基于AADL的软件重构工具设计与实现

李 龙 董云卫 覃杨森 张 凡
(西北工业大学计算机学院 西安 710072)

摘 要 模态(Mode)表示的是软件可操作配置的状态,在一个或多个模态内可对资源和属性进行重新配置,即重构。目前,在对VxWorks系统下C程序的AADL架构进行模态的提取和蓝图制定过程中,还没有现成的工具。因而,设计一个基于AADL架构的软件重构工具将给软件架构的重构提供更大的帮助。为了搭建起基于AADL架构的软件重构可扩展工具平台,在Eclipse开源开发环境下,设计了SRM²(Software Reconfiguration Middleware based on Mode)插件工具。SRM²工具主要完成对C程序的AADL架构的扫描进而描述程序架构的静态蓝图信息,以及结合代码(探针设计和植入)在VxWorks系统下的运行信息生成动态蓝图,从而指导软件的重构。

关键词 模态,重构,AADL,蓝图
中图法分类号 TP391 **文献标识码** A

Design and Implementation of Software Reconfiguration Tool Based on AADL

LI Long DONG Yun-wei QIN Yang-sen ZHANG Fan
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract Modes represent alternative operational states of software. In multiple modes system, we can make different configuration such as resources and properties. Up to now, there isn't any tool which can abstract the modes of AADL architecture and establish blueprint in the program written in C language under the VxWorks system. Therefore, designing a software reconfiguration tool based on AADL architecture will provide great help to software architecture reconfiguration. To set up an expandable platform of software reconfiguration for AADL architecture under Eclipse open source development environment, we designed SRM² (Software Reconfiguration Middleware based on Mode) plug-in tool. SRM² firstly scans AADL architecture in C program as to describe the static blueprint information of program architecture, then it generates dynamic blueprint according to code run-time information, which can be got by an code probe, under the VxWorks system, the tool can guide software reconfiguration.

Keywords Mode, Reconfiguration, AADL, Blueprint

架构分析与设计语言(AADL[1])是一种可扩展的标准构架描述语言。AADL具有标准元模型(meta-model)、图形定义和文本语言属性,以其良好的构件封装特性以及对非功能属性描述的支持被应用于高可靠嵌入式软件的设计和开发中,并且还可以通过用户可定义属性和用户定义附录进行扩展。基于Eclipse框架的开源开发环境,设计了SRM²(Software Re-configure Middleware based on Mode)工具,目的在于由C代码的AADL架构提取其模态信息实现静态蓝图规划,以及获取软件运行时的动态信息并结合蓝图规划信息生成动态蓝图来模拟软件运行过程中模态的迁移过程,最终达到指导软件的重构[2]的目的。

1 工具开发基础

1.1 软件重构

随着软件规模的增大以及在计算系统中重要性的增强,软件领域开展动态重构[3]变得十分必要。在AADL的描述中,模态(Mode)表示的是可操作配置的状态,在一个模态内可对资源和属性进行配置。多模态是指软件系统处于多种可配置的状态,并且在模态之间能够进行模态迁移。软件重构是实现多态计算的重要手段,它能够在构件发生故障之后,在不改变软件功能和外部可见性的条件下对软件结构进行重配置进而能够使系统适应环境的改变,提高系统资源的利用率,并通过检测软件的执行状况动态地优化系统,使其在不需要人工干预的情况下自动地恢复系统错误和失效[4],以满足高可靠需求。

1.2 Eclipse插件体系结构

Eclipse插件开发平台是众多扩展点和扩展的结合[5]。这种架构使Eclipse非常易于扩展。在Eclipse SDK中,除了Eclipse平台外,还包括了Java开发工具(JDT)和插件开发环境(PDE)两个组件来支持插件开发。Eclipse平台和JDT、

到稿日期:2010-08-09 返修日期:2010-11-10 本文受国家自然基金重点课题(60736017),国家863计划课题(2009AA01Z147)以及2009年度西北工业大学基础研究基金(JC200917)资助。

李 龙(1986-),男,硕士生,主要研究方向为嵌入式软件,E-mail:dragon629@163.com;董云卫(1968-),男,教授,博士生导师,主要研究方向为嵌入式软件设计、验证与测试;覃杨森(1986-),男,硕士生,主要研究方向为嵌入式软件;张 凡(1979-),男,讲师,主要研究方向为嵌入式软件设计、验证。

PDE 构成了 Eclipse 的三层结构。在三层结构基础上，可以遵循 Eclipse 平台定义的插件结构自行编写插件来扩展 Eclipse 平台的功能。

工具 SRM² 正是在 Eclipse 提供的插件机制下进行设计和开发的。SRM² 通过扩展 Eclipse 平台的扩展点，实现相应的功能需求开发，而且自身也获得了更大的功能扩展空间。

2 软件模态划分及组成设计

随着软件规模的不断扩大以及完成的功能增多，本文将软件系统分为多个层次。其中，使命是为达到特定目标而聚合在一起的可以满足高性能和高可靠性要求的计算任务集合。从软件需求分析的角度，使命完成了软件需求分析阶段定义的各个软件需求。软件系统根据完成任务的不同被分为多个使命，使命是对软件粗粒度的划分。从软件设计的角度，单一使命是由一个或多个功能模块完成，各个功能是实现并完成使命的基础。从软件开发的角度，根据构件化软件开发的原理，功能由一个或多个构件实现，构件是对软件细粒度的划分。这样，软件系统就存在使命-功能-构件这样的三层结构，如图 1 所示。

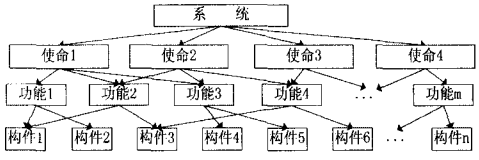

图 1 使命-功能-构件三层结构

为了对资源进行有效的配置，并优化软件系统，引入了模态的概念，模态是指系统中可配置的状态，配置的对象包括各种资源和属性。从运行的角度，软件系统在运行时具有多种不同的运行方式，模态是用来描述这些不同的运行方式的概念。

根据软件结构的不同层次，本文又将软件模态分为多个层次。在系统层次，软件系统具有多种使命，软件系统为完成某一种使命需要调用计算资源，执行相应软件功能的运行方式就成为系统的一种模态，记为 SM_i(System Mode)，i 表示系统的第 i 个模态。系统完成每个使命都会执行相应的模态，并且只有在该模态处于激活状态下，对应的使命才能处于运行状态。

在使命层次，一个使命的完成可以用不同的软件功能组合执行，因此，即便是同一使命也可以有多种运行方式，称其为使命模态，记为 M_iM_j(Mission Mode)，它表示使命 M_i 的第 j 个模态。在使命模态下，多个功能可以处于同一种使命模态，并且，只有处于同一个使命模态下的软件功能才可以处于同一运行状态。

在功能层次，一个功能的执行由多种软件构件组合完成，每一种软件构件的组成方式就称为使命下的功能模态，记为 $M_iF_jM_k$(Mission Function Mode)，它表示使命 i 下的功能 j 的第 k 个模态。一个构件可以处于一种或多种功能模态中，当且仅当该模态处于激活状态下，该功能模态对应的构件处于运行状态。

模态的层次关系可表示为：

(1)系统模态 SM_i(System Mode)：系统模态的集合 $A=\{SM_i \mid 1\leqslant i\leqslant n, n$ 为系统使命的总数$\}$

(2)使命模态 M_iM_j(Mission Mode)：使命模态的集合 $Bi=\{M_iM_j \mid 1\leqslant j\leqslant n, i$ 为系统 S 的某一使命编号，n 为完成使命 M_i 的功能配置方式的总数$\}$

(3)功能模态 $M_iF_jM_k$(Mission Function Mode)：功能模态集合 $C_{ij}=\{M_iF_jM_k \mid 1\leqslant k\leqslant n, i$ 为系统 S 的某一使命编号，j 为该使命下的某一功能 F 编号，n 为完成该功能 F_j 的软件构件配置方式的总数$\}$

软件架构[6]的重构主要通过软件模态的迁移实现。由于篇幅原因，此处只简单说明模态的迁移类型分为下述 3 类：系统模态迁移、使命模态迁移和功能模态迁移。

3 SRM² 工具体系结构设计

3.1 工具框架

SRM² 工具的设计主要是以 VxWorks 环境中运行的 C 软件代码的 AADL 架构抽象代码为对象，提取软件架构的模态信息实现重构蓝图以及结合蓝图信息和软件运行时的信息(通过探针的设计和植入实现)动态模拟软件运行过程中的模态迁移过程，进而反馈指导软件的重构。

根据前面软件模态的组成和软件重构的这些特点，结合构件化设计的思路，从易扩展的角度考虑，将 SRM² 工具分为 4 大功能模块：系统侦测、侦测分析、AADL 模态提取、蓝图规划。SRM² 工具的整体功能结构如图 2 所示。

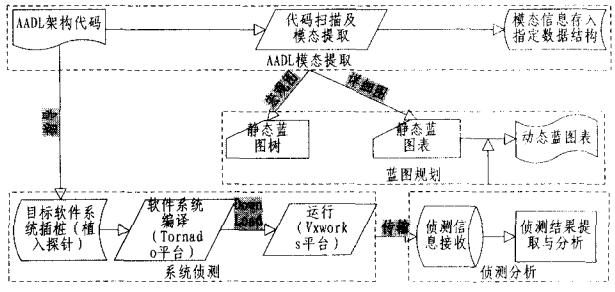

图 2 SRM² 工具结构图

(1)系统侦测：设计探针格式、制定侦测信息的存储格式以及设计宿主机和目标机的通信方式。具体流程为：通过 C 代码的 AADL 架构来指导探针的插桩策略的制定与实施，将植入探针后的 C 代码在 Tornado 平台上进行编译并 down 到 VxWorks 目标机平台上运行，软件在运行过程中会向宿主机发送模态迁移信息(即侦测信息)，该模块为宿主机端的侦测分析做好准备，便于后续功能模块的实施。

(2)侦测分析：设计侦测信息接收器接收系统侦测模块发回的侦测信息，并对发回的侦测信息进行提取、存储、分析和校对。存储的侦测信息为蓝图规划阶段做好部分准备。

(3)AADL 模态提取：根据软件模态及组成中制定的三层关系以及各层间的逻辑结构关系，设计模态提取的策略和流程，进而提取 AADL 文本中的模态信息，并对模态信息进行整理、分析和归类。接着对静态蓝图进行规划，其中包括静态蓝图树和静态蓝图表的规划。静态蓝图树用于描述 AADL 模态的宏观架构信息；静态蓝图表用于描述 AADL 模态的详细信息。

(4)蓝图规划：对动态蓝图进行规划，同时结合 AADL 模态提取模块中生成的静态蓝图和侦测分析模块输出的动态侦测信息，最终生成 AADL 模态的动态蓝图。该蓝图可以动态

模拟目标软件系统的模态迁移过程，从而便于设计人员对软件代码进行重构或者软件代码运行中的自动重构。

4 SRM² 工具功能构件设计

4.1 系统侦测模块设计

系统侦测模块活动图如图 3 所示，并按如下步骤进行：

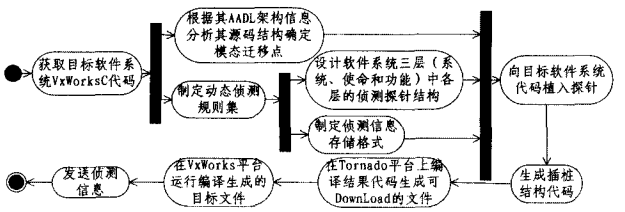

图 3 系统侦测模块活动图

步骤一 根据目标软件系统的 AADL 架构信息分析软件系统源码，用于指导后面的插桩[7]策略。

步骤二 动态侦测规则集的制定。其主要包括：元素、侦测信息的存储格式制定和探针设计。

元素：动态侦测规则集的元素为探针。探针具有层次化意义，可分为系统层探针、使命层探针和功能层探针。

侦测信息的存储格式制定：[关键字]:[模态名称]

例如："mission: SM1"，"function: MM2"，"component: MFM2"

说明：mission，function 和 component 是文件关键字，分别代表使命、功能和构件，":"是分隔符，"SM1"，"MFM2"和"MM2"是模态名称。输出规则集的制定主要是为了和后面的动态侦测信息文件的读取衔接起来，否则无法读取动态侦测信息文件。

探针的设计分为探针头、探针体和探针尾 3 个部分。

(1)探针头：探针头如图 4 所示，它在整个 VxWorks 程序中只有一个，主要植入在 VxWorks 主程序的开始部分，其作用是对所有探针体进行初始化设置。在探针头的内部，首先初始化 SOCKET 与相关缓冲区变量，然后加载网络设置配置文件并创建 UDP 套接字，接着判断套接字的创建和端口绑定是否成功，若失败则直接终止程序。

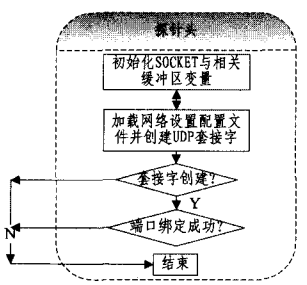

图 4 探针头

(2)探针体：探针体如图 5 所示，它在整个 VxWorks 程序中有 N 个(N 的值根据 AADL 程序中所有模态迁移事件的数目确定)，主要植入在 VxWorks 各个任务内部[8]的迁移事件的处理程序片段内部，其作用是根据不同的迁移事件类型发送不同的模态迁移信息。在探针体内部，首先，根据迁移事件类型分别作不同处理，"S"表示系统模态的迁移，"M"表示使命模态的迁移，"F"表示功能模态的迁移；判断完成则将不同类型的侦测信息按照约定的协议发送网络数据(侦测信息)。

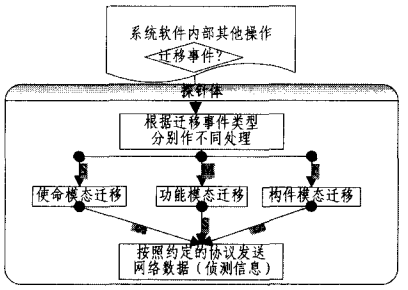

图 5 探针体

(3)探针尾：探针尾如图 6 所示，它在整个 VxWorks 程序中只有一个，主要植入在 VxWorks 主程序的最末尾部分，其主要作用是对所有探针体进行收尾工作。在探针尾内部，只有一个操作，即释放 UDP 套接字。

图 6 探针尾

步骤三 探针植入策略的制定。泵源：系统层初始模态；驱动：模态触发事件。AADL 4 个层次(系统层、使命层、功能层和构件层)中，每个层次都有一个初始模态，而模态运行由模态迁移事件所触发，故从顶层(系统层)出发，找到其初始模态(泵源)，再由模态触发事件触发(驱动)后续模态，这样模态就会源源不断地被触发运行。根据这一思想，插桩策略主要就是沿着模态迁移路径根据需要而植入相关层次的探针信息。

步骤四 将植入探针后的软件系统代码载入 Tornado 平台，然后进行编译与链接，得到目标代码。

步骤五 将上步生成的目标代码 down 到 VxWorks 平台运行。软件在 VxWorks 平台运行过程中通过网络通信向主机端发送侦测信息。

4.2 侦测结果提取与分析设计

在 VxWorks 平台下运行的基于 AADL 架构设计的 C 代码源程序中，VxWorks 平台是作为目标机，另有一台 windows 平台的宿主机与其进行网络通信，实时接收目标机端的动态侦测信息并存入指定文件。其活动图如图 7 所示，并按如下步骤进行：

图 7 侦测结果提取与分析活动图

步骤一 信息侦测接收器设计。信息侦测接收端核心是 SOCKET 网络通信。

步骤二 开启接收端软件，接收 VxWorks 宿主机端发送的侦测信息。

步骤三 分析接收到的侦测信息，按规定的文件输出规则集，将侦测信息写入指定的文件进行存储，以备后续蓝图规划所用。

步骤四 关闭信息侦测接收器。

4.3 模态提取模块设计

在对架构的模态扫描提取中，使用"多次扫描，分层建立

结构"的思路，按如下步骤进行，其模态扫描提取流程如图 8 所示。

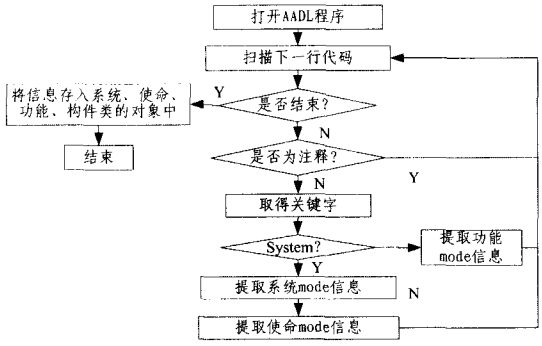

图 8　模态提取模块框架流程图

工具对 AADL 程序文件进行扫描，扫描过程中，主要通过程序中的关键字符完成对程序结构的识别，将识别出来的系统、使命、功能和构件的模态名称、模态之间的迁移条件以及三者相互之间的从属关系等信息存储到预定义数据结构中。

首先，打开 AADL 文本程序，开始扫描代码；判断是否结束，结束则将扫描分析得到的模态信息存入系统、使命、功能和构件数据类的对象中，然后结束程序，否则判断是否为注释，是注释则继续扫描代码下一行，否则取得关键字，判断关键字是否为"system"，是则提取系统的模态信息和提取使命的模态信息，否则提取功能的模态信息，下一步继续扫描代码，直到结束。

4.4　蓝图规划模块设计

蓝图规划模块设计包括：静态蓝图树、静态蓝图表和动态蓝图表。

各个蓝图设计均采用 MVC(Model-View-Controller)设计模式[9]开发，即模型-视图-控制器设计模式。MVC 设计模式将数据与视图分开，通过控制器来控制与数据的交互。以下分别对各蓝图设计进行描述：

静态蓝图树：蓝图树是从宏观的角度描述 AADL 模态信息。其树型结构分为 4 层，分别显示系统、使命、功能和构件的模态信息。根目录为系统，叶子节点为构件，中间部分为使命和功能信息，各层间有相应的逻辑关系进行关联，为了便于信息甄别，描述过程中各层分别采用不同的图标和颜色来更清晰地说明其含义。

静态蓝图表：蓝图表中包含了 AADL 模态的详细信息。首先规划重构蓝图信息表，进而根据重构蓝图信息表映射蓝图表的视图，将之前提取的模态信息与蓝图表的数据对象关联起来并映射到重构蓝图表中。创建蓝图表需要创建 3 部分内容：创建表格控制器(Controller)，创建表格模型(Model)和创建组织表格视图(View)。同时也为了便于信息甄别，描述过程中表格的各个栏目采用不同的图标和颜色标识各自所代表的含义。

动态蓝图表：动态蓝图表用来图形化描述程序的模态迁移过程的信息。首先设计蓝图表的表达形式，然后将静态蓝图表的模态信息与从 VxWorks 平台接收的动态侦测信息文件相结合，取得所需要的数据信息，再运用 MVC 设计模式将数据信息与动态蓝图表中的数据对象相关联，映射成动态蓝图表。最后动态蓝图演示部分采用多线程机制动态模拟 AADL 模态的迁移过程。采用多线程机制是为了提高程序的效率和性能，同时也使模态迁移演示的效果更加清晰直观。

在整个操作执行过程中，可以随时执行"重新演示"动作而重放动态演示过程，而且在动态演示过程中可以随时调整其演示速度。其中，在动态演示过程中，采用不同颜色突显存在模态迁移的模态名称，从而使其表达含义更加清晰明确。

5　SRM² 工作台设计

SRM² 工具的工作台扩展了 Eclipse 平台的 org. eclipse. ui. actionSets 功能构件，生成的插件会在 Eclipse 工作台的菜单中添加"蓝图"菜单项并在工具栏中添加"SRM2"工具栏按钮。通过以上两种触发方式，可启动 SRM² 工具工作台。SRM² 工具工作台独立于 Eclipse 工作台，在没有破坏 Eclipse 工作台的透视图、编辑器和视图布局的前提下，可便捷地进行功能交互操作。工具的主窗口面板分成 4 个区域：(1)AADL 文本显示域：主要是显示打开的 AADL 文本程序以及显示打开的动态侦测信息文件；(2)蓝图表域：主要负责描述静态蓝图表；(3)蓝图树域：主要负责描述静态蓝图树；(4)模态迁移演示域：主要负责模态迁移的动画演示，其中，在该区域中拥有其独自的右键弹出菜单事件选项，"动态演示"表示开始动画，"暂停"选项可以暂停当前的动画演示，"继续"表示继续当前的动画，可以随时执行"重新演示"动作而重放动画演示过程，而且在动画演示过程中可以随时调整动画演示速度(慢、一般和快)。

在以上的设计中，保证了工具具有如下几个特点：(1)对 Eclipse 工具集是友好的。构架抽象工具的工作台没有破坏 Eclipse 工具透视图的编辑器和视图布局，只是添加了新的菜单项和新的工具栏按钮。(2)功能的扩展是独立的和方便的。由于构架抽象工具是独立的插件，不是通过对 Eclipse 工具集的修改创建的，因而，Eclipse 工具的版本变更不会引起构架抽象工具的失效。当然，也由于此原因，对其功能的扩展是方便的，不会担心因为功能的扩展导致与 Eclipse 工具的不兼容。

结束语　SRM² 工具首先从本地文件目录中导入 AADL 架构文件并对 AADL 架构文件进行扫描，扫描完成后将识别出来的系统、使命、功能和构件的模态名称、模态之间的迁移条件以及三者相互之间的从属关系等信息存储到预定义数据结构中。而后则根据这些信息加上预定义的树型结构生成静态蓝图树，静态蓝图树主要是从宏观上描述 AADL 程序的模态信息，要想了解 AADL 程序模态的详细信息，可以参考静态蓝图表；通过该工具的另一功能，根据 AADL 架构的模态信息加上制定的蓝图规划表来生成静态蓝图表。在完成了静态蓝图的生成之后，接下来就是生成动态蓝图，首先要导入动态侦测信息，然后结合静态蓝图的模态信息来生成动态蓝图，动态蓝图采用动画形式演示，主要是模拟并演示程序中的模态迁移过程。另外，工具的功能有些还需要进一步完善，对于代码中涉及到的除了进程和线程外的其他模态信息，如子程序、连接和总线等的模态信息，在该重构工具中给出了相应的扩展接口，以备后续功能扩展。

参 考 文 献

[1]　Gluch D P, Feiler P H, Hudak J J. The Architecture Analysis &

Design Language(AADL): An Introduction[M]. Carnegie Mellon University, 2006

[2] Flower M. 重构——改善既有代码的设计[M]. 北京: 中国电力出版社, 2003

[3] 肖汉. 基于 Java 平台的软件重用技术的研究与应用[D]. 北京: 北京邮电大学, 2005

[4] Whisnant K, Kalbarczyk Z T, Iyer R K. A system model for dynamically reconfigurable software[J]. IBM Systems Journal, 2003, 42(1)

[5] 张鹏, 姜昊, 许力, 等. Eclipse 插件开发[M]. 北京: 电子工业出版, 2008: 183-212

[6] Perry D E. Software engineering and software architecture [C]// Feng Yu-lin, ed. Proceedings of the International Conference on Software: Theory and Practice. Beijing: Electronic Industry

[7] Ronsse M, Maebe J, Bosschere K D. Software instrumentation using dynamic techniques[J]. Program Acceleration through Application and Architecture, 2002: 63-65

[8] 罗国庆, 等. VxWorks 与嵌入式软件开发[M]. 北京: 机械工业出版, 2003: 15-25

[9] 伽玛, 等. 设计模式——可复用面向对象软件的基础[M]. 李英军, 等译. 北京: 机械工业出版, 2005

(上接第 120 页)

中, 正常业务逻辑与异常处理逻辑相互混杂、交织, 给开发和维护工作带来了诸多不便。本文提出了基于 ECA 规则的语义流程异常处理机制, 构建了异常本体, 提出了 5 类异常处理动作和异常处理 ECA 规则。业务流程开发人员可以使用这些具有高层语义的 ECA 规则来定义异常处理逻辑。在运行时, 由异常处理中间件对截获的异常进行解析并指示流程执行引擎采取对应异常处理措施, 从而实现运行时异常处理。

本文方法没有考虑某些复杂的应用场景, 仅仅考虑了异常处理中间件对单一异常发生时的处理机制。进一步的工作是将研究重点放在发生异常时, 异常处理中间件生成动态植入流程执行引擎的可执行异常处理动作的生成算法上, 进一步提高应用的动态适应能力。

参 考 文 献

[1] Yang F Q, Lü J, Mei H. Technical framework for Internetware: An architecture centric approach[J]. Science in China Series F: Information Sciences, 2008, 51(6): 610-622

[2] Liu A, Li Q, Huang L S, et al. FACTS: A Framework for Fault-Tolerant Composition of Transactional Web Services[J]. IEEE Transactions on Services Computing, 2010, 3(1): 46-59

[3] Alves A, Arkin A, Askary S, et al. Web Services Business Process Execution Language Version 2.0[R]. OASIS Web Services Business Process Execution Language (WS-BPEL) TC, 2007

[4] 岳昆, 王晓玲, 周傲英. Web 服务核心支撑技术: 研究综述[J]. 软件学报, 2004, 15(3): 428-442

[5] Hepp M, Leymann F, Domingue J, et al. Semantic business process management: a vision towards using semantic Web services for business process management[C]// Proc. of IEEE International Conf. on e-Business Engineering (ICEBE 2005). Beijing: IEEE Computer Society Press, 2005: 535-540

[6] 曹虹华, 应时, 杜德慧, 等. 一种面向语义 Web 服务的软件设计语言和设计方法[J]. 电子学报, 2007, 35(12A): 129-135

[7] Chan K S M, Bishop J, Steyn J, et al. A fault taxonomy for Web service composition[C]// Proc. of the Int'l Workshops on Service-oriented Computing 2007. Heidelberg: Springer-Verlag, 2009: 363-375

[8] Pleisch S, Schiper A. Approaches to fault-tolerant and transactional mobile agent execution-An algorithmic view[J]. ACM Computing Surveys, 2004, 36(3): 219-262

[9] Casati F, Sayal M, Shan M-C. Developing E-Services for Composing E-Services[C]// Proc. of International Conf. on Advanced Information Systems Engineering. Heidelberg: Springer-Verlag, 2001: 171-186

[10] Luo Z, Sheth A P, Kochut K, et al. Exception Handling for Conflict Resolution in Cross-organizational Workflows[J]. Distributed and Parallel Databases, 2003, 13(3): 271-306

[11] Cardoso J, Sheth A, Miller J, et al. Quality of Service for Workflows and Web Service Process[J]. Journal of Web Semantics, 2004, 1(3): 281-308

[12] Dobson G. Using WS-BPEL to implement software fault tolerance for Web services[C]// Proc. of the 32nd EUROMICRO Conf. on Software Engineering and Advanced Applications. Los Alamitos: IEEE Computer Society Press, 2006: 126-133

[13] Zeng L, Lei H, Jeng J-J, et al. Policy-driven Exception-management for Composite Web Services[C]// Proc. 7th IEEE International Conf. on E-Commerce Technology (CEC2005). Munich: IEEE Computer Society Press, 2005: 355-363

[14] Erradi A, Maheshwari P, Tosic V. Recovery Policies for Enhancing Web Services Reliability[C]// Proc. 4th International Conference on Web Services (ICWS'06). Chicago: IEEE Computer Society Press, 2006: 189-196

[15] Chiu D K W, Li Q, Karlapalem K. A Meta Modeling Approach for Workflow Management System Supporting Exception Handling[J]. Information Systems, 1999, 24(2): 159-184

[16] Wiesner K, Vaculin R, Kollingbaum M, et al. Recovery Mechanisms for Semantic Web Services[C]// Proc. 8th IFIP WG 6.1 International Conf. on Distributed Applications and Interoperable Systems (DAIS2008). Oslo: Springer-Verlag, 2008: 100-105

[17] Vaculin R, Wiesner K, Sycara K. Exception Handling and Recovery of Semantic Web Services[C]// Proc. of 4th IEEE International Conf. on Networking and Services (ICNS 2008). Gosier: IEEE Computer Society Press, 2008: 217-222

[18] Men P, Duan Z H, Yu B. Utilizing Fuzzy Petri Net for Choreography Based Semantic Web Services Discovery[J]. Lecture Notes in Computer Science, 2007, 4546: 362-380

[19] Cao H H, Ying S, Du DH, et al. Orchestrating Semantic Web Services with Semantic Programming Language[C]// Ceballos S, ed. Proc. IEEE International Workshop on Semantic Computing and Systems (WSCS'08). 2008: 101-106

[20] 解丹, 应时, 曹虹华, 等. 基于语义的服务资源描述模型 RDF4S[J]. 武汉大学学报: 理学版, 2008, 54(1): 71-76

[21] 曾志浩, 应时, 曹虹华. 基于 RDF4S 语义服务描述模型的服务资源搜索框架[J]. 计算机科学, 2008, 41(11): 258-262