计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):52−59 [doi: 10.15888/j.cnki.csa.006032]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

工具服务化集成中数据交互方式设计①

刘庆庆[1,2], 王丹丹[1], 宋梦蝶[1,2], 杨　涛[1,2], 王　青[1]

[1](中国科学院 软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院大学, 北京 100049)

摘　要: 针对当前工具服务化集成中数据交互普遍存在的语法异构和语义异构问题, 本文设计了一种基于 JSON 格式的语法定义形式和基于通用词表的语义异构处理方法的数据交互方式, 并对其实现方法和技术进行了详细的论述. 最后, 在一种软件开发工具服务化集成的实际案例中进行了实验, 对所提出的数据交互方式进行了实现并对其有效性进行了验证. 实验结果表明, 所设计的数据交互方式较好地解决了上述两个问题, 为工具集成提供了良好的数据交互基础.

关键词: 工具服务化集成; 数据交互; JSON; 通用词表

引用格式: 刘庆庆,王丹丹,宋梦蝶,杨涛,王青.工具服务化集成中数据交互方式设计.计算机系统应用,2017,26(11):52–59. http://www.c-s-a.org.cn/1003-3254/6032.html

Design of Data Interaction Manner in Service-Oriented Tool Integration

LIU Qing-Qing[1,2], WANG Dan-Dan[1], SONG Meng-Die[1,2], YANG Tao[1,2], WANG Qing[1]

[1](National Engineering Research Center for Fundamental Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: In view of the widespread problems like syntax heterogeneity and semantic heterogeneity in the service-oriented tool integration, this paper proposes a new data interaction manner and gives a detailed description of its implementation method and related technology. The manner's syntax definition is based on JSON format and it also takes an advantage of general vocabulary to handle the problem of semantic heterogeneity. At last, we implement the manner in an actual case of the integration of common software development tools, encapsulating tool's function as service. The experimental results show that the design of data interaction manner can well solve the problems mentioned above, thus provides good data interaction base for tools integration.

Key words: service-oriented tool integration; data interaction; JSON; general vocabulary

软件工程是一系列过程的集合, 通常包括需求分析、概要设计、详细设计、编程实现、测试和部署等阶段. 为了提高工作效率, 每个阶段都提供了许多辅助工具. 如项目管理工具 JIRA, 代码管理工具 Gitlab, 测试管理工具 Mantis 等. 这些工具的使用对实现软件工程自动化、提高软件过程中的工作效率等有着重要的意义, 然而不可忽视也产生了一些问题, 主要表现在: (1) 各个工具相对分散, 不利于工具间的交互和协作, 造成"信息孤岛"; (2) 缺乏对各个工具的统一管理, 不利于对软件开发过程的控制; (3) 对软件开发人员之间的交流和协作造成了很大的不便; (4) 不利于软件开发过程的标准化和规范化. 因此, 如何有效地将这些分布、异构的、运行在不同平台上的工具集成起来为项目管理与开发测试提供一套完整的软件项目支撑环境

① 基金项目: 国家自然科学基金 (61432001)
收稿时间: 2017-02-13; 修改时间: 2017-02-26; 采用时间: 2017-03-02

已成为一个亟待解决的问题.

工具集成是一个非常复杂的问题, 它涉及到多个不同抽象层面, 如用户界面集成、控制集成、异构的数据交互等[1]. 其中异构的数据交互[2]是最核心也是最关键的问题之一. 它的目的在于维护数据源整体上的一致性, 实现工具间数据通信和数据操作. 当前所采用的数据交互策略或耦合度高, 实现复杂, 数据冗余和不一致现象严重; 或数据表示复杂、解析繁琐、可行性低. 这些都制约着大规模、开放、灵活的工具集成系统的实现. 因此提供简单、高效、通用的数据交互方式对实现工具的灵活集成有着举足轻重的作用.

本文通过对工具集成中已有数据交互方式的分析和研究, 发现在当前工具服务化集成中普遍存在两个问题: (1) 语法异构: 不同工具的数据表示形式差别很大, 缺乏统一、简单的数据语法表示形式; (2) 语义异构: 不同工具的数据存在着严重的数据冗余和不一致现象. 鉴于此, 本文设计了一种基于 JSON 格式的语法定义形式和基于通用词表的语义异构处理方法, 并应用到基于总线的数据交互方式中. 本文的结构如下: 第 1 节为相关背景及技术介绍, 第 2 节为数据交互方式的设计, 第 3 节为数据交互过程, 第 4 节给出案例分析, 第 5 节是结语.

1 相关背景及技术介绍

数据交互作为工具集成中最基础也是关键的部分, 引起了大量国内外学者的研究兴趣, 目前存在的有代表性的数据交互模式大致分为如下几类.

点对点模式[3]. 这种模式在两种不同的工具数据源之间建立相互转换, 进行直接交互, 由于其基本采用硬编码的方式, 扩展性差, 当异构数据源增多时, 实现起来极其麻烦.

中心库模式. 该模式通过对各个工具数据模式的分析和转换, 将其中的数据存储到统一的中心库中, 并通过中心库对外提供统一的数据交互接口. 邹晓宇[4]和 B.Plogar 等人[5]分别以关系型数据库和文件系统为中心库, 实现数据交互.

模型转换模式. 该模式常用于基于元模型[6-8]的工具集成框架中, 将异构数据表示成一种元模型, 利用模型转换的思想对数据表示进行统一, 并通过转化后的统一模型对外提供交互接口. 常用的技术有基于 UML 的转换技术和基于图的转换技术[9], 以及基于 XML 的转换技术[10-12]. 丁庆涌等人[13]利用 Boost Spirit 技术实现数据格式转换和交互, 林毅等人[14]提出了基于元数据的模型转换和交互方式. 基于数据总线[15]的模式. 通过定义统一的数据总线使所有工具间的交互均通过数据总线完成, 避免工具间直接交互. Bergstra JA 等人[16]以脚本的形式给出了总线的定义, 郭兵等人[17]基于 CORBA 规范定义总线, 李松犁等人[18]以自定义的 SDModel 作为总线的定义形式, 并在此基础上实现工具集成. 本文选择在基于数据总线的模式下, 对提出的数据交互方式进行验证. 相比于其他交互模式, 基于总线的交互模式对工具间交互的控制能力更强, 也更加灵活方便[16].

1.1 现有的语法表示形式

XML 和 JSON 是两种常见的数据交换格式[19]. XML 由于其表示能力强, 跨平台性好等优点被广泛应用, 但也因为其文件格式复杂, 解析困难等为它的使用带来了不便. 相比之下, JSON 是一种更加简单、更为理想的数据交换语言[20], 因为: (1) JSON 采用完全独立于语言的文本格式, 易于阅读和编写; (2) JSON 只构建于两种结构: key/value 键值对和键值对数组, 使其利于机器解析和生成, 是一种轻量级的数据交换格式; (3) JSON 格式的数据均经过压缩处理, 相比于 XML 庞大的文件格式占用的带宽要小很多, 传输速率也更快. 因此本文选择基于 JSON 格式的语法定义形式.

1.2 现有的语义异构处理方式

语义异构的处理涉及语义分析和字符串相似度比较两个阶段, 语义分析可以通过去停用词和同义词表替换来实现, 最终归结到字符串相似度的计算. 常见的字符串相似度匹配算法有编辑距离[21]、最长公共子序列算法[22]、贪婪字符串匹配算法[23]、Needleman-Wunsch[24]算法和 Smith-Waterman[25]算法等.

编辑距离指两个字符串之间相互转换到对方形式所需要的最小编辑次数. 字符的替换、插入、删除是三种允许的编辑操作. 字符串相似度通过编辑距离的大小来衡量, 编辑距离越大则相似度越小.

最长公共子序列指将两个给定字符串分别删去若干字符后得到的最长相同字符序列. 子序列中的字符并不要求连续出现, 仅需满足出现的顺序和其在源串中的顺序一致即可.

贪婪字符串匹配算法对两个字符串进行贪婪式搜索以找出最大公有子串, 它需对要计算的两个字符串

进行多次搜索, 每次找到当前字符串中未“标注”部分的最长公共子串, 并将其标注为“已使用”, 避免最大匹配重复使用.

Needleman-Wunsch 算法对于长度为 m 和 n 的两条序列 SqA 和 SqB, 构造得分矩阵 X, 该矩阵的最后一个元素 X(m, n) 即为最优匹配的得分, 通过回溯得到最优匹配结果.

Smith-Waterman 算法是一种局部比对算法, 通过字母的匹配、删除和插入操作, 使两条序列达到同样长度, 在操作过程中, 尽可能保持相同的字母对应在同一位置. 比对时, 找出待比对序列中的某一子片段的最优比对. 该方法可能会揭示一些匹配的序列段, 而本来这些序列段是被一些完全不相关的残基所淹没的.

2 数据交互方式的设计

数据总线[16]概念的提出借助了计算机硬件中“总线”的概念, 是一个虚拟的同构或者异构组件间交互的通道, 它以一种通用的方式为各系统提供数据定义、数据传送和逻辑控制, 是各个组件之间通信的桥梁.

在工具集成中, 不同工具的数据源主要存在两种异构: 语法异构和语义异构. 语法异构是指不同的工具存储数据的格式各不相同; 语义异构指数据项的内容和含义的差异, 包括同名异义, 异名同义等. 针对这两种异构在数据交互中的问题, 本文对数据交互方式进行了设计: 为了解决数据源语法上的差异, 本文利用 JSON 格式对数据的表示形式进行统一定义; 为解决语义异构问题, 本文提出了基于通用词表的数据处理方法, 下面将详细介绍语法异构和语义异构的处理方式.

2.1 语法异构处理

由于数据总线要和各个异构数据源交互, 其数据格式必须具有良好的灵活性和可扩展性.

本文采用 JSON 作为数据总线的统一数据格式, 解决不同数据源语法异构问题, 各个字段本身也是自定义的 JSON 格式, 可以灵活扩展. 数据格式最外层中包含 8 个固定字段, 具体含义见表 1.

表 1 字段含义

字段	说明	类别
cmd	工具的标识, 服务注册并且经过验证后系统自动分配的标识号	
account	工具授权账号	工具管理相关
password	工具注册时提供的访问密码	
task_id	当前任务节点的id, 唯一标识当前任务节点	
assignee	节点负责人	节点相关
task_data	具体工具返回给数据总线的局部数据	
content	数据总线所包含的全局数据, 根据通用词表提取task_data中的数据至数据总线, 作为全局数据, 并推送给下一个工具	数据总线相关
callBack	工具回调地址	控制相关

在该定义中, cmd、account、password 属于工具相关字段, 每个工具都有唯一的标识和账号信息, 用于工具管理和权限控制; task_id、assignee、task_data 是和工作流节点相关的字段, 其中 task_id 是工作流节点的标识, assignee 是指定的节点负责人, task_data 字段存储具体工具返回的数据信息, 由各个工具推送给数据总线, 它也是一个 JSON 字符串, 其中的字段可以自定义和扩展, 不同工具所采用的字段名和含义可能相同, 也可能不同, 通过语义分析对 task_data 中的局部数据字段进行解析, 提取 task_data 中的局部数据到 content 字段中, 成为全局数据, 并推送给之后的各个工具; content 字段是一个全局字段, 存储所有工具的字段信息, 并对所有工具可见; callBack 是工具的回调地址, 主要控制工作流在完成某一节点的处理后的流转方向.

这样, 所有数据总线接收到的数据格式和推送给各个工具的数据格式都是统一的, 解决了不同工具数据源语法异构问题.

基于定义的语法格式, 下面给出一个例子:

```
{
  “cmd”: “2”,
  “account”: “5AV-GWEkKv1wmq66MH7d”,
  “password”: “”,
  “task_id”: “2163199”,
  “task_data”: “{
    “issueType”: “Sub-task”,
    “assignee”: “yangke2015”,
    “reportor”: “yangke2015”,
    “creator”: “shelly@163.com”,
```

“issue_id”: “SSRA-211”
}”,
“callback”: “http://ip:port/taskmgr/api/task/dealWithData”,
“content”: “{
“issueType”: “Sub-task”,
“assignee”: “yangke2015”,
“reportor”: “yangke2015”,
“creator”: “shelly@163.com”,
“issue_id”: “SSRA-211”
}”
}

2.2 语义异构处理

在集成过程中, 由于被集成的各个工具之间本身就存在业务相关性, 很多数据用语在不同的工具中可能表达不同的含义, 也可能只是名字不同而含义相同, 为了解决异构数据源的语义异构问题, 避免数据冗余和不一致, 提出了基于通用词表的处理方式.

各个工具将其在日常开发过程中常用的业务用语注册到通用词表中, 新注册的业务用语和通用词表中已存在的字段进行相似度比较, 对语义完全相同的字段不再进行重复注册, 并根据通用词表中的字段名对相同含义的参数进行统一命名处理, 在保证字段完整性的前提下, 避免字段冗余.

通用词表在注册时包含字段名、字段别名、字段描述三个属性信息, 在计算新注册字段和已有字段的相似度时, 需要分别对三个属性的相似度进行计算. 同时, 为了减少无意义字符的影响, 首先对两个字符串进行分词、去停用词处理, 经过处理后的字符串长度明显减小; 为了解决字符串异名同义的问题, 引入同义词典, 对同义词进行替换. 由于处理后的字符串都相对较短, 而且不要求全局匹配, 因此, 本文采用 Smith-Waterman 算法计算两个字符串相似度, 具体如公式 (1) 所示.

$$\mathrm{sim} = \sigma_1 \times \mathrm{sim}_{\mathrm{name}} + \sigma_2 \times \mathrm{sim}_{\mathrm{nike}} + \sigma_3 \times \mathrm{sim}_{\mathrm{des}} \quad (1)$$

其中 sim 表示整个字段的相似度得分, σ_1、σ_2、σ_3 分别代表字段的名字、别名、描述信息在相似度计算中的权值, 用户可以根据具体情况设置, $\mathrm{sim}_{\mathrm{name}}$、$\mathrm{sim}_{\mathrm{nike}}$、$\mathrm{sim}_{\mathrm{des}}$ 分别表示按照字段的名字、别名、描述信息计算出的相似度得分. 根据相似度的计算结果, 依照 TopK[26]的思想给出相似度得分最高的 K 个字段, 用户根据推荐情况判断语义相同的字段是否已有存在. 若确定语义相同的字段已经存在, 则不再重复注册, 并以通用词表的字段名为标准修改自己的字段名; 反之则继续注册, 新的字段被加入通用词表中.

随着所集成的工具的增多, 通用词表的信息不断积累, 而且只有在遇到新业务时才需要扩充通用词表, 经过长时间的积累, 通用词表的信息量和范围越来越大, 所需要录入的也相对减少.

3 数据交互过程

基于第 2 章设计的数据交互方式, 工具间的数据交互过程如图 1 所示.

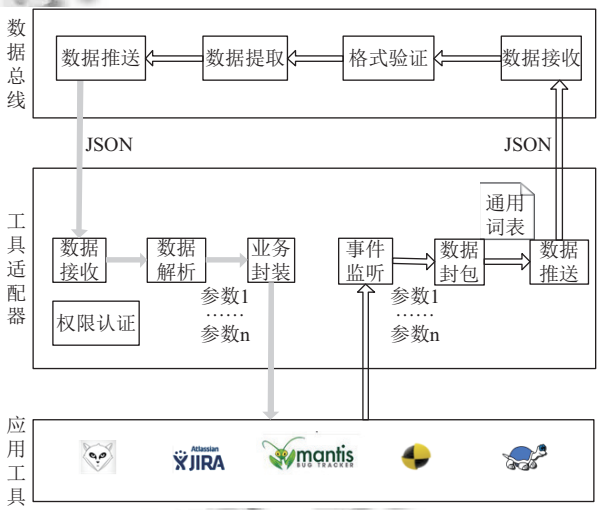

图 1 数据交互过程

在整个工具集成平台中, 工具与工具间以及工具与集成平台间的交互均通过数据总线和各个工具适配器相互配合来实现.

其中, 数据总线必须满足 2.1 中定义的语法形式. 其具体的功能包括数据接收、格式验证、数据提取和数据推送. 数据接收主要接收工具适配器推送给数据总线的数据; 格式验证是对接收的数据进行格式校验, 确保满足 2.1 中所定义的语法格式; 数据提取是对数据中的 task_data 字段中的局部数据进行处理, 提取其中的内容到全局字段 content; 数据推送则是将处理之后的数据推送给各个工具适配器.

工具适配器是各个工具自己维护的数据接入层, 一方面, 它接收数据总线传过来的满足语法定义的 JSON 数据, 通过对相关字段 (content) 的解析获取需要的参数信息, 进而调用有关的业务服务, 实现对工具的

操作, 这一过程如图中实心箭头所示; 另一方面, 各个应用工具的数据经过工具适配器的封装后, 表示成满足语法定义的 JSON 形式, 并推送给数据总线, 由数据总线对其数据格式进行校验, 并进行数据提取, 之后再通过数据总线推送给其他工具, 这一过程如图中的空心箭头所示.

工具适配器的实现方式没有统一的规定和限制, 各个工具根据各自的架构特征完成该层的设计与实现, 但需要遵循统一的接入规范, 如 RESTFul 等. 总体上看, 工具适配器的功能是相对独立和固定的, 具体包括权限认证、数据接收、数据解析、数据封包、数据推送和事件监听、业务封装. 权限认证负责对当前用户的操作权限进行检测; 数据接收负责接收来自数据总线传递来的数据; 数据解析对接收到的数据进行解析, 获取工具自身需要的数据; 数据封包是对经过工具处理之后的数据进行重新封装, 以满足数据总线所定义的语法形式, 在封包的过程中, 会首先将数据中要用到的参数信息注册到通用词表中, 按照 2.2 中的设计对参数进行语义分析和处理, 根据处理的结果对参数的命名进行调整, 避免字段冗余和不一致; 数据推送是将封装好的数据重新推送给数据总线, 驱动工作流节点继续向下流转; 事件监听主要是监听工具中的事件信息并获取相关数据; 业务封装是对工具中常见的业务进行服务封装, 实现通过适配器操作工具.

整个过程中, 数据总线和工具适配器相互配合, 依照定义好的语法格式和通用词表信息, 共同完成数据通信和信息交互. 当新的工具需要接入集成平台时, 只需增加连接数据总线和工具的工具适配器, 负责总线数据格式和工具自身数据格式之间的转换和解析, 以及工具业务逻辑的封装, 避免了工具间直接通信.

4 案例分析

本文以软件集成开发平台为例, 对所提出的数据交互方式的有效性进行了验证, 下面将给出实验的详细介绍.

4.1 软件集成开发平台简介

软件集成开发平台是一个 Web 的应用工具集成平台, 主要用于集成软件生命周期中的常用辅助工具, 如任务管理工具 JIRA、代码管理工具 Gitlab、测试管理工具 Mantis, 文档管理工具 SecFile 等, 从而实现软件开发全生命周期的集成管理. 该集成平台以基于工作流的工具集成框架为基础, 将各个工具的功能封装成服务, 注册到自定义的流程模板中, 形成可配置的流程. 本文所提出的数据交互方式即应用到该集成平台中. 平台的架构如图 2 所示.

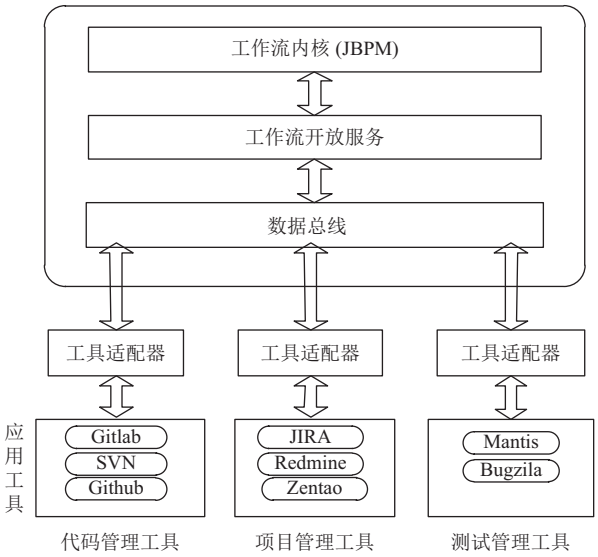

图 2 软件集成开发平台架构图

整个框架分为五层: 工作流内核层、工作流开放服务层、数据总线层、工具适配器层以及应用工具层.

(1) 工作流内核层主要是以 JBPM(Java Business Process Management) 为驱动, 负责整个软件过程的业务流程定义、管理、状态转移和协作执行等.

(2) 工作流开放服务层是对工作流的一些功能进行封装, 以通用服务的形式开放出去, 负责工具注册和管理、流程管理等.

(3) 数据总线层是各个工具和集成平台进行通信的桥梁, 这里使用了文中定义的语法形式, 并通过它实现工具和工具之间以及工具和整个集成平台之间的数据交互.

(4) 工具适配器层主要是为了实现具体工具和数据总线间的适配, 所有的应用工具都必须采用定义好的数据语法格式进行描述, 并进行功能封装和数据封装. 工具适配器和数据总线相结合, 按照本文所描述的交互方式, 实现工具间的数据交互.

(5) 应用工具层是软件工程的各个阶段所需要使用的具体辅助工具, 可以根据需要自行选择和组合使用.

4.2 数据交互过程

在软件集成开发平台中存在如下由 JIRA、Mantis、

Gitlab 组成的业务流程: 测试人员在 Mantis 中提出一个 bug, 自动在任务管理工具 JIRA 中创建一个 issue, 经过 PM 的确认后, 如果确定它不是一个 bug 则直接把该 issue 关掉, 反之, 将它指派给一个人; 当 Gitlab 中提交该任务相关的代码并创建一个合并请求后, JIRA 中相应的 issue 下面会添加一条评论信息, 记录该 issue 的相关动态, 当 Gitlab 中的合并请求通过代码评审之后 JIRA 中相应的 issue 状态会改变, 同时 Mantis 中对应的 bug 的状态也会随之改变. 这一业务流程如图 3 所示.

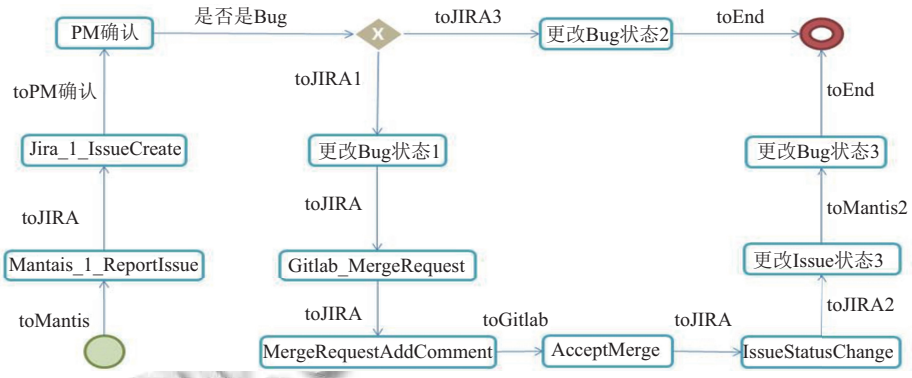

图 3 软件开发流程

4.2.1 语法异构问题处理

语法异构问题依照 2.1 所给出的设计进行处理, 下面以前两个节点为例进行详细的说明.

首先, Mantis 必须经过 Mantis 适配器将自身数据封装成 2.1 中所定义的语法格式, 通过数据总线传递给 Jira 适配器. 具体数据格式和内容如下:

```
{
  “cmd”: “34”,
  “account”: “LKS-HDclisuemg82MDDHDFS”,
  “password”: “”,
  “task_id”: “346479”,
  “task_data”: “{
    “project”: “SDE”,
    “summary”: “系统任务列表页面的翻页功能失败”,
    “description”: “进入系统首页, 看到任务列表, 点击页面翻页, 失败”,
    “issue_type”: “Bug”,
  }”,
  “callback”: “http://ip:port/taskmgr/api/task/dealWithData”,
  “content”: “{
  }”
}
```

数据总线在收到数据之后, 首先会对数据格式的合法性进行验证, 包括 JSON 字符串的有效性, JSON 字符串中所包含的字段是否和定义的字段完全一致等; 然后根据通用词表的参数信息, 将 task_data 中的局部信息提取到 content 字段, 成为全局数据, 供之后的工具使用.

工具适配器在接收到数据总线的数据后, 首先对数据进行解析, 获取所需要的数据并调用相关服务. 为了验证本文所使用的 JSON 格式相比于 XML 在数据交互中的优势, 以 JIRA、Mantis、Gitlab 的集成中使用的数据为例, 分别从数据大小, 传输效率, 解析效率三个角度对二者进行对比.

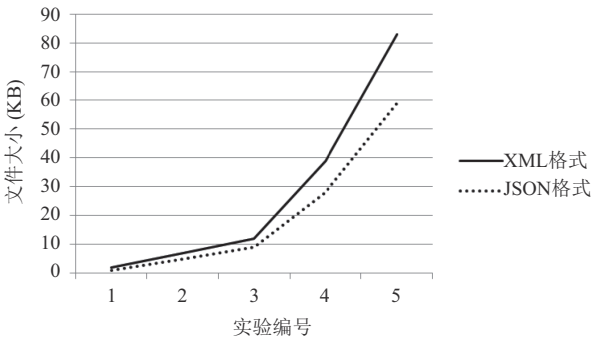

图 4 XML 和 JSON 文件大小比较

由图 4 可见, 表达同样的数据信息, JSON 格式相比于 XML 格式要小很多, 这主要是因为 XML 对任意

属性都要求开始和结束标签, 而 JSON 只需要一个属性名称.

数据大小将直接影响传输效率, 实验中分别统计了同样内容的数据以 XML 格式和 JSON 格式进行传输所消耗的时间, 为了减小误差, 每次实验进行 10 次, 然后求取平均值作为最终结果, 如表 2 所示.

表 2 JSON 和 XML 格式数据传输时间

数据大小(KB)	JSON (ms)	XML (ms)
2	5.89	7.57
5	14.72	19.29
12	20.08	28.51
20	25.51	36.98
50	48.37	73.52

具体的实验数据与网络环境及测试机的软硬件配置密切相关, 本文所采用的实验环境如下:

操作系统: Windows 7 旗舰版;

CPU: Intel(R)Core(TM)i7;

内存: 4.00 GB;

服务器: Tomcat 7.0;

网速 1.50 MB/s~2.56 MB/s.

虽然不同的实验环境会影响实验结果, 但是整体上的趋势是不变的.

在数据解析方面, XML 是基于 DOM 的树状结构, 解析时需要考虑父子节点之间的复杂关系, JSON 只构建于两种结构: key/value 键值对和键值对数组, 解析起来要简单很多. JSON 在这些方面的优势均为数据交互提供了良好的基础.

4.2.2 语义异构问题处理

在语义异构处理方面, 依照 2.2 的设计, 依然以前两个节点为例进行详细说明.

在该集成平台中使用了服务封装等思想, 各个工具都是以服务的形式注册到流程模板中, 每一个服务都有一些参数需求, 其中 Mantis_ReportIssue 服务和 Jira_CreateIssue 服务所需要的参数信息如表 3 所示.

表 3 服务参数信息表

Mantis		Jira	
Project	项目名	Project	项目名
Summary	项目概述	Summary	项目概述
Description	详细描述	Description	详细描述
Issue_type	Issue类型	Issuetype_name	Issue类型

可以看到, 参数 Issue_type 和 Issuetype_name 所表述的含义其实是完全相同的, 只是在不同的工具中被命名成不同的名字, 利用本文所设计的语义异构处理方式对参数进行处理.

处理结果如图 5 所示, 可以看到当参数 issuetype_name 再进行注册时, 会给出相似字段的提示, 由于相同含义的字段已经存在, 则不再对进行重复注册, 并以通用词表为依据修改 Jira 中的参数命名.

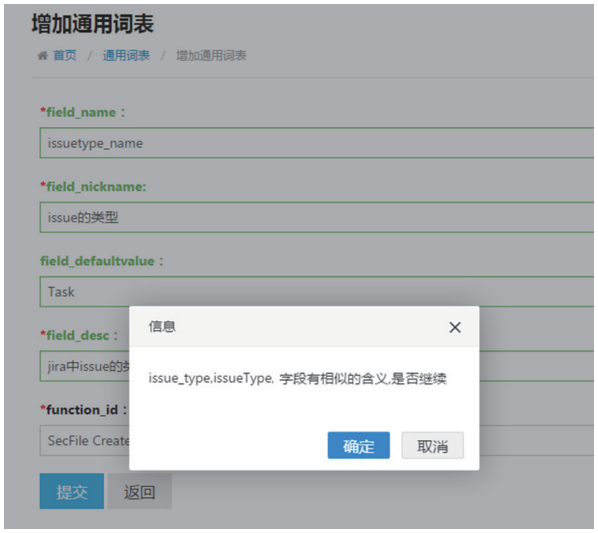

图 5 语义异构处理示例

应用该语义异构处理方式, 对 JIRA, Gitlab, Mantis 集成中, 参数注册情况进行分析, 结果如表 4 所示.

表 4 参数注册统计

工具名	所需参数数目	实际注册参数数目	注册参数减少的比例(%)
JIRA	27	27	0
Gitlab	9	6	33.33
Mantis	6	3	50

由于 JIRA 是第一个注册的工具, 需要依据其参数信息对通用词表进行初始化, 由表 4 可见, 当 Gitlab 和 Mantins 再相继集成进来时所需要注册的参数数目明显小于其实际需要的参数数目, 这是由于通用词表中已有一些相同含义的参数存在并被本文所使用的算法检测到, 可以直接使用, 进而避免了数据冗余和不一致的出现.

可以看到, 整个流程中, 工具间的数据交互都是按照本文中所设计的交互方式通过数据总线和各个工具适配器相互配合来完成, 其间对语法和语义异构的问题进行了处理. 当需要集成新的工具时, 只需要加入该工具的适配器, 使工具满足定义好的数据交互语法格式, 相比于现有的集中复制或者点对点的数据交互方

式要更加简单灵活.

5 结语

本文基于数据总线的数据交互模式, 对工具集成中的数据交互方式进行设计和改进, 提出了一种基于JSON 格式的语法定义形式和基于通用词表的异构语义处理方式. 所设计的数据交互方式将工具从紧密的业务关联中解耦, 所有工具间交互以及工具与集成平台之间的通信都是通过数据总线完成, 避开工具间直接通信, 并由各个工具适配器维护工具和数据总线之间的数据适配性. 通过实际的软件集成平台的对所提出的数据交互方式的可行性进行了验证.

目前, 数据交互方式的规范性和通用性还有待提高, 数据在交互过程中的安全尚未经过特殊的安全保障处理. 这些将成为我们下一步的研究重点.

参考文献

1 陈磊. 软件测试工具集成研究与应用[硕士学位论文]. 衡阳: 南华大学, 2015.

2 王凯. 面向工具集成的数据集成技术研究[硕士学位论文]. 西安: 西安电子科技大学, 2009.

3 张庆福, 万麟瑞. 基于 SOA 的异构数据集成软件架构研究. 计算机技术与发展, 2011, 21(5): 17–21.

4 邹晓宇. 基于工作流的软件过程开发平台的研究[硕士学位论文]. 长沙: 国防科学技术大学, 2005.

5 Polgár B, Ráth I, Majzik I. Model-based integration framework for development and testing tool-chains. FORMS/FORMAT 2010: the 8th Symposium on Formal Methods for Automation and Safety in Railway and Automotive Systems. Berlin, Heidelberg, Germany. 2011. 227–235.

6 Karsai G, Lang A, Neema S. Design patterns for open tool integration. Software & Systems Modeling, 2005, 4(2): 157–170.

7 Amelunxen C, Klar F, Königs A, *et al*. Metamodel-based tool integration with moflon. Proc. of the ACM/IEEE 30th International Conference on Software Engineering. Leipzig, Germany. 2008. 807–810.

8 Zhang WQ, Møller-Pedersen B, Hansen KT. Metamodel-based tool integration. Norsk Informatikkonferanse, 2011: 807–810.

9 BRAUN P. Metamodel-based integration of tools. Proc. of the ESEC/FSE Workshop on Tool Integration in System Development. Helsinki, Finland. 2003.

10 朱夏, 王茜. 异构系统间数据交换模型的设计与实现. 东南大学学报 (自然科学版), 2006, 36(2): 226–231. [doi: 10.3969/j.issn.1001-0505.2006.02.010]

11 邱丽丽, 俞烽. 异构数据动态交互平台设计与实现. 计算机应用与软件, 2013, 30(3): 182–185.

12 陈祖元. 基于消息队列的某异构数据交互系统的设计与实现[硕士学位论文]. 北京: 中国科学院大学 (工程管理与信息技术学院), 2014.

13 丁庆涌. 面向工具集成的数据格式转换技术[硕士学位论文]. 西安: 西安电子科技大学, 2007.

14 林毅, 宁洪, 王挺, 等. 基于元数据的数据整合平台. 计算机应用, 2008, 28(S2): 209–212.

15 蔡勇, 桑楠, 熊光泽. 一种基于工具总线的 CASE 集成模型. 计算机应用, 2004, 24(3): 137–140.

16 Bergstra JA, Klint P. The ToolBus coordination architecture. International Conference on Coordination Languages and Models. Cesena, Italy. 1996. 75–88.

17 郭兵, 刘强, 赵平原, 等. 一种基于 CORBA 规范的工具总线 LambdaBus 的设计与实现. 计算机应用, 2003, 23(8): 37–40.

18 李松犁, 张型龙, 肖俊超. 面向服务集成的自动化服务注册方法. 计算机应用与软件, 2016, 33(6): 59–63, 72.

19 Nurseitov N, Paulson M, Reynolds R, *et al*. Comparison of JSON and XML data interchange formats: A case study. Proc. of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering. San Francisco, California, USA. 2009. 157–162.

20 Lin B, Chen Y, Chen X, *et al*. Comparison between JSON and XML in applications based on AJAX. Proc. of 2012 International Conference on Computer Science & Service System. Nanjing, China. 2012. 1174–1177.

21 Levenshtein VI. Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady, 1966, 10(8): 707–710.

22 牛永洁, 张成. 多种字符串相似度算法的比较研究. 计算机与数字工程, 2012, 40(3): 14–17.

23 于海英. 字符串相似度度量中 LCS 和 GST 算法比较. 电子科技, 2011, 24(3): 101–103, 124.

24 Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 1970, 48(3): 443–453. [doi: 10.1016/0022-2836(70)90057-4]

25 Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology, 1981, 147(1): 195–197. [doi: 10.1016/0022-2836(81)90087-5]

26 Deshpande M, Karypis G. Item-based top-N recommendation algorithms. ACM Trans. on Information Systems, 2004, 22(1): 143–177. [doi: 10.1145/963770]