

Web应用中UML到Event-B的系统性转换方法

智玉华, 邹盛荣, 刘晓莹, 耿雪

(扬州大学 信息工程学院, 江苏 扬州 225009)

摘要: 在面向对象的软件开发过程中, UML被广泛应用于建模, 但因其语义的不精确性使得模型难以进行形式化验证。同时, Event-B作为一种基于数学逻辑的形式化方法, 尽管精确但难以理解。为了充分发挥两种方法的优势, 提出一种将UML与Event-B形式化方法相结合的解决方案, 针对中型Web应用系统, 提出了用例图、类图、顺序图、组件图和部署图5种UML图到Event-B的转换方法。通过将其应用于实际Web应用系统, 并利用Rodin平台的证明器解除模型中的证明义务, 验证了这一转换方法的可行性和有效性。该方法的应用, 不仅提高了UML的精确性和系统的可靠性, 减少了故障风险, 还增强了形式化方法的可理解性, 有助于其推广和应用。

关键词: 统一建模语言; 形式化方法; Event-B; Web应用

DOI: 10.11907/rjdk.251018

中图分类号: TP311

文献标识码: A

开放科学(资源服务)标识码(OSID):

文章编号: 1672-7800(2025)008-0112-09



Systematic Transformation Method from UML to Event-B in Web Application

ZHI Yuhua, ZOU Shengrong, LIU Xiaoying, GENG Xue

(School of Information Engineering, Yangzhou University, Yangzhou 225009, China)

Abstract: In the process of object-oriented software development, UML is widely used in modeling, but because of its semantic imprecision, it is difficult to verify the model formally. At the same time, as a formal method based on mathematical logic, Event-B is accurate but difficult to understand. In order to give full play to the advantages of the two methods, a solution combining UML and Event-B formal methods is proposed. For medium-sized Web application systems, five UML diagrams to Event-B conversion methods are proposed, including use case diagram, class diagram, sequence diagram, component diagram and deployment diagram. By applying these methods to the actual Web application system and using the prover of Rodin platform to release the proof obligation in the model, the feasibility and effectiveness of this transformation method are verified. The application of this method not only improves the accuracy of UML and the reliability of the system, reduces the risk of failure, but also enhances the comprehensibility of the formal method, which is helpful to its promotion and application.

Key Words: UML; formal method; Event-B; Web application

0 引言

统一建模语言(Unified Modeling Language, UML)是一个支持模型化和软件系统开发的图形化语言, 为软件开发的所有阶段提供模型化和可视化支持, 包括由需求分析到规格、构造和配置^[1]。一个完善的系统开发过程须从系统规范开始, 并严格依据规范进行开发。需求规范中隐含的问题会在开发过程中产生重大影响。然而, UML作为一种

半形式化的建模语言, 存在不精确性, 难以进行形式化验证^[2]。目前UML已提出了14种图, 但尚未明确规定哪些图适用于Web应用系统开发。设计Web应用系统时可能出现图表重复或缺少关联, 某些重要信息在不同图中缺乏连接或未能形成完整的系统视图, 导致设计不完善或理解困难。确保各种图之间的连贯性和完整性对Web应用系统有效设计和开发至关重要^[2]。形式化方法是基于数学的技术, 用于高安全性系统的验证, 消除规范歧义和不协调, 确保系统需求的理解、建模和分析完整^[3]。Event-B是

收稿日期: 2025-01-07

扫描二维码阅读全文:



作者简介: 智玉华(2000-), 女, 扬州大学信息工程学院硕士研究生, 研究方向为形式化方法; 邹盛荣(1968-), 男, 博士, 扬州大学信息工程学院副教授、硕士生导师, 研究方向为形式化方法; 刘晓莹(1999-), 女, 扬州大学信息工程学院硕士研究生, 研究方向为形式化方法; 耿雪(1999-), 女, 扬州大学信息工程学院硕士研究生, 研究方向为形式化方法。本文通讯作者: 邹盛荣。

一种流行且易于使用的形式化方法,具有精确的语义,是最新的B方法,但其严谨的数学逻辑使许多软件需求分析人员难以理解和应用^[4]。相较之下,UML虽然直观易懂,但表达能力不足。因此,将UML与Event-B相结合成为研究重点。这一结合可以利用UML的可视化特点捕捉需求,并通过Event-B的形式化验证提升软件系统的可靠性。将UML和Event-B进行结合,不仅增强了形式化方法的可理解性和可读性,还降低了理解门槛。这使得形式化方法更加易于被广泛接受和应用,从而推动其在软件开发过程中的使用,并促进高质量软件的开发。

1 相关研究

UML与形式化方法的结合已有一些研究。例如,文献[5]提出了一种将UML活动图转换为FoCaLiZe形式公式的方法,并利用FoCaLiZe进行了形式化建模和验证。FoCaLiZe是一种基于证明的形式语言,但其只关注简单的UML活动图,行为表达式仅限于类操作调用,未在实际应用中进行测试。文献[6]提出为保证UML类图的一致性,将形式化方法与UML类图结合起来对类图精化过程进行分析验证。文献[7]提出了UML顺序图构建形式化语义,为基于UML模型的可靠性分析和验证提供了语义支持。通过用例图、类图、顺序图和状态图,将UML系统性地转换为Event-B形式化方法。该方法在保留UML直观性的同时,引入了形式化验证。这4种图能够全面且清晰地表达中型系统的结构和行为,为模型的形式化验证提供基础,确保转换过程中的一致性和准确性。这种方法适用于中型系统的建模需求,兼具表达简洁性和形式化分析的优势。文献[8]探讨了系统需求分析阶段的形式化分析方法,通过模型检测来验证UML规范的约束和系统安全属性,但并不能完整发现UML语义的错误和系统的部分关键缺陷。

现有UML 14种图的零散转换方法在实际应用中存在转换不一致和冲突,影响模型的可理解性,也降低了开发效率^[9]。因此,有必要进行系统化研究,以构建完整的转换方法。通过明确UML图与Event-B元素的映射关系,建立统一的转换规则,可以减少转换中的错误,提升模型的质量与实用性。这将为促进形式化方法在软件开发中的应用创造有利条件,使UML与Event-B的结合更加紧密。

2 系统性转换方法

一般的Web应用系统通常被归类为中型系统,代码量5 000~50 000行^[7]。这类系统使用UML的用例图、类图、顺序图、组件图和部署图5种图形,可以有效地进行系统建模并清晰地表达系统的结构和行为。

在软件需求分析阶段,用例图用于抽象描述系统功

能,重点建模用户与系统的交互及功能实现,它为系统的功能实现提供了高层次的框架^[10]。类图为系统的静态结构建模,详细描述了系统中的类、属性、方法及其相互关系^[11]。在Event-B中,类的属性可以转化为变量,而类之间的关系(如继承、关联等)可以通过集合和集合之间的关系来表示。Event-B使得这些静态结构可以通过形式化的集合和变量进行建模,确保模型的一致性和正确性。顺序图则进一步细化了系统中的动态行为,展示了不同对象之间的交互和消息传递过程,帮助开发团队理解系统中各个环节协同工作的流程^[12]。在设计架构时,组件图和部署图进一步对系统的物理架构进行建模,帮助开发者理解系统组件如何组织和部署在硬件设施上^[13,14]。组件图描绘了系统的模块化结构,展示了不同模块之间的依赖关系和协作方式,确保各个功能模块可以有效协同工作。部署图则描述了系统的硬件架构和软件组件的部署方式,帮助开发和运维团队理解系统的物理资源配置,以及组件如何在服务器或云环境中部署运行,确保系统的可扩展性和高可用性。

通过这5种UML图形,可以全面而系统地表达Web应用系统的需求、设计和实现过程。为了系统地将UML模型转换为Event-B形式化方法,本文基于用例图、类图、顺序图、组件图和部署图,提出了一种系统化的转换策略。该策略明确了UML各元素与Event-B元素的对应关系,并通过表格清晰展示了这些映射,帮助开发者在转换过程中保持模型的一致性和准确性。本文详细描述了从UML到Event-B的具体转换步骤,包括从用例图的功能抽象到类图的静态结构、顺序图的行为交互等。通过这一系统化方法,开发者能够更有效地完成UML到Event-B的转换,为系统的形式化验证提供坚实基础。

2.1 用例图

用例图到Event-B形式化方法的转换已被提出,文献[15]详细介绍了具体步骤。用例图中的参与者和用例转化为Event-B中的常量,各种关系通过Event-B事件表示,确保模型一致性并增强实用性。具体而言,关联关系和扩展关系会被转换为Event-B抽象机器中的事件,以体现系统的基本交互和扩展逻辑,而泛化关系和包含关系则依赖于Event-B的精化机制实现。当泛化关系涉及用例图中的参与者时,将在扩展的上下文中添加相应的公理,以明确参与者之间的泛化关系,确保模型的准确性和一致性。该方法能够准确表达参与者之间的层次结构,进一步保障了模型在各个精化层次上的一致性和逻辑完整性。通过这种系统化的转换策略,用例图中的复杂关系可以被精确地映射到Event-B模型中,从而为系统的形式化验证和分析提供可靠的基础。当泛化关系涉及用例时,可以通过在精化的机器中引入事件来表示用例之间的泛化关系。用例图到Event-B方法的转换规则,如表1所示。

2.2 类图

在类图到Event-B的转换过程中,类图中的类、属性和

Table 1 Use case diagram conversion rules

表 1 用例图转换规则

用例图	Event-B	转换解释
参与者	常量	在上下文中将参与者转换成常量
用例	常量	在上下文中将用例转换成常量 泛化关系通过精化机制实现:参与者的泛化通过扩展上下
泛化	事件/公理	文添加公理表示;用例的泛化通过在精化的机器中添加事件表示
关联	事件	关联关系在抽象的机器中转换成事件,事件名称为用例的名称。在事件中添加动作语句表示关联关系的实现
扩展	事件	扩展关系在抽象的机器中转换成事件,事件名称为扩展用例的名称。在事件中添加动作语句表示扩展关系的实现
包含	事件	包含关系通过 Event-B 的精化机制实现。一方面,可以精化抽象机器中的抽象事件。另一方面,也可以在精化的机器中创建新的事件来精化抽象机器中的事件。在事件中
		添加参数,通过参数的传递对表示包含关系的变量赋值

关系会转换成 Event-B 模型中的常量、载波集等元素。在这个转换过程中,可以为这些映射后的元素添加不变量,以对其行为和状态进行约束,从而确保模型的准确性和一致性^[16]。此外,类图中的方法会转为 Event-B 中的事件,动作模块中可添加语句详细描述方法的执行过程,增强模型的表达能力与准确性。通过这种方式,类图的行为在 Event-B 模型中得到了完整表达,从而准确保留了原有类图的语义,并确保了形式化模型的正确实现。类图到 Event-B 的转换规则,如表 2 所示。

Table 2 Class graph conversion rules

表 2 类图转换规则

类图	Event-B	转换解释
类	变量/常量/载波集	类可以转换成 Event-B 中的变量、常量或者是载波集等元素
属性	常量	类中的属性可以转换成 Event-B 中的常量或者是变量
方法	事件/公理	类图中的方法转换成 Event-B 中的事件。在转换的事件中,会自动的生成一个参数,默认名称为 this_类名,守卫条件中声明该参数的类型为该方法所属的类的类型。事件的动作模块中可以添加语句表示方法的具体执行过程
关系	常量/变量	关系可以转换成 Event-B 中的常量或者是变量。当关系转换成常量时,上下文中会自动的根据关系两端的多重性声明常量的类型。当关系转换成变量时,机器中会根据关系两端的多重性添加不变量声明关系的类型

2.3 顺序图

已有研究提出了从顺序图到 Event-B 形式化方法的一些转换方法^[17]。将顺序图转换为 Event-B 方法时,通信对象和消息转为常量,消息传递顺序通过 Event-B 机器中的变量控制,从而清晰表示对象之间的交互和流程。在 Event-B 模型中,消息传递过程可通过事件表示。守卫条件判断消息的传递顺序,使用变量确保消息按序发送。动作模块中明确源对象、目标对象和待传递的消息,并更新目标对象的状态以表示消息接收成功。同时,将消息顺序的值加一,标志着该消息的传递完成。通过这种形式化方法,可以确保消息传递的顺序性和正确性。顺序图到 Event-B 方法的转换规则,如表 3 所示。

Table 3 Sequence diagram conversion rules

表 3 顺序图转换规则

顺序图	Event-B	转换解释
通信对象	常量	通信对象转换成常量,在表示通信对象的集合中分区
消息	常量	消息转换成常量,在上下文中表示消息的集合中分区 顺序图中消息的传递顺序通过 Event-B 机器中的变量值
传递顺序	变量	决定,在事件中对表示消息传递顺序的变量值进行判断和更改 通信对象之间消息的传递过程转换成事件。事件的守卫条件(guards)判断消息的传递顺序,动作中添加语句
消息传递	事件	表示消息传递的源对象、目标对象和待传递的消息。在事件的动作模块中,通过对表示消息传递顺序的变量进行自增,标志着该条消息的传递结束。这不仅记录已传递消息数量,还确保后续消息按正确顺序进行传递

2.4 组件图

基于已经提出的方法,组件图在转换成 Event-B 时,组件图中的组件被转换成的 Event-B 中的常量、变量和载波集,组件之间信息传递所需的接口用 Event-B 中的事件来表示。组件之间的关系通过精化来实现,在事件中添加动作语句来表达。组件图到 Event-B 方法的转换规则,如表 4 所示。

Table 4 Component diagram conversion rules

表 4 组件图转换规则

组件图	Event-B	转换解释
组件	常量/变量/载波集	组件可以转换成 Event-B 中的变量、常量或者是载波集等元素。组件内部的属性和方法可以映射到 Event-B 中的变量和常量。每个组件的状态和数据在 Event-B 模型中以变量的形式呈现,表达组件的行为和相互关系
接口	事件	接口转化为 Event-B 中的事件,每个事件定义了组件之间的交互或者功能,描述了状态的变化以及组件之间如何通过事件进行协作和通讯。在 Event-B 中,事件包括前置条件(Guards)和后置条件(Actions),定义了接口的具体功能
关系	Sees/Refines	UML 中组件之间的关系(如依赖、继承或实现)可以在 Event-B 中通过 Sees 或 Refines 来表达。Sees 表示该模块需要访问另一个模块的某些信息。Refines 则表示一个模块在抽象层次上的细化,表示某个抽象模块被具体实现模块所替代,通常用于继承或依赖关系
数据传输	常量/变量/载波集	模块之间传递的数据抽象为变量,通过定义事件的前置条件和后置条件,事件操作确保数据在模块之间的一致性。组件间传输的数据可以是系统状态的一个部分,并通过变量在不同组件之间传递

2.4.1 模块与关系

在 Event-B 模型中,可以将组件图中的参与者和组件转换为常量,以便在上下文中进行定义和使用。首先,定

义常量 COMPONENTS 表示组件。组件之间的关系用集合表示, C1、C2 表示组件, RELATION 表示组件之间的关系。事件 Activate_C1 表示 C1 依赖于 C2。交互关系可以细化为事件间的依赖顺序, 同步关系表示为某事件必须与另一事件同步执行, 异步关系松耦合部分可以描述为中介事件。具体转换方法如图 1 所示。

```
CONTEXT C0
SETS COMPONENTS
CONSTANTS RELATIONAXIOMS
  axm1: finite(COMPONENTS)
  axm2: RELATION  $\subseteq$  COMPONENTS  $\times$  COMPONENTS
END

VARIABLES state
INVARIANTS
  inv1: state  $\in$  COMPONENTS  $\rightarrow$  {idle, active, finished}
EVENT Activate_C1
WHEN
  grd1: state(C2) = finished
  grd2: state(C1) = idle
THEN
  act1: state(C1) := active
END
```

Fig. 1 Component and interaction transformation abstract modeling

图 1 组件和交互转换抽象建模

2.4.2 依赖关系

图 2 定义了两个组件 C1 和 C2, 并表示它们之间存在一种依赖关系: C2 必须先完成任务, 然后才能启动 C1。在 Event-B 中通过上下文(Context)和机器(Machine)来定义系统的静态和动态行为。在 Context 中定义 COPO-NENTS 表示系统中的所有组件, 定义常量 C1 和 C2 表示具体的 2 个组件。依赖关系的抽象建模示意图, 如图 2 所示。



Fig. 2 Abstract modeling of dependencies between components

图 2 组件间依赖关系抽象建模

2.5 部署图

部署图到 Event-B 的转换目前只在少数 UML 到 Event-B 的系统转换中提及, 并未有更加详尽的转换。基于其他已转换的 UML 图, 部署图在转换成 Event-B 时, 硬件节点直接转换为 Event-B 中的常量、变量或者载波集。

在 Event-B 中, 部署在节点里面的组件转换成 Event-B 模型中的常量或者载波集, 静态展示组件的状态。同时, 通信路径在抽象机器中转换为事件, 并加入动作语句表示其实现, 以确保系统性能与功能达到预期。部署图到 Event-B 方法的转换规则, 如表 5 所示。

Table 5 Deployment graph conversion rules

表 5 部署图转换规则		
部署图	Event-B	转换解释
节点	常量/变量/载波集	硬件节点可直接转换为 Event-B 中的常量、变量或者载波集。通过给变量赋值来表示节点状态
组件	常量/载波集	部署在节点里面的组件转换成 Event-B 模型中的常量或者载波集, 静态展示组件的状态
通信路径	事件	通信路径在抽象的机器中转换成事件。在事件中添加动作语句表示该通信的实现, 以及组件节点状态的转换

2.5.1 节点转换

为了将网站管理系统所需的硬件资源(如服务器等设备)和部署在这些硬件节点上的软件资源组件转换为 Event-B 中的常量和载体集, 需要在上下文中定义节点和组件的初始状态以及它们之间的静态关系, 特别是明确哪些组件部署在哪些节点上。软件组件和硬件节点转换, 如图 3 所示。

```
context DeploymentContext
Sets
  NODES
  COMPONENTS
Constants
  DEPLOYED
  COMMUNICATION
Axioms
  axm1: partition(NODES, {Node1, Node2})
  axm2: partition(COMPONENTS, {Comp1, Comp2})
  axm3: DEPLOYED = {(Comp1, Node1), (Comp2, Node2)}
  axm4: COMMUNICATION  $\subseteq$  NODES  $\times$  NODES
end
```

Fig. 3 Conversion of software components and hardware nodes

图 3 软件组件和硬件节点转换

2.5.2 节点通信

机器定义系统的动态行为, 以组件加载到节点和通信路径激活为例。定义两个变量 deployedComponents 和 activeCommunication, 前者动态跟踪已经部署的组件, 后者动态跟踪已经激活的通信路径。inv1 确保已部署的组件必须符合组件和节点的定义, inv2 确保激活的通信路径必须是有效的节点关系。组件间实现通信初始化定义, 如图 4 所示。

事件 Event_Load 模拟一个组件部署到一个节点的行为。grd1、grd2、grd3 是守卫条件, 确保组件和节点合法, 并且部署关系正确。act1 更新 deployedComponents 集合。组件部署转换, 如图 5 所示。

```

machine DeploymentMachine
Variables
  deployedComponents
  activeCommunication
invariants
  inv1: deployedComponents  $\subseteq$  COMPONENTS  $\times$  NODES
  inv2: activeCommunication  $\subseteq$  NODES  $\times$  NODES

```

Fig. 4 Preliminary conversion of component communication

图 4 组件通信初步转换

```

Event_Load
any comp node
where
  grd1: comp  $\in$  COMPONENTS
  grd2: node  $\in$  NODES
  grd3: (comp, node)  $\in$  DEPLOYED
then
  act1: deployedComponents := deployedComponents  $\cup$  {(comp, node)}
end
end

```

Fig. 5 Component deployment transformation

图 5 组件部署转换

事件 Event_ActivateCommunication 模拟 2 个节点之间的通信路径激活。grd1、grd2、grd3 是守卫条件,确保通信路径合法。act1 更新 activeCommunication 集合。路径激活方式如图 6 所示。

```

Event_ActivateCommunication
any node1 node2
where
  grd1: node1  $\in$  NODES // 第一个节点有效
  grd2: node2  $\in$  NODES // 第二个节点有效
  grd3: (node1, node2)  $\in$  COMMUNICATION // 通信路径有效
Then
  // 激活通信
  act1: activeCommunication := activeCommunication  $\cup$  {(node1, node2)}
end

```

Fig. 6 Path activation

图 6 路径激活

3 在 Web 管理系统中的应用

网站管理系统是基于 Web 技术开发的系统,通常是指能够通过网页浏览器进行访问和操作的系统。Web 系统具有跨平台性、易扩展性和易维护性等特点,适用于各种应用场景,包括但不限于网站、电子商务系统及在线招聘平台等。在开发 Web 系统时,需要考虑数据库设计与管理、系统架构设计、安全性和性能优化等方面。

本文将以上网招聘系统为例进行应用,来验证本文所提出的系统化转换方法。抽象的招聘系统中主要包含用户浏览和筛选职位、用户投递简历以及招聘者查看简历等功能。表 6 概述了该招聘网站的主要需求,其中 Req 表示模型中的需求标签。

在精化后的网上招聘系统模型中,首先,应聘者输入

Table 6 Total requirements

表 6 总需求

抽象模型	需求叙述
Req1.1	用户登录注册,并进行身份选择
Req1.2	应聘者浏览相关岗位,系统根据应聘者身份提供相关岗位
Req1.3	应聘者申请岗位并根据招聘需求投递简历
Req1.4	招聘者查看申请岗位应聘者上传的简历

岗位需求,并由浏览器将这些需求传递给服务器;其次,服务器处理后反馈相应的信息给浏览器;然后,当所有功能正常运行后,将各个功能模块整合为组件,并通过组件接口实现组件之间的通信;最后,所有组件被整合到用户端服务器中,与其他软硬件结合,共同构成该网上招聘系统的硬件节点。各功能精化的网上招聘系统模型的具体需求,如表 7 所示。

Table 7 Specific requirements

表 7 具体要求

抽象模型	需求叙述
Req2.1	应聘者输入岗位需求,浏览器获取并向服务器传递信息
Req2.2	服务器响应需求,将所需信息反馈给浏览器
Req2.3	各类功能整合成组件,创建组件接口,实现组件间通信
Req2.4	将所有组件整合成用户端服务器,并创建服务器接口
Req2.5	服务器通过接口与其他已创建接口的软硬件如数据库等节点实现信息交流
Req2.6	服务器、数据库、防火墙等硬件之间实现通信

3.1 用例图

依据网上招聘系统的需求描述,得出网上招聘系统的用例图如图 7 所示。

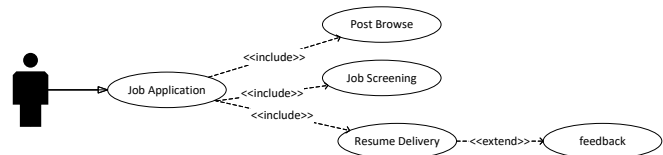


Fig. 7 Job application use case diagram

图 7 岗位应聘用例图

根据上述 UML 用例图到 Event-B 的转换方法,本文将用例图中的参与者和用例都转换成常量,各种关系转换成机器中的事件。抽象的机器中创建了 BOOL 类型的变量 userReq,表示用户请求岗位浏览和应聘的系统响应状态。抽象机器 RS_m0 中创建事件 Job_Application,表示用例图中 User 参与者与 JobApplication 用例之间的关联关系。事件中添加的 BOOL 类型的参数 anyStatement,表示用户请求使用岗位应聘的系统响应状态。事件中的语句 act1-act3 对表示关联关系的变量分别赋值,语句 act4 将参数 anyStatement 的值赋到变量 userReq,通过 anyStatement 的取值来决定变量 userReq 的值。事件 Job_Application 的具体内容如图 8 所示。

3.2 类图

网站管理系统的类图如图 9 所示。根据系统控制需求的描述,得到 ApplicationInformation、ApplicationInforma-

```

Job_Application
ANY
  anyStatement
WHERE
  grd1 : anyStatement ∈ BOOL
THEN
  act1 : actor := user
  act2 : usecase := post
  act3 : association :∈ {user} → {post}
  act4 : userReq := anyStatement
END

```

Fig. 8 Event Job_Application implementation

图 8 事件 Job_Application 实现

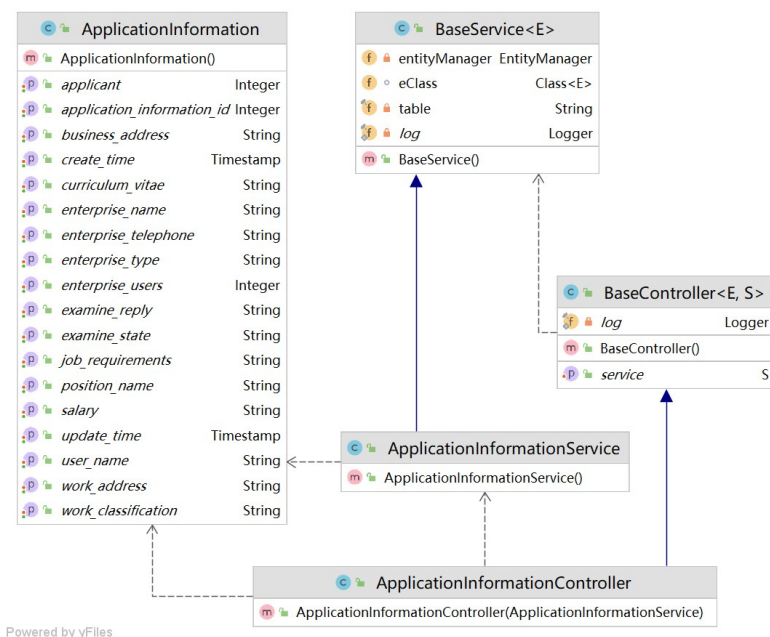


Fig. 9 Job application class diagram

图 9 岗位应聘类图

```

ApplicationInformationController:
ANY
  anyStatement_01
WHERE
  grd1: anyStatement_01 ∈ BOOL
  grd2: Post_statement_01 = ATV ∧ Post_statement_01 = usable
THEN
  act1: userReq_02 := anyStatement_01
  act2: User_01 := user_01
  act3: post := post_01
  act4: resume_001 := resume_01 H :∈ {resume_01} → {post_01}
END

```

Fig. 10 Browsing and applying for job conversion

图 10 浏览并应聘岗位转换

3.3 顺序图

招聘系统对于用户投递简历请求的一次响应是很复杂的执行过程。在系统响应请求的过程中,需要浏览器、

tionController、ApplicationInformationService,并将这些类转换成 Event-B 中的载波集。ApplicationInformation 里的 applicant、enterpriseUsers 等属性转换成抽象机器中的变量,分别表示系统内部的状态、系统对申请者请求的响应状态和岗位负责人对系统信息回复的状态。类中实现系统接受请求、系统响应请求等方法转换成 Event-B 中的抽象事件。

类图中的 ApplicationInformationController 方法表示用户请求浏览岗位并应聘该岗位,该方法转换成 Event-B 机器中的事件。在转换而来的 ApplicationInformationController 事件中,需要添加一个参数 anyStatement 表示用户请求的最终状态,如图 10 所示。

服务器、数据库等交互控制才可以完成。如图 11 所示,该顺序图详细地描述了用例图中的 Job_Application 用例的具体实现过程。用户向岗位投递简历后,浏览器会将该简历数据封装后传送给服务器进行处理。如果该简历不符合规定,则会向浏览器反馈失败信息;如果简历符合规定,服务器反馈上传成功信息并传递给数据库进行存储。

顺序图中 Job_Application 的消息传递过程被转换为事件,如图 12 所示。守卫条件 grd1 判断消息的传递顺序是否为 3。在事件的动作中表示消息的传递过程,事件中的动作语句 act1-act3 对消息传递的源对象、目标对象和待传递的消息赋值。动作 act4 通过将消息传递顺序的变量 NumberofResume 值增加 1,来表示当前的消息传递结束。Job_Application 事件的具体内容如图 12 所示。

3.4 组件图

建筑行业招聘系统的组件图如图 13 所示。根据前文所述将组件图转换成 Event-B 的方法,本研究将组件图中

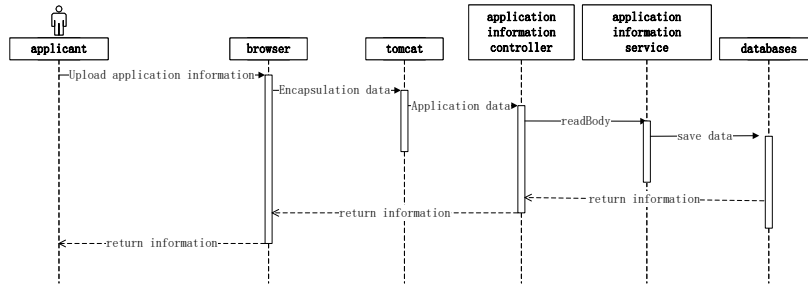


Fig. 11 Job application sequence diagram

图 11 岗位应聘顺序图

```

Job_Application
ANY
anyStatement
WHERE
grd1 : anyStatement ∈ BOOL
THEN
act1 : actor := user
act2 : usecase := post
act3 : association : ∈ {user} → {post}
act4 : userReq := anyStatement
END

```

Fig. 12 Information transmission of job application

图 12 岗位应聘信息传递

的组件和接口转换成载波集和常量,组件之间的关系由事件表示。

转换的过程中以数据封装为例,首先,将封装模块组件转换成常量,用户通过浏览器将简历传递给服务器并进

入数据封装阶段。其次,设置初始数据在模块间传递顺序为 1,守卫条件 grd1 判断数据传递顺序,grd2 判断该组件是可行的,grd3 判断组件接口是否正确。如果所有条件都得到满足,动作语句 act1-act2 分别对数据源和组件进行赋值。通过参数 anyReq 值,决定变量 Req 的值。最终,数据传递成功,数据传递到的组件增加 1。具体实现如图 14 所示。

```

ApplicationInformationController:
ANY
anyStatement_01
WHERE
grd1 : anyStatement_01 ∈ BOOL
grd2 : Post_statement_01 = ATV ∧ Post_statement_01 = usable
THEN
act1 : userReq_02 := anyStatement_01
act2 : User_01 := user_01
act3 : post := post_01
act4 : resume_001 := resume_01 H : ∈ {resume_01} → {post_01}
END

```

Fig. 14 Package of components

图 14 组件封装

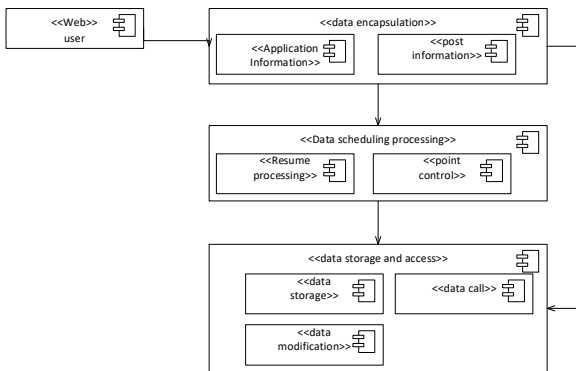


Fig. 13 Job application component diagram

图 13 岗位应聘组件图

3.5 部署图

网上招聘系统的物理部分部署图如图 15 所示。根据 Web 应用系统的需求,主要有用户浏览器、防火墙、服务器和数据库等物理部件。通过部署图,可以清晰地了解系统中各个组件的部署情况,快速定位问题,并采取相应的措施进行解决。只有这些物理部件无故障,且数据能够在部件之间正常传输,网上招聘系统才能顺利运行并被正常使用。

本文将这些部件转换为 Event-B 中的载体集,用 BOOL 类型的变量表示其状态,并将它们之间的关系表示

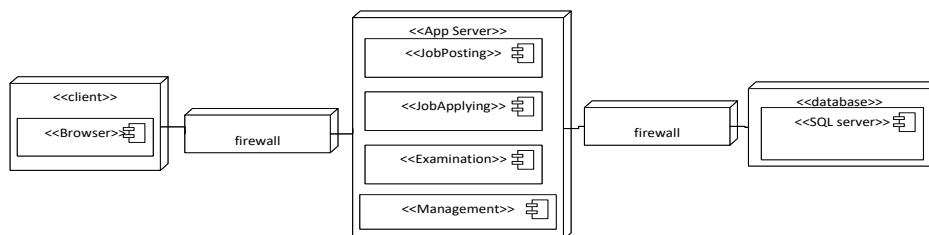


Fig. 15 Online recruitment website deployment diagram

图 15 网上招聘网站部署图

为 Event-B 中的事件, 同时在精化的事件中添加表示物理部件状态的变量 `userbrowserState`、`firewallState`、`SQLState` 和 `appserverState`。在 INVARIANTS 中约束各部件之间的关系, 在事件 `SystemDeployment` 中, 守卫条件确定该系统能够顺利进行时各部件的状态, 动作 `act1` 表示用户请求的状态。事件 `SystemDeployment` 的具体内容如图 16 所示。

```
MACHINEDeploymentDiagram_0
VARIABLES
user
firewall
userbrowser
appServer
SystemDeployment:
ANY
anyState
WHERE
grd1:firewallState = availability not theorem
grd4:anyState ∈ BOOL not theorem
grd2:appserverState = usable not theorem
grd3:SQLState = writable not theorem
THEN
act1:userReq := anyState
END
```

Fig. 16 Node deployment
图 16 节点部署

4 模型证明

本文使用了上述所提到的 UML 到 Event-B 的系统转换方法, 并在 Rodin 平台中为网上招聘系统建模。Rodin 平台中含有丰富的证明器, 可以对所创建的网上招聘模型进行验证。本文采用 Rodin 平台中自带的证明器和 Atelier B 等提供的外部证明器, 对模型中产生的证明义务进行解除。

将网上招聘系统的 UML 图转换为 Event-B 形式化方法后, 模型中生成了系列需要验证和证明的义务, 如图 17 所示。图 17(a)展示了抽象机器 `m0` 中产生的 12 条证明义务, 这些义务是对系统设计的关键属性进行验证的要求, 确保了模型在高层次上的一致性和正确性。抽象机器的证明义务主要与系统的基本性质和定义相关, 为后续的精化提供了坚实的基础。图 17(b)则显示了在精化机器 `m1` 中产生的 9 条证明义务, 这些义务关注于更具体的实现细节和功能验证。这些证明义务的有效解除是通过形式化证明和交互式证明的方法来实现, 确保了模型在不同层次上的正确性和可行性。通过这一过程, 不仅增强了模型的可靠性, 还为实现系统设计目标提供了必要的保障, 从而为网上招聘系统的成功实施奠定了基础。

5 结语

本文提出了 Web 应用中 UML 到 Event-B 的系统性转

Element Name	Tot.	Aut.	M...	Rev.	Un...
RS_m0	12	12	0	0	0
INITIALISATI...	7	7	0	0	0
inv1	0	0	0	0	0
inv2	0	0	0	0	0
inv3	1	1	0	0	0
JobApplication	1	1	0	0	0
inv4	0	0	0	0	0
inv5	1	1	0	0	0
ResumeDeliv...	1	1	0	0	0
inv6	0	0	0	0	0
inv7	1	1	0	0	0
inv8	0	0	0	0	0
inv9	0	0	0	0	0
JobPosting	1	1	0	0	0
inv10	1	1	0	0	0
inv11	0	0	0	0	0
inv12	3	3	0	0	0
inv13	2	2	0	0	0
inv14	0	0	0	0	0
SystemFaults	1	1	0	0	0
inv15	0	0	0	0	0
inv16	0	0	0	0	0
inv17	3	3	0	0	0
SystemMaint...	1	1	0	0	0

Element Name	Tot.	Aut.	M...	Rev.	Un...
RS_m1	9	8	0	0	1
INITIALISATI...	2	2	0	0	0
JobApplication	0	0	0	0	0
ResumeDeliv...	0	0	0	0	0
JobPosting	0	0	0	0	0
SystemFaults	2	2	0	0	0
SystemMaint...	2	2	0	0	0
inv1	0	0	0	0	0
inv2	0	0	0	0	0
inv3	0	0	0	0	0
inv4	4	4	0	0	0
SystemDeplo...	1	1	0	0	0
AppserverSta...	2	1	0	0	1
inv5	4	3	0	0	1

(a) Formal verification of abstract machine (b) Refined machine formal verification
(a) 抽象机器形式化验证 (b) 精化机器形式化验证

Fig. 17 Formal verification of abstract machine and refined machine
图 17 抽象机器形式化验证和精化机器形式化验证

换方法, 系统的将半形式化的建模语言 UML 转换成 Event-B 形式化方法。同时, 将该系统的转换方法应用到网上招聘系统中, 基于 Rodin 平台为网上招聘系统建模, 对模型中产生的证明义务进行解除。

该系统的转换方法是将 UML 与 Event-B 形式化方法结合, 这种方法具有诸多显著优势。一方面, 通过将半形式化的建模语言 UML 进行精确化处理, 能够在软件设计的早期阶段对所建立的模型进行形式化验证。这种验证机制有助于识别潜在的问题和错误, 提高软件设计的可靠性和准确性, 从而降低后期修复成本, 确保更高质量的软件交付。这种方法对于软件从业人员尤为重要, 因为他们能够帮助他们在模型开发阶段迅速识别并解决问题; 对于安全关键型系统(如医疗、金融、航空等领域), Event-B 提供了强大的形式化验证功能, 能够对系统的安全性、可靠性和一致性进行严格验证, 从而确保系统在各种异常和错误情况下仍能可靠运行。另一方面, 结合 UML 和 Event-B 增强了形式化方法的可理解性和可读性, 降低了理解门槛。这使得形式化方法更加易于被广泛接受和应用, 从而推动其在软件开发过程中的使用。这种转换不仅为软件开发提供了更加严谨的验证手段, 还降低了形式化方法的学习门槛, 促进了高质量软件的开发。

本文采用了用例图、类图、顺序图、组件图和部署图, 系统地将 UML 转换为 Event-B 形式化方法。对于一般的中型 Web 系统而言, 这 5 种图已能够很好地进行表达。这些图形从不同维度全面地表达了 Web 应用系统的各个方面, 帮助开发团队在需求分析、设计、实现及部署过程中保持一致性、清晰度和高效性, 从而确保系统功能的实现、架构的合理性以及部署的可行性。

本文的不足之处在于, 讨论 UML 到 Event-B 转换时侧重于 Web 应用场景, 导致所提出的转换方法或模型在其他领域的适用性受到限制, 可重用性大大降低。此外, 本文所提出的 UML 到 Event-B 转换方法依赖大量手动过程

或复杂的人工干预,这种方法的可用性就会受到限制。特别是在大规模系统或多个项目中,重复手动操作可能导致效率低下,限制了本方法的广泛应用。未来将针对该转换方法在物联网系统等分布式系统中的可用性开展进一步研究。

参考文献:

- [1] QI Y. Design and realization of library management system based on UML and Java[J]. Electronic Technology and Software Engineering, 2020(20): 137-139.
齐燕. 基于UML和Java的图书管理系统的设计与实现[J]. 电子技术与软件工程, 2020(20): 137-139.
- [2] ZOU S R, ZHOU T, GU A H, et al. UML object-oriented requirements analysis and modeling tutorial [M]. Beijing: Science Press, 2018.
邹盛荣, 周塔, 顾爱华, 等. UML面向对象需求分析与建模教程[M]. 北京: 科学出版社, 2018.
- [3] SEIDEWITZ E. UML with meaning: executable modeling in foundational UML and the ALF action language[J]. Ada Letters, 2014, 34(3): 61-68.
- [4] GENG X, ZOU S R, YAO J Y, et al. Event-b modeling for use case diagrams and application in immune system model [C]//7th International Conference on Computational Intelligence and Applications, 2022: 6-10.
- [5] MESSAOUD A, RENAUD R, CHOUKRI-BEY B Y, et al. Formal modeling and verification of UML Activity Diagrams (UAD) with FoCaLiZe[J]. Journal of Systems Architecture, 2020, 114: 101911.
- [6] XIE J C, JIANG J M, CHEN H H, et al. Consistency analysis of UML class diagrams[J]. Software Guide, 2024, 23(2): 40-47.
谢嘉成, 蒋建民, 陈华豪, 等. UML类图的一致性分析[J]. 软件导刊, 2024, 23(2): 40-47.
- [7] GENG X, ZOU S R, LIU X Y, et al. A systematic conversion method from UML to Event-B[J]. Computer Technology and Development, 2023, 33(12): 113-120.
耿雪, 邹盛荣, 刘晓莹, 等. UML到Event-B的系统化转换方法[J]. 计算机技术与发展, 2023, 33(12): 113-120.
- [8] LIU X Y. A formal analysis method for system requirements based on UML [D]. Beijing: Beijing Jiaotong University, 2015.
刘心钰. 基于UML的系统需求形式化分析方法[D]. 北京: 北京交通大学, 2015.
- [9] ZOU S R, GENG X, YAO J Y, et al. Systematic transformation method from UML to Event-B [M]. California: Scientific Research Publishing, 2024.
- [10] ALTURAS B. Connection between UML use case diagrams and UML class diagrams: a matrix proposal [J]. International Journal of Computer Applications in Technology, 2023, 72(3): 161-168.
- [11] VIESCA A R, LAIL M A. Automated mitigation of frame problem in UML class diagram verification [C]//2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, 2023: 841-850.
- [12] HAMROUCHE H, CHAOUI A, MAZOUZI S. A graph transformation approach for modeling and verification of UML 2.0 sequence diagrams [J] Computing and Informatics, 2022, 41(5): 1284-1309.
- [13] GENG X, ZOU S R, YAO J Y. Systematic transformation method from UML to Event-B [C]//IEEE 22nd International Conference on Software Quality, 2022: 770-771.
- [14] BELLETTINI C, MARCHETTO A, TRENTINI A. WebUml: reverse engineering of web applications [C]//Italy: Proceedings of the 2004 ACM Symposium on Applied Computing, 2004.
- [15] SUN W X, ZHANG H, FU Y Z, et al. A method for the translation from UML into Event-B [C]//International Conference on Software Engineering and Service Science, 2016: 349-352.
- [16] HAN P, CHENGLIE D, LEI R, et al. Modelling the embedded control system using iUML-B pattern state machine [J]. Journal of Control Science and Engineering, 2018(1): 1468172.
- [17] YOUSRA B H, AHLEM B Y, LEILA J B A, et al. From sequence diagrams to Event-B: a specification and verification approach of flexible workflow applications of cloud services based on meta-model transformation. [C]//IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), 2017: 187-192.

(责任编辑: 陈维捷)