Reengineering Legacy Systems with RESTful Web Service*

Yan Liu[1], Qingling Wang[1], Mingguang Zhuang[1] and Yunyun Zhu[2]
Tongji University, Shanghai 200092, China[1]
Department of Information Technology, Uppsala University[2]
{givemeareason, wqingling, zhuangmg}@gmail.com[1], yuzh9454@student.uu.se[2]

Abstract

Most of the SOA (Service Oriented Architecture) applications are not brand new and usually evolved from legacy systems. Legacy systems carry out the enterprise's most crucial business information together with business processes and many organizations have leveraged the value of their legacy systems by exposing parts of it as services. Most of current web services are SOAP-RPC style services. The evolution of the Web 2.0 phenomenon has led to the increased adoption of the RESTful services paradigm and reengineering legacy system to SOA with RESTful web services is not only for reusing but will bring other benefits due to its special features. In this paper, the key issues for reengineering legacy systems with RESTful web services are discussed. A common process for reengineering legacy systems to REST-style is proposed. The candidate web services are identified by legacy systems comprehension at first. Then blend services are generated based on relationships between entities and constraint rules specified. The generated URIs are refined and split carefully to represent the RESTful web services.

Keywords

Legacy system, Reengineering, RESTful web service

1. Introduction

The legacy systems or existing working systems of enterprise usually contain the mixture of different techniques and protocols which is hard to maintain and update. With the development of service-oriented technology [2], nowadays it's possible to wrap the existing functionalities with web services and build ready-to-use services which can be mash-up to construct new system without further modification. The software architect and the program manager should consider the services capabilities of existing systems and match them with end users' new requirements in order to build a new service system for the organization semi-automatically with low efforts.

In our work, the main participants in a service-oriented application are classified as follows. *Service Provider (SP)* who is the key information owners and usually provides ready-to-use services with the support of exclusive informative entities. *Service Integrator (SI)* who is the solution provider and will take advantage of some existing software frameworks or utilities and deliver suitable service platform to end users. With the domain knowledge deposited, the SI can also design their service based on exclusive knowledge or domain related informative entities. *Service Users (SU)* who is the end consumer of the services includes end users and the leaf applications in the services chain.

Legacy systems for an organization usually carry the crucial informative entities which can be used to provide exclusive services and business knowledge. All of above three kinds of participants in a service chain can reengineer their legacy or existing systems with web services to reuse the existing features and generate new features with very low development efforts. Effective reengineering methodologies and frameworks are becoming increasingly important, which can wrap existing functionalities to provide ready-to-use services without additional technology or platform constraints .

The key motivations to reengineer legacy systems with web services can be summarized as follows.

- *Functionality reuse*, the reusability is based on services wrapping instead of rewriting existing code which will bring down the total efforts.
- *Services capabilities mining,* the new services-oriented system construction will be driven by end users' requirements together with services capabilities owned by the enterprise in consideration. This can avoid redundant construction and prefactor services by mining legacy systems even without direct demands from end users.
- *Future mash-up*, under the revolution of Web 2.0, mash-ups represent a promising new development approach to building composite applications and enable easier, faster integration of existing web applications.
- *System maintain and update,* the comprehension methodologies for reengineering legacy systems and wrapping strategies can also be applied for system maintenance and updating.

The intended contributions of this paper are:

1. Introducing the idea to classify main participants for a

* This project is supported by Tongji Young Talent Program under grant number 2100219007.

0730-3157/08 $25.00 © 2008 IEEE
DOI 10.1109/COMPSAC.2008.89

service-oriented system as 3 different roles,

2. Comparing the SOAP-RPC style and RESTful web services briefly and select RESTful web services techniques to reengineer legacy systems to services-oriented architecture,
3. Introducing the key problems to be solved in order to reengineer legacy systems with RESTful web services, and
4. Presenting the reengineering approach, including the candidate services identification based on informative entities, the blend services generation based on existing relationships, the final URIs split and standard operations selection.

This paper is organized as follows. In Section 2, the related two different services-oriented architectures are compared and the advantages of RESTful web services are introduced. In section 3, we present the key issues of reengineering legacy system to RESTful web services. Section 4 proposes our approach for reengineering the legacy system with RESTful web services. The conclusion and future work is in section 5.

2. Background and Related Technologies

Currently, there are two types of web services building approach: the SOAP-RPC style, which implement services with tightly coupled operations and design the interactions between distributed parties based on business process modeling, and the RESTful style, which handle the web services and common web invocation in the same architecture, designs services by identifying resources types and implement remote invocation with standard HTTP operations.

As a standard recommended by W3C, SOAP-RPC web services are widely adopted by SPs in the last several years, which use SOAP for communication between applications and Web Service Description Language (WSDL) for services description. But SOAP-based Web Services are usually designed for remote application invocation. SOAP based on HTTP and XML, but which doesn't use the same invocation style with HTTP. It's difficult to design web services and web application in the same architecture.

Representational State Transfer (REST) is an approach for getting information from a Web site by reading a designated Web page that contains an XML (Extensible Markup Language) file that describes and includes the desired content [1]. Since REST has been introduced by the dissertation [10] of Fielding, it steps into a new picture of designing and implementing Web Services and is proposed as an alternative view to loosely-coupled services in general.

RESTful service has a universal API which offers a formal and stable interface. It explicitly avoids statefulness no matter the service is static or changeable and does not provide the service interface to strong typed messages which makes it very suitable for the clients and developers using scripting language to combine the new services and the local interfaces together.

In our work, RESTful web services are used with the following issues in consideration:

- Design and implement web services and web applications in the same architecture. RESTful web services can be treated as a common web application. Both the new features of the web application and services can be designed in a same vision which can simplify the analysis and design process.
- Integrate with RIA (Rich Internet Application) conveniently. The client side for a RESTful web services can be a simple web browser, a rich internet client or a complicated application. With development of web2.0 and RIA technologies, RESTful web services have more advantages compared with SOAP-RPC style web services which usually take a complicate application as the service client.
- Possibility for effective semantic services composition. It's difficult for SOAP-RPC style service to implement effective web services mash-up because of the diversity of operations and the process-driven design methodologies. For RESTful style, web services are designed based on resources and there are more possibilities to mash-up web services with unique resource representation in the same domain and the small set of standard operations in HTTP will also make it simple to define and select the operations.

3. Key issues of reengineering legacy system with RESTful web services

The RESTful web services provides many promising advantages, it still has some weakness. The following problems should be considered and design carefully during reengineering legacy systems.

Pb1. Identifying and locating resources appropriately – resources are not data, it is the service designer's idea of splitting up data into "a list of RESTful thing". It's a challenge for service designers to find out the data sets and split up them to proper resources. In our approach, the resources can be identified automatically based on the static structure, existing relationships and existing knowledge and rules in legacy systems and then the designer can refine the resources with experiences or REST design patterns.

Pb2. Design URIs with higher interoperability and scalability – how to confirm the scoping information [4] ("why should server operates on this data instead of that data?" [3]) kept in the URIs. In our work, constraints on resources and rules extracted from legacy systems can be represented by special part in the URIs.

Pb3. The operations assignment – standard operations are used in RESTful web services. Once the resources are indentified and URIs are designed, the system should assign standard operations to services semi-automatically with the comprehension of existing methods in legacy systems.

Pb4. Considering future mapping – there are no widely adopted standards for RESTful web services. Diversified service designs will come forth for this. The hierarchy of RESTful web services should be designed with future mapping with other services provider in consideration.

Pb5. Possible semantic description – the semantic description for RESTful web services now is very informal.

At the same time, suitable legacy system comprehension methodologies should also be developed. We use informative entities driven methodology because most RESTful web services are based on content-oriented resources.

The proposed reengineering procedure is as follows. The detailed approach will be discussed in section 4.

1. Extract candidate resources with informative entities identification. The key informative entities can be found with reverse engineering and model layer analysis. For SPs, the candidate resources can also be identified by domain experts. For SIs, advanced candidate resources can be found out from the local knowledge repository.
2. Create blend services with static relationships analysis in legacy systems. Different kind of relationships can be mapped to different part of resources. Important association relationships can be used to generate blend resources. Effective blend resources design can help the SIs establish a simple services hierarchy for future extensions.
3. Design URIs to represent services. The RESTful web services will be determined by the exposed resources, scope, constraint on the resources and standard operations. The URIs will carry the resource representation together with scope and even some business rules can be exposed via the URIs. We design a set of mapping strategies between indentified resource types and URIs which can generate URIs automatically.
4. Analyze existing operations in legacy systems, refine URIs and then assign standard operations. With legacy system operations clustering, typical and related informative entities can be determined to help the designer assign standard operations to each resource.
5. Design Service representations. Since there is no standard representation for RESTful services, the detailed services representation will be verified and refined in this step manually.
6. Wrap existing features with RESTful web services. Once the resources designed, URIs specified and the standard operations assigned. It's very easy to wrap existing features with middle interfaces to generate the RESTful web services automatically.

4. Proposed Approach

RESTful web services yields many benefits and alleviate many problems associated with 'traditional web services' (web services through such things as SOAP, and the older XML-RPC protocols [9]). In this section, we will discuss a common process of reengineering legacy system to RESTful web services by analyzing the source codes.

The web system is designed for different purpose with different structure and enhancement. With the development of software engineering and design patterns, most information systems are built under the MVC triad of classes. In our approach, analysis the source codes of legacy system is started from the MVC extraction.

Figure1 shows the common reengineering process. It addresses both behavioral and architectural aspects of legacy system reuse. We will discuss details of reengineering steps later.

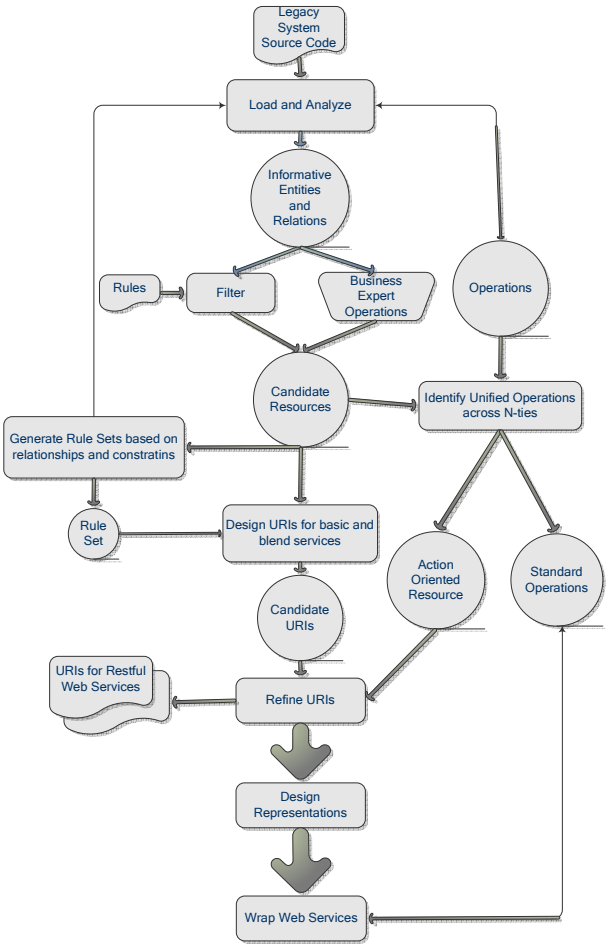


Figure 1.The Common Process

4.1 Identify Candidate Web Services

In the extraction of candidate web service, we land a novel idea of representing service capabilities based on statistical study on WSDLs. We conduct a statistical study on more than 400 WSDL documents collected from XMethods.net, Amazon and Google. According to the analysis result on these WSDLs, service capabilities can be represented by informative entities and standard actions which related to each informative entity separately [4].

Under this conclusion, the informative entities should be exposed as resources. In this step, the only task is to identify and collect all informative entities.

From MVC view, each informative entity corresponds to one independent entity in Model layer. But analysis on model layer is not enough since it may not contain all potential resources. Some other informative entities extraction sources are used as shown below.

- ER Diagram – usually the informative entities are represented as tables in ER diagram.
- UML Diagrams – class diagram, sequence diagram, state diagram [5] and other diagrams often include informative entities.
- Requirements – informative entities represent service capabilities, so we can find them from the requirements inversely.
- Documents – technical documents written during development can help to find the informative entities.
- Experts – skillful architect can find the informative entities directly.

Once get the extraction results, we can filter them to identify resources by some predefined rules or map them to candidate resources by business experts. Each service has several standard operations and this source code analysis process can help to dig out useful information about the operations at the same time.

4.2 Generate Blend Web Services based on Existing Relationships

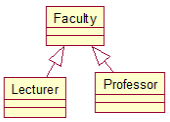
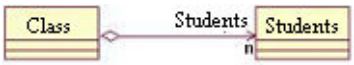
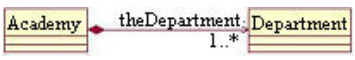
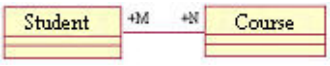
The informative entities are data sets which represent potential resources. Some of the services are determined by more than one informative entity. We use the following relationships to generate blend web services or add additional information in the URIs.

- Generalization Relationship
- Aggregation Relationship
- Composition Relationship
- Association Relationship
- Parallelism Relationship

The resources are classified by the relationships between them as the informative entities are cooperating with each other under these relationships. We can identify resource types and design URIs under predefined rules. These rules are used to describe relationships and specify the scope of the recognized information.

Once we get the relation graph with informative entities extracting, we can design URIs for different resources under it. In a resource-oriented service, the URI contains all the target information and will determine the dataset for design. First we assume there is a root URI: http://myschool.example.com. Table1 shows how to generate blend web services vividly by following given mapping rules.

Table 1. Resource Relation Types and URIs' Mapping Rules

Relation Type	Sketch Map	Mapping Rule	URI demonstration
Generalization		"/" represents generalization	http://myschool.example.com/faculty, http://myschool.example.com/faculty/professor
Aggregation		(1) ":" represents aggregation (2) If the part entity can exist independently without the whole entity in some context, it should have its own URI.	(1) http://myschool.example.com/class http://myschool.example.com/class/student (2) http://myschool.example.com/student
Composition		":" represents composition	http://myschool.example.com/academy, http://myschool.example.com/academy:department
Association		"-" represents association	http://myschool.example.com/student-course

The generalization relationship represents a hierarchy and each entity in this relation has a piece of scoping information; aggregation represents typical whole/part relationship; composition is exactly same as aggregation except that the 'part' is controlled by the 'whole' and cannot exist independently; association represents an entity associates with another.

There is another condition, namely the parallelism relationship, need to be considered. For parallelism, there is no direct relation between two entities, but they could be connected via a third party (i.e. the longitude and latitude of a map). To this kind of relationship, we combine the two entities on the same level with comma in a hierarchy (i.e. http://mymap.examle.com/place/longitude, latitude).

Sometimes the path variables mentioned above are not quite appropriate. We can use query variables to name some resources. For example, Google search URI: http://www.google.com/search?q=REST. Here the scoping information [{q=REST}] tells the browser how to track this URI.

4.3 Refine URIs and Define Standard Operations

In order to build powerful new service-oriented system upon the legacy systems, we need to dig out the inner business logics of the legacy system as well as the existing functions.

In section 4.1 and 4.2 we have figured out how to determine the potential resources and the rules. Then we will analyze the source code based on the previous results to refine the candidate resources, URIs and get all useful operations to do the next wrapping step. The detail processing flow is shown in figure2.

The pseudo code for the operation analysis process is given in Figure 3. The variables are defined as follows: 'Ri' refers to the set of resources. 'r' corresponds to a specific resource in 'Ri'. 'Oi' refers to the set of operation, lower case 'o' corresponds to a specific operation in 'Oi'. 'RO' refers to the operations set related with some resource. 'RSO' refers to the standard operations set related with some resource. 'NRNSO' refers to set of other operations not in RO and RSO. Lower case 'rn' corresponds to the action-oriented resource. 'RNi' refers to the new resource set. The computational complexity of this algorithm is $O(n^2)$.

Once found out all the resources and unified operations of a legacy system and then designed URI for each resource, we can begin to construct the services-oriented system with RESTful web services. The final step is to design concrete representations of services and then to build RESTful Web Service from legacy system automatically.

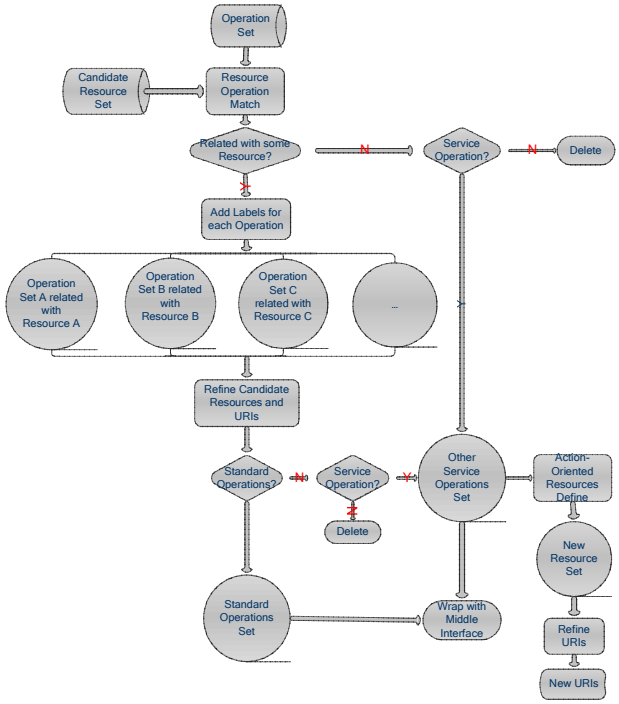


Figure 2. Flow Diagram of Operations Analysis Process

```
1   begin
2       for each r in Ri do
3           for o in Oi from 1 to k do
4               if (related_with(r, o) == true)
5                   add_labels_for_functions (o); /* add some labels to mark the operation, i.e. the
6   relation between the informative entity and operation (return value, parameter and so on); i.e. the
7   precondition operations and so on.*/
8                   RO.add (o)
9               else if (is_service_operation (o) == true) NRNSO.add (o)
10              endif
11      refine_candidate_resources (Ri, RO)
12      for each o in RO do
13          if (standard_operations (o) == true)
14              RSO.add (o)
15          else if (is_service_operation (o) == true)
16              NRNSO.add (o)
17          endif
18      for each o in NRNSO do
19          rn = create_new_resource (o) // action oriented resource
20          RNi.add (rn)
21      for r in Ri from 1 to m do
22          for rn in RNi from 1 to n do
23              relation = find_relation(r, rn)
24              URI = refine_URI (relation)
25  end
```

Figure 3. Pseudocode of Operations Extraction

http://www.tongji.edu.cn/library
http://www.tongji.edu.cn/library:bookroom1

Figure 4. A plain-text representation

4.4 Design representations and Build RESTful Web Services

We need to represent each part of resource in a fixed format. The representation is to convey the state of the resource and either plain-text, JSON, XML or XHTML representation is applicable. Figure 4 shows an example of RESTful Web Service representation in plain-text.

So far we've finished the decomposition and labeling work, next step is to wrap all the results into REST. Finally, keep the original system. Based on the existing legacy application, we provide interfaces to expose individual services inside of the legacy system.

5. Conclusion and future work

In this paper, we proposed a common process to analyze legacy and existing systems in order to build RESTful web services semi-automatically.

The key problem for RESTful web services design is to split the resource type with suitable granularity. In our approach, the candidate services can be indentified from the key informative entities in the legacy systems. The informative entities represent the real services capabilities for the systems and most of them can be found out with legacy system comprehension. The relationships between informative entities are used to generate blend services automatically which can help the RESTful web services designer to define resource types based on a common vision of the systems. The realization of the final URIs is also illustrated in our work. Furthermore, the Tripartite Matching Mechanism which includes user's new requirements, left functionalities of legacy system and RESTful web services can be added in.

The approach present in this paper is more helpful for the data-driven or entity-driven system. During the reengineering, we consider the potential web service on the granularity of informative entity but neglect the smaller granularity.

There are several issues yet to be investigated, one of which is the dependency relationship representation in the URIs. Generally speaking, one informative entity may use many other services which are treated as resources in RESTful web services. In order to improve the semantic mash-up of services, how to specify this dependency relationship in the URIs still need to be studied in the future. The second issue is regarding blend services. In our approach, blend services can be found out based on the static association relationship between informative entities. However, some services need more than two connected informative entities and the services are usually provided across several different organizations with a set of informative entities. How to specify this resource type will be investigated with further detail. The third issue is about ontology. Since the classes, operations and web application configuration parameters in legacy systems are usually defined by different organizations, we are investigating the possibility to map the indentified unified operations in the legacy systems using domain ontology and generate the URIs based on domain ontology.

6. References

[1] Roy T. Fielding, Richard N. Taylor, "Principled design of the modern Web architecture", ACM Transactions on Internet Technology (TOIT), v 2, n 2, May, 2002.

[2] Thomas Erl, "Service-Oriented Architecture", Prentice Hall PTR, 2004.

[3] Leonard Richardson, Sam Ruby and David Heinemeier Hansson, "RESTful Web Services", O'Reilly Media, Inc, 2007.

[4] Yan Liu, Mingguang Zhuang, Qingling Wang and Qi Lu, "A novel approach for service capabilities representation based on statistical study on WSDL", International Conference on Internet and Web Applications and Services(ICIW 2008), 2008.

[5] Grady Booch, James Rumbaugh and Ivar Jacobson, "Unified Modeling Language User Guide, 2nd Edition", Addison-Wesley Professional, 2005.

[6] Harry M. Sneed AneCon GmbH, "Integrating legacy software into a Service oriented Architecture", Proceedings of the Conference on Software Maintenance and Reengineering, (CSMR 2006), 2006.

[7] Xiaofeng Wang, Shawn X.K. Hu, Enamul Haq, and Harry Garton, "Integrating Legacy Systems within The Service-oriented Architecture", Power Engineering Society General Meeting, IEEE, 2007.

[8] Semih Cetin, N. Ilker Altintas, Halit Oguztuzun, Ali H. Dogru, Ozgur Tufekci and Selma Suloglu. Legacy "Migration to Service-Oriented Computing with Mashups", International Conference on Software Engineering Advances, (ICSEA 2007), 2007.

[9] Fielding, R. T, "Architectural Styles and the Design of Network-based Software Architectures (PhD Thesis) ", UC Irvine, Information and Computer Science, 2000.

[10] Maseud Rahgozar and Farhad Oroumchian, "An effective strategy for legacy systems evolution", Journal of Software Maintenance & Evolution. Issue 5, 2003.

[11] Grace Lewis, Edwin Morris and Dennis Smith, "Analyzing the Reuse Potential of Migrating Legacy Components to a Service-Oriented Architecture", Software Maintenance and Reengineering, (CSMR 2006), 2006.

[12] Daniel Szepielak, "REST-based Service Oriented Architecture for Dynamically Integrated Information Systems", PhD Symposium at ICSOC 2006. 2006.