第35卷 第19期 计 算 机 工 程 2009年10月
Vol.35 No.19 Computer Engineering October 2009
·软件技术与数据库· 文章编号：1000—3428(2009)19—0033—03 文献标识码：A 中图分类号：TP311

基于SCA的遗留系统移植研究与实现

曹 迪，陈 平，鲍 亮，胡圣明
(西安电子科技大学软件工程研究所，西安 710071)

摘 要：针对传统遗留系统在面向服务计算移植过程中存在的重复性和多样性问题，提出一种基于服务组件框架(SCA)的遗留系统移植方案，将遗留系统移植到SCA组件系统中，通过python扩展生成器生成可以被该脚本调用的模块，在硬件平台上进行模拟，实验结果表明，该方案是可行的，能够避免因面向服务实现形式的差异而造成的重复移植。

关键词：面向服务架构；遗留系统；服务组件框架

Research and Implementation of Legacy System Migration Based on SCA

CAO Di, CHEN Ping, BAO Liang, HU Sheng-ming
(Software Engineering Institute, Xidian University, Xi'an 710071)

【Abstract】Aiming at the problems of repeatability and diversity in process of service-oriented computing migrations in traditional legacy systems, a legacy system migration scheme based on Service Component Architecture(SCA) is proposed, which transplants the legacy system into SCA system. The modules which can be scheduled by script are generated by using python extension generators. It is simulated on hardware platform, and the results show this scheme is feasible, and can avoid the repetition migration caused by the differences of service-oriented implementation shapes.

【Key words】Service-Oriented Architecture(SOA); legacy system; Service Component Architecture(SCA)

1 概述

随着面向服务架构(Service-Oriented Architecture, SOA)的广泛应用，大量企业级应用即将采用SOA技术实现，如何有效重用遗留系统成为人们关注的热点，之前SOA的各种实现形式为从遗留系统到SOA的移植提供了多种不同选择，基于Web services的移植是种常用的选择，但如果企业级的SOA应用中存在其他形式的服务时，基于Web services的移植系统就需要再次为新的SOA系统进行移植，这将会导致大量的重复移植工作，从某种程度上讲，这是由于SOA服务形式不统一造成的[1]。

服务组件框架(Service Component Architecture, SCA)提供一套符合SOA思想的规范，以SCA构建企业级的服务应用系统能够实现业务逻辑与底层基础架构、服务质量和传输的分离。根据以上SOA的发展现状及SCA的特点和优势，本文提出一套将现有遗留系统移植到SCA组件的方案。

2 遗留系统移植方案分析与设计

2.1 迁移平台分析

由于系统中组件的基本形式主要为动态链接库(.DLL)和应用程序(.EXE)，因此移植方案基于以上2种组件形式开展。

从现有平台的DLL和EXE程序移植到基于SCA的组件，首先要了解SCA系统的部署环境，如图1所示，SCA的运行环境可以视作一个SCA Domain, Component是SCA集合中基本的业务功能组成元素，而Composite则作为SCA集合中的逻辑单元[2]。Component包含services,properties,reference以及实现的概念。从本质而言，services,references体现了具体接口与通信协议的分离，在某种意义它是一种新型的接口，而implementation则体现了接口与实现模块的分离。

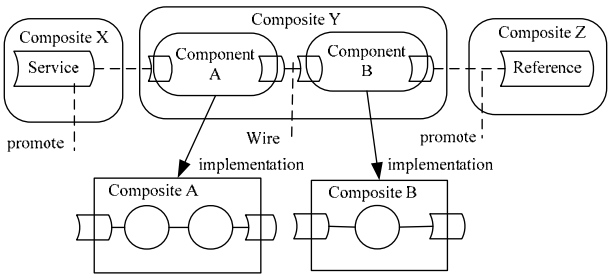

图1 SCA Domain结构

采用何种形式作为implementation的实现类型，直接关系到移植后的组件的灵活性和扩展性。以DLL形式遗留程序为例，如果以DLL或者任何需要编译的模块作为其实现的实体，那么任何对模块的改变都免不了编译环节，而且Component间的依赖关系的变化也将带来一系列代码修改与编译环节[3]。因此，implementation的实现类型必须考虑在不改变遗留程序结构情况下，提高它的灵活度，最好能够在Component间关系改变时只修改一些配置文件而不需要重新编译。同时，考虑到遗留系统的多样性，如果实现类型局限于原有类型，则将会导致新的遗留程序类型无法通过现有方案移植。综合考虑，需要通过一种中间的实现形式来实现对原有的遗留系统接口的访问，而且这种中间形式要有简单易

基金项目：国家部委预研基金资助项目
作者简介：曹 迪(1983－)，男，硕士研究生，主研方向：SOA，面向对象技术，软件体系结构；陈 平，教授、博士、博士生导师；鲍 亮、胡圣明，博士研究生
收稿日期：2009-04-10 **E-mail：**caodixy@163.com

懂的特点，python 的特性决定了以它作为 SCA 实现类型的优势，首先 python 是脚本语言，灵活易懂，其次，python 在混合编程方面的优势使它能够直接或者通过扩展访问所有的遗留系统程序。

根据上述分析，作为一个部署中的移植后的遗留系统的 SCA 集合，它包含了对组件进行描述的 SCDL 文件，对原有遗留接口实现访问的中间 python 代码文件以及原有的遗留程序和必要的扩展模块[4]。本文的移植就是在上述环境下进行研究和实现的。

2.2 移植方案设计

根据上述对 SCA 平台的分析，实现对遗留程序的移植可以分 4 个步骤，如图 2 所示。

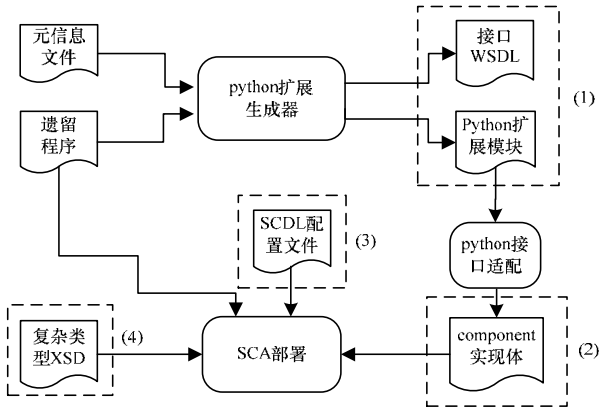

图 2 移植方案

对图 2 的说明如下：

(1)首先以遗留程序和元信息文件作为输入，通过 python 扩展生成器实现对遗留程序的扩展包装，使原始接口可以通过 python 扩展模块形式进行访问。

(2)通过中间的 python 代码实现对遗留程序扩展模块的转调，同时将遗留系统中的复杂类型的接口参数转化为 python 中的 Element Tree，并在代码中实现复杂类型对象的实际构造和传参。

(3)用服务组件定义语言(Service Component Definition Language, SCDL)对其实现体进行描述。

(4)在 TUSCANY SCA 环境中，所有复杂类型的处理都是以服务数据对象(Service Data Objects, SDO)为输入输出类型的，因此，需要在部署环境中置入复杂类型的 XSD 文件。

就本质而言，移植的流程实现了功能的移植，概念的移植和数据移植 3 个部分移植，第(1)步、第(2)步完成了功能的移植，在不改变原有接口功能的情况使遗留程序对外的表现形式从过去的二进制代码和应用程序转变为 SCA 系统所支持的 py 文件。第(3)步实现了对概念的移植，通过对功能移植后的 python 模块的 Component 化，并在业务层上对其 Composite 化，使其从一个单纯的 python 模块转变为 SCA 模块。第(4)步实现了数据的移植，通过对复杂类型的序列化描述，实现数据在输入输出时的转换，并完成输入输出数据的验证。

3 功能移植的核心技术研究与实现

3.1 Python 扩展生成器的体系结构

功能移植环节的本质是通过 python 扩展生成器生成可以被 python 脚本调用的 python 模块，比较常用的方法是使用 python/c API 在 C++代码中编写扩展模块或者使用 boost. Python 类库编写扩展代码，对外暴露出 C++代码中的接口，然后进行重新编译，从而生成 python 扩展模块。然而这都需要取得遗留程序的源代码，即使有源代码，针对每个遗留程序进行扩展代码的编写和编译也是一个复杂耗时的过程。企业级的应用要求服务的集成人员并不需要了解太多的技术细节，因此，python 扩展生成器的自动化是必要的。如图 3 所示，python 扩展生成器主要包括 3 个环节，首先输入遗留程序和元信息描述文件，建立相应的元信息模型，然后根据元信息模型生成相应的中间转调代码和扩展代码，最后借助于 makefile 文件生成器所生成 make 文件，完成中间代码的编译和链接，生成最终的 python 扩展模块，而 WSDL 文件则作为 python 扩展生成的附属品，以便于后续的基于 Web services 的调用。

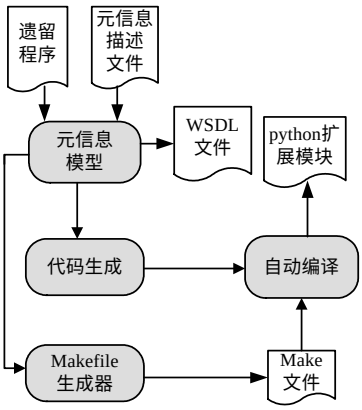

图 3 扩展生成器结构

3.2 元信息模型

服务元信息是集成框架进行包装的信息基础，集成框架用一组内建的元信息模型来存储这些信息供包装的不同阶段使用。而元信息模型提供一组接口供客户程序操作元信息，通过这组接口所有服务元信息就可以被添加到元信息模型中，反之亦可取出。元信息模型主要包含 2 种信息：服务包装信息和接口信息。

服务包装信息大体分为 2 类：服务类型无关信息和服务类型相关信息。服务类型无关信息是各种不同类型遗留系统服务所共有的，如服务的名字、目标存放位置等；服务类型相关信息则由于各遗留系统服务实现技术的不同而不同。

接口描述信息是遗留系统对外提供服务的详细接口定义，集成框架以这些定义为基础，生成接口的服务的代理接口及 WSDL 文件。

4 遗留系统的概念迁移

遗留系统通过 python 扩展生成器扩展后生成 python 扩展模块，实现遗留程序的功能迁移，但如果没有相应的配置文件来完成它在概念上的转换，它不能称为一个 SCA 服务组件。在对 SCA 平台迁移进行分析时可以看出使用 py 文件作为 Component 的实现体，有灵活易扩展的优势，经过接口适配后，SCA 组件的 Component 实现体已由过去的 C++代码形式转化为以 py 文件为扩展模块代理的 python 代码片段。但 Component 并不直接对外暴露其 Services，而是通过 Composite 完成其逻辑层次的概念的。Composite 组件是 SCA 规范中最基本的单元，是部署的最基本单位，因此，必须通过 SCDL 文件的配置来完成 Composite 的概念迁移。根据 SCA1.0 规范中 Composite 的定义：

```
<element name="Composite" type="sca:Composite"/>
<complexType name="Composite">
    <sequence>
    <element name="include" type="anyURI" minOccurs="0" maxOccurs="unbounded"/>
    <choice minOccurs="0" maxOccurs="unbounded">
    <element name="service" type="sca:Service"/>
    <element name="property" type="sca:Property"/>
    <element name="component" type="sca:Component"/>
    <element name="reference" type="sca:Reference"/>
    <element name="wire" type="sca:Wire"/>
    </choice>
    </sequence>
    <attribute name="name" type="NCName" use="required"/>
    <attribute name="targetNamespace" type="anyURI" use= "required"/>
    <attribute name="local" type="boolean" use="optional" default= "false"/>
    <attribute name="autowire" type="boolean" use="optional" default= "false"/>
    <attribute name="constrainingType" type="QName" use= "optional"/>
    <attribute name="requires" type="sca:listOfQNames" use= "optional"/>
    <attribute name="policySets" type="sca:listOfQNames" use= "optional"/>
    <anyAttribute namespace="##any" processContents="lax"/>
    </complexType>
```

完成遗留系统在概念的上 Composite 化移植，需要完成以下几项信息的配置：component 元素，service 元素，property 元素，reference 元素，wire 元素以及 name 等一系列属性的配置。

当遗留系统作为单一的一个整体出现时，即其功能迁移和接口适配后的 py 文件作为一个单一 component 实体出现在 composite 中，reference, wire 等概念是不存在的，因为无论 wire 还是 reference 都体现了 composite 内部或者跨越 composite 的 component 之间的关联关系，而进行迁移的实体是作为一个整体出现的，可以视作单 component 的。那么只要完成 component 元素、service 元素以及 property 元素的配置，就可以完成 SCDL 文件，从而完成 composite 的概念化。表 1 给出 SCDL 的必须配置信息与现有信息的映射关系。

表 1　配置信息模式表

SCDL 元素	含义实体	配置模式
Component	Python 适配文件	name="组件名" module="python 适配文件名"
Service	与具体协议绑定的访问接口	Reference value= Component 名称
Property	Composite 中定义的可配置变量	name="属性名" type="类型" value=默认属性值

5　复杂类型的数据迁移

TUSCANY SCA 核心对复杂类型的处理统一采用 SDO 结构，而对应的 python 实现扩展则采用 Element Tree 结构作为复杂类型的输入输出类型，两者通过 XML 字符串作为转换中介，在由 Element Tree 解析出的 XML 字符串构建 SDO 结构时，需要复杂类型的 SHEMA 文件来完成验证以及数据图结构的建立。反之，当 TUSCANY SCA 完成数据处理后，在输出数据，解析 XML 字符串时，同样需要 XSD 文件作为解析的依据，并最终由 XML 字符串来构造 Element Tree，作为 python 实现的输出结构。图 5 给出了 XSD 文件在数据迁移过程中的位置。

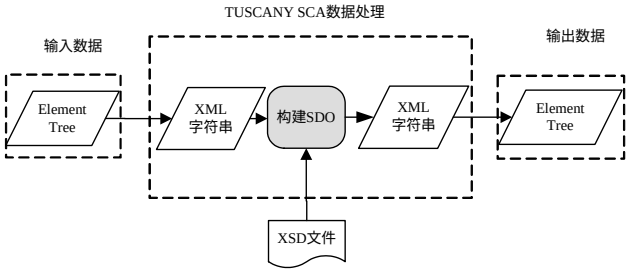

图 5　TUSCANY SCA 复杂数据处理流程

6　迁移后的 SCA 组件部署

经过上述各部分的移植，实现了从遗留系统到 SCA 组件的各部件，要使其能为 SCA 环境所用，必须将其部署到 SCA 的系统中去。SCA 环境以 Composite 作为其访问单位，在概念迁移过程中的 SCDL 文件配置，实现 Component 组件的概念化，但该 Composite 部署到 SCA 环境中后，还需要对属性等部分作以配置。SCA1.0 规范定义 Composite 可以包含一个或多个 Component，而 Component 又可以将 Composite 作为其实现的实体，因此，完成 SCA 组件的部署还要用一个实例 Composite 文件来描述 SCA 组件。实例 Composite 文件格式如下所示：

```
<composite "name="实例名">
  <component name="实例 Component 名">
    <implementation.composite
      name="移植后的 composite 名" />
  </component>
</composite>
```

7　结束语

本文提出一种基于 SCA 的遗留系统移植方案，介绍其工作流程和核心技术，并从移植的 3 个不同的角度对其进行分析，研究一种自动化的 python 扩展生成方案，今后的工作将围绕移植方案的自动化展开。

参考文献

[1] Jesus B. Legacy Information Systems: Issues and Directions[J]. IEEE Software, 1999, 16(5): 103-111.

[2] Eric N. Understanding SOA with Web Services[M]. 北京: 电子工业出版社, 2004.

[3] Jack H. Code Generation in Action[M]. [S. l.]: Oreilly & Associates Inc., 2003.

[4] Graham B. Service Component Architecture Specifications[EB/OL]. (2008-07-15). http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications.

编辑　陈　文