, and attach operations to them. Most of these operations are fine-grained and domain specific. These operations are invoked in a particular sequence to realize a process. This invocation sequence (process) can either be implicit in an implementation or be explicitly modeled as a meta-operation attached to an entity. The same core principles are used in component-based architectures and Service-Oriented Architectures (SOA) albeit with advances in binary component reuse, interfaces definition and interoperable protocols. Due to the exposure of reusable and single-purpose functions as stateless services, there is also a tendency to regress to a pure action-oriented paradigm in SOA.

The WWW embodies new architecture methods for distributed data transfer. Roy Fielding proposed the REpresentational State Transfer (REST) principles [1] which describe the ideas realized in HTTP1.1. The REST principles add several constraints on top of the classic client/server architecture styles. These principles include: stateless and context-free requests, standardized and unified interfaces, and URI identifiable and inter-linked resources. All interesting information is exposed as abstract resources in a Resource-Oriented Architecture (ROA) [4]. Resources are identified through URIs and may have multiple representations (e.g. HTML/XML/PDF). A system can manipulate resources via a uniform interface (i.e. standardized HTTP operations) and exchange representations of resources. The REST principles are largely responsible for many of the good characteristics of the Web, such as adaptability, data visibility and interoperability [14].
However, REST was designed for data transfer and "resources" are essentially nouns representing abstract data. Latest attempts [17-18] in applying REST to process-intensive system are limited to extending the existing workflow languages to support integration with RESTful services. There is no guidance on how a process-intensive system can be architected using the REST principles. Nonetheless the WWW has increasingly been used for process-intensive business applications, and this has created confusions, architecture mismatches and limitations [11].

In our earlier paper [11], we proposed some initial ideas on how to use existing WWW infrastructure to build process-intensive systems. In this paper, we generalize those specific techniques into a new architecting method. Our method tailors and enhances the REST principles towards process-intensive systems. Our method is different from past methods in the following aspects:

- Unlike past methods that promote fine-grained domain specific operations, our method uses a limited set of standard operations on "nounified" domain-specific actions.
- Rather than using a separate process coordinator (e.g. a BPEL engine), process coordination information sits behind an operation and is exchanged dynamically among participants using microformat-based process fragments.
- Rather than exposing only operations and data, additional process information (e.g. invocation sequence) can also be exposed at design time and exchanged at run time.

This new method promotes loose coupling, adaptability and interoperability for distributed process-intensive systems. A natural environment to support the method is the WWW. However, the architecting method is general and could be supported by other types of technologies.

2. Design method

Although the method is not limited to WWW/HTTP, we will use WWW/HTTP examples of a travel booking process for illustrative purposes. The process requires interactions among several parties, forming a complex process with a mixture of human-intensive and automated tasks.

Our method is currently supported by an application framework, designed for applying the method in the WWW environment. The current application framework extends an open source web framework (RESTlet [5]) to supply additional WWW-specific infrastructure support.

Step 1: Define tasks, roles and distributed process coordination

A central goal of our method is to establish communication and coordination mechanisms among the participants of a business process from a peer-to-peer and distributed point of view (rather than with centralized coordination). In our method, a business process could initially be modeled using any process modeling notation, e.g., BPMN [3] (Figure 1) or UML activity diagram. The purpose of this step is to identify all the tasks at different levels of abstraction, parties (roles) and loose (perhaps underspecified) peer-to-peer connections of these tasks. Unlike traditional use of a process specification language, the goal is not to define precise and complete process control flow but to borrow the notation for task structure identification. This step contains the following three sub-steps:

1.1. Identify all tasks required to realize a process. It is helpful to differentiate tasks performed by humans from fully automated ones.

1.2. Identify the roles/parties for each task.
1.3. Organize the task structure. Some structures capture loose sequencing relationships among tasks. Other structures capture the hierarchy of tasks at different levels of abstraction and details. For instance, a booking task could be divided into two sub-tasks: book flight, and book hotel.

The whole process definition may not look like a control flow but instead may have a tree structure with loosely connected process fragments. Explicit conditional routing (e.g. diamonds in BPMN and UML activity diagram) is possible, but it is also possible to attach such information to a task as process fragment information or rules to be exchanged among parties. A task may have different routing information attached to support a variety of situations as needed. One good language for representing a process in a tree structure with flexible channels among them is Little-JIL [2].

**Step 2: Identify operations and sub-processes and make them into entities in a design.**

The hierarchical tasks modeled in Step 1 can be realized as entities by a variety of design methods. For example, in the OO paradigm, a hierarchy of classes can be used. In the WWW environment, a set of resources identifiable through URIs can be used.

Granularity is the primary concern here. The size and complexity of a task may vary according to different needs. Reuse, the ability to evolve, and integration effort [6] all play a role in selecting the right level of granularity. For example, it would be difficult to reuse a large task, and the integration of many fine-grained tasks may move complexity into the integration and testing phase. In a distributed system, round trip delays between many smaller tasks may also have performance consequences. However, infrastructures such as WWW provide various mechanisms to alleviate the situation. The bottom line is that there is no optimal size and complexity of a task. There are some related entities in our method:

- **Process/Sub-process** – can be regarded as a process instance factory.
- **Case** – an instance of a business process, generated once the process is instantiated.
- **Task** – a logical unit of work in a business process and modeled as fine-grained domain specific operations in the traditional method. In our method, every task is an independent noun entity.
- **State** – represents the current status of one case. An implementation of a task may advance a case from one state to another. Exposure of states is useful for visibility into the running process.

To better illustrate this step, we now use the WWW environment as an example. Each entity can be regarded as a URI-identified resource [4]. For example in the travel booking process, the URI of a process entity can be designated as a process resource identifiable through www.restfulbp.com/travelbooking. The states can be annotated on Figure 1, which helps the participant to advance a process instance.

**Step 3: Define a limited set of uniform methods to manipulate the entities**

Our preliminary set of methods resemble the HTTP standard methods but with important differences. We outline our adaptation of these HTTP verbs in Table 1. The standard operations could be further extended, but these extensions should be limited, to avoid including ad hoc RPC-styled operations as used in traditional approaches. For example, a designer might choose to support a DELETE operation to allow an administrator to dynamically remove process elements.

**Table 1. New meaning of the HTTP methods.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Process</th>
<th>Case</th>
<th>Task</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Access process spec.</td>
<td>Access instance status</td>
<td>Access task spec.</td>
<td>State-specific cases</td>
</tr>
<tr>
<td>PUT</td>
<td>N</td>
<td>T</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>POST</td>
<td>Create a new case</td>
<td>N</td>
<td>Implement a task</td>
<td>N</td>
</tr>
<tr>
<td>DELETE</td>
<td>N</td>
<td>Cancel a case</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Step 4: Define the output of all methods**

When a method on an entity is called, information must be returned to facilitate process coordination and indicate status. This information may contain meta information or status information about an entity (tasks/process/case/state) and more importantly, process coordination information such as possible next-steps and connectedness among relevant tasks.

A case entity has two kinds of outputs. One represents the current status of a case, which is often returned in GET method to any counter-party of the business process. This ensures the visibility of a running process to all parties involved for more informed local decision making. The other one represents all possible next-steps for a counter-party to advance the case. This information can only be returned to the relevant counter-parties with...
appropriate authentication so that they can advance the case by calling the POST method on the URI provided in for each possible next step.

**Step 5: Define the communication formats**

In step 5, we define the formats to be used to communication information about processes. In the WWW, microformats are a popular and lightweight approach where information intended for end-users is embedded in existing XTHML tags with additional semantics. In our framework, we use microformat-based messages to communicate routing information (next steps, task connectedness and sequence/parallel semantics) at run-time. Although the URI of each task is recommended to be as descriptive as possible, microformats are still useful to supply additional semantics for tasks.

Since we use microformat-based messages to dynamically communicate routing information that defines all possible next-step actions during execution, we need to design some extra constructs to represent various workflow patterns [10]. In our method, we only need to describe the relationship of two related steps, rather than the whole control flow as traditional centralized modeling languages tend to do. Compared to languages like BPEL, the construction of our communication microformat is simpler. One basic construct can be used to represent several workflow patterns. Our framework can support all the 20 workflow patterns [10] although some patterns need some workarounds.

**4. Conclusion and future work**

In this paper, we proposed a software design method for business process systems. Our method tailors the REpresentational State Transfer (REST) principles used for hypermedia data transfer to process-intensive systems by making process constituents into resources and attaching a limited set of standard operations to them. Distributed processes interoperate by exchanging process information which contains not only the typical meta-information about a process but also process fragments that indicate possible next-steps and interconnectedness. We implemented our method in a Web environment by providing a framework to support RESTful business processes execution. We have conducted initial evaluations of our proposed approach by comparing with traditional. From the quality perceptive, systems using our architecting methods promote several quality attributes including process visibility, process interoperability, and run-time adaptability.

**8. Acknowledgments**

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

**9. References**