计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(9):12−23 [doi: 10.15888/j.cnki.csa.008086]

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

基于 Event-B 的软件工程形式化方法综述①

彭　寒[1], 张晓丽[1], 刘洲洲[1], 曹国震[1], 景月娟[1], 王　瑾[2], 李添锐[2]

[1](西安航空学院 计算机学院, 西安 710077)
[2](西安石油大学 计算机学院, 西安 710065)
通讯作者: 彭　寒, E-mail: hansbeng2016@gmail.com

摘　要: 在当今泛在计算和软件定义的大趋势下, 形式化方法逐步成为指导软件需求定义、分析软件设计方案、验证软件制品正确性的重要方法, 渗透到软件工程的全寿命周期. Event-B 作为一种“构造即正确”的方法, 为软件工程形式化方法的应用提供了支撑. 本文对现有的基于 Event-B 的软件工程形式化方法进行了分类阐述, 主要分为 Event-B 控制结构、面向对象的 Event-B、可重用的 Event-B 以及实时 Event-B 模型, 并对各种 Event-B 模型对软件开发全寿命周期的支持进行了汇总, 为软件工程形式化方法提供参考和借鉴.

关键词: 软件工程; 形式化方法; 面向对象; 可重用形式化模型

引用格式: 彭寒,张晓丽,刘洲洲,曹国震,景月娟,王瑾,李添锐.基于 Event-B 的软件工程形式化方法综述.计算机系统应用,2021,30(9):12–23. http://www.c-s-a.org.cn/1003-3254/8086.html

Overview on Formal Methods of Software Engineering Based on Event-B

PENG Han[1], ZHANG Xiao-Li[1], LIU Zhou-Zhou[1], CAO Guo-Zhen[1], JING Yue-Juan[1], WANG Jin[2], LI Tian-Rui[2]

[1](School of Computer Science, Xi'an Aeronautical University, Xi'an 710077, China)
[2](School of Computer Science, Xi'an Shiyou University, Xi'an 710065, China)

Abstract: In today's general trend for ubiquitous computing and software definition, formal methods have gradually become an important way to guide the definition of software requirements, analyze software design schemes, and verify the correctness of software products, which penetrates the entire life cycle of software engineering. Event-B, as a "correct by construction" method, supports the application of formal methods in software engineering. This paper classifies and expounds on the existing formal methods in software engineering based on Event-B, which are mainly divided into Event-B control structure, object-oriented Event-B, reusable Event-B, as well as real-time Event-B models. It also summarizes the support from various Event-B models for the whole life cycle of software development and provides references for the formal methods in software engineering.

Key words: software engineering; fomal method; object-oriented; reusable formal model

1　引言

随着物联网、云计算、信息物理融合系统时代的到来, “软件定义”已成为当前计算机科学以及软件学科的研究热点. 软件的泛在化导致软件系统的规模日渐庞大, 复杂度不断攀升, 让软件设计和验证的难度不断增长. 为应对软件设计、开发和验证中的复杂性, 国内外研究团队不断提出新的、先进的软件设计与验证理论来提升软件开发效率、保证软件制品的安全性和可靠性. 软件工程的形式化方法[1] 是目前最有前景的软件开发方法之一. 在形式化方法的支持下, 研究人员

① 基金项目: 陕西省重点研发计划 (2018GY-028, 2020GY-084); 陕西省自然科学基金 (2020JM-633); 陕西省教育厅科研计划 (20JG014)
Foundation item: Key Research and Development Program of Shaanxi Province (2018GY-028, 2020GY-084); Natural Science Foundation of Shaanxi Province (2020JM-633); Scientific Research Program of Education Bureau, Shaanxi Province (20JG014)
收稿时间: 2020-12-13; 修改时间: 2021-01-11; 采用时间: 2021-01-27; csa 在线出版时间: 2021-09-02

能够使用严格的数学模型来描述系统的需求, 并验证给定的系统或系统模型是否满足所要求的行为属性[2]. 虽然形式化方法已经被证明是保证系统的正确性和一致性的良好方法, 但是其在软件工程中的应用一直无法大范围推广. 其原因在于形式化方法的学习成本较高、可理解性差, 且形式化模型在结构化、模块化和复用性方面存在一定局限.

Abrial 等发明的 Event-B[3] 是目前最贴近软件工程的一种形式化语言, 其逐步精化的思想和自动代码生成的能力, 不仅保证了模型的正确性和一致性, 同时又能对软件开发的全寿命周期提供良好的支持. Event-B 方法已经成为支持软件工程形式化的主要方法之一.

本文首先对已有的基于 Event-B 的软件工程形式化方法进行了分类阐述, 主要分为 Event-B 模型的控制结构、面向对象的 Event-B 模型、可重用的 Event-B 模型、实时 Event-B 模型. 而后对基于 Event-B 的软件工程形式化方法的未来发展方向提出了预测和建议; 最后对本文内容进行总结.

2 基于 Event-B 的软件工程形式化方法现状分析

2.1 Event-B 模型的控制结构

Hallerstede 等[4] 指出, Event-B 在结构化建模方面的一个重要缺陷就是无法提供描述事件发生次序的机制, 也就是无法表达系统的控制流. 为了对事件排序, 通常需要在 Event-B 模型中额外的构造一个抽象计数器. Hallerstede 等认为主要原因在于 Event-B 为了达到最简化的符号系统, 过多的舍弃了 B 方法中的大量的控制结构. 因此, 原生的 Event-B 模型的控制流通常都是不可见的, 因为一个 Event-B 模型中的事件都是“平坦”的, 建模者看到的就是一个大的事件集合, 既没有像 UML 或者 SysML 那样将系统分解为子系统、构件、对象, 也没有像序列图、流程图那样易理解的符号系统来表达事件之间的关系. 为解决这个问题, 研究人员在这方面所做的工作包括边精化和节点精化、事件精化结构、流语言以及 CSP||B 方法.

(1) 边精化和节点精化

为了能够清晰地表达 Event-B 模型的事件次序, Hallerstede 等介绍了一套结构化的符号来描述 Event-B 模型的 3 种基本结构 (step, loop, choice)[4] 和 1 种复合结构 box. 在其符号系统中, 采用断言作为节点, 并用事件在转换边上标记. 采用这种方法, 能够让 Event-B 模型中的事件次序变得更加清晰, 同时也能够让不变式变得更加简单. 在此基础上, Hallerstede 等在文献 [5] 中提出了对状态转换系统进行精化的两种方法: 边精化和节点精化. 从 Event-B 模型的角度来说, 边精化就是对事件的分解, 将一个事件分解为几个子事件; 而节点精化则是对状态的精化, 是将一个抽象状态精化为几个具体的状态.

Hallerstede 等的工作[4,5] 为 Event-B 模型的控制结构提供了很好的建议和思路, 但是并没有提供显式的图形化工具来支持 Event-B 的控制流建模. 但是边精化和节点精化的思想为后续的各种 Event-B 控制结构表示方法提供了理论指导.

(2) 事件精化结构

Butler[6] 首先提出了事件精化图的思想, 希望能够用 Jackson 结构图的风格来表达 Event-B 模型中抽象事件和精化事件之间的关系. Fathabadi 等 [7] 在此基础上提出了 3 种基本的原子分解结构, 分别为: (1) 顺序结构, 表示抽象事件被分解为几个顺序执行的精化事件; (2) or 结构, 表示抽象事件被分解为几个可能的精化事件; (3) 循环结构, 表示精化事件被分解为一个能够多次重复执行的精化事件. Fathabadi 等将这种方法应用于多媒体协议的开发, 并指出, 在建立大型复杂系统的 Event-B 模型时, 精化层次过多必然会让模型的复杂度提升, 而原子分解方法可以在一定程度上缓解这种复杂性. 随后, Fathabadi 等 [8] 在建立水星探测系统的 Event-B 模型时, 又提出了事件原子分解的 Xor 结构和 All 结构, 分别表示“抽象事件被分解为不可能同时发生的多个精化事件”以及“抽象事件被分解为一系列必须全都发生的精化事件”, 并为原子分解模式添加了多实例的情况. 2012 年, Fathabadi 等 [9] 将原子分解方法正式命名为 AD (Atomicity Decomposition) 方法, 并定义了原子分解语言 ADL (Atomicity Decomposition Language) 来描述原子分解模式; 同时, Fathabadi 等采用模型转换技术, 将 ADL 定义的原子分解模式转换为对应的 Event-B 模型. Fathabadi 在他的博士论文[10] 中, 对其工作做了总结, 共提出了 8 种事件分解模式, 包括 5 种控制流模式 Sequence 模式、Loop 模式、And 模式、Or 模式、Xor 模式和 3 种 Replicator 模式: All 模式、Some 模式和 One 模式.

此后, 南安普顿大学的 Alkhammash 等[11] 将

AD 方法和 UML-B 方法结合, 提出了一种综合结构化精化和控制流精化的 Event-B 模型构建方法, 将需求划分为面向数据的需求、面向事件的需求、面向约束的需求、面向控制流的需求以及其它需求. 它使用 UML-B 来建模面向数据的需求, 并使用 AD 方法来建模面向控制流的需求. 采用 UML-B 和 AD 方法就可以覆盖整个系统需求建模, 并保证需求的可追溯性; 同时还提出了 AD 方法不仅能够描述多级精化模型中的事件精化关系, 还能够表达控制流需求.

Fathabadi 等持续改进 AD 方法, 并将其重新命名为 ERS (Event Refinement Structure) 方法[12], 将 AD 插件做了进一步的完善, 变为 ERS 插件.

事件精化结构的优势在于, 系统建模人员可以从宏观上评估各种精化结构的优劣, 为评估不同的系统精化策略提供依据. 同时 AD/ERS 图形化插件所表达的事件精化结构可以生成大部分的 Event-B 控制流代码, 免除了建模人员的手工编码负担.

事件精化结构的局限在于, 它更适合表达各精化层级之间的静态的精化关系, 也就是事件精化的静态结构. 在表达系统的动态行为方面, AD/ERS 方法所使用的树形结构图并不直观. AD/ERS 方法的另一个局限就是很难映射到类似 CSP 或者 LTS 这样的行为语义模型上, 因此也就难以方便的验证模型的行为属性.

(3) 流语言

纽卡索大学的 Iliasov[13] 提出了一种不修改 Event-B 模型而对其施加控制流约束的方法, 称之为流语言. 流语言采用 ena、dis、和 fis 分别表达一个事件使能下一个事件, 一个事件不使能下一个事件, 以及一个事件可能会使能下一个事件; 进一步的, 流语言采用 And, Or 和 Xor 来表达事件之间的并发、“非排斥或”和“排斥或”关系, 从而得到了一个结构化的语言, 以支持对 Event-B 模型的事件顺序的建模和验证. Iliasov 用流插件提供了图形化的符号来建模事件序列. 从图形含义的角度来说, 流插件用节点表达事件, 而用迁移来表达事件之间的使能 (enable) 关系.

流语言的优势在于, 它能直接用平面的图形来表达事件之间的触发-响应关系, 也就是事件的发生次序, 从这个角度来说, 它比 AD/ERS 方法更好. 同时它也提供了并发事件和互斥事件的表达方式, 对并发程序的建模提供了支持.

流语言的局限性也很明显, 那就是它所采用的事件-触发形式的控制流表达方式与通用的状态转换系统的符号不一致, 要将其转换为某种行为语义模型, 还需要做大量的转换工作. 另外, 流插件存在的另一个问题是, 它希望添加一个语法糖来修改 Event-B 本身的语法, 例如 inc.*(double). 这就修改了 Event-B 的语法结构, 使得模型更加难以理解.

(4) CSP||B

正如 Hallerstede 等指出的[4], Event-B 的缺陷之一是没有提供表达事件次序的机制, 即, 它无法表达系统的控制流. 从某种意义上说, AD/ERS 方法、iUML-B 状态机和 Flow 方法都可被视为是一种控制流建模语言. 但是这些建模语言都不是严格的形式化的控制流模型. 为了对 Event-B 的控制流进行形式化的建模, 研究人员提出了一些方法, 其中最典型的就是 CSP||B 方法, 其初衷是为经典 B 提供控制流语义[14]. 由于 CSP 是一种进程代数, 具有明确的行为语义, 且 CSP 中的失效-发散语义与 Event-B 模型中的相应事件类型极其类似, 因此 CSP||B 是一种非常适用于 Event-B 的行为语义模型.

Butler[15] 主持开发了从 CSP 转换到经典 B 的工具 csp2B. 其主要思路是使用 CSP 表达经典 B 中的事件次序, 而用经典 B 表达事件的计算部分. csp2B 将 CSP 控制模型转换为经典 B 之后, 能够自动生成证明责任, 以保证模型的一致性. 这样, 经典 B 模型的精化就可以使用 CSP 或者 CSP 和 B 的结合来完成. 系统模型的行为精化由 CSP 来保证, 而动作精化 (也就是数据精化) 由 B 模型来保证, 最终的系统模型就是 CSP 模型和 B 模型的组合, 因此这种方法又被称为 CSP||B. Treharne 等[16] 对 CSP||B 方法进行了改进, 让 CSP 进程和 B 模型中的事件的隔离性更强. 随后, Butler 又对经典 B 的模型检查工具 ProB 进行了改进[17], 让其能够检查 CSP 和 B 编写的组合规约的一致性, 并能够检查使用纯粹的 B 编写的规约是否满足用 CSP 描述的事件迹. Schneider 和 Treharne[18,19] 对 CSP 和 B 模型组合后的系统性质进行研究, 提出了一种保证组合后的模型不会发散的约束. Colind 等[20]使用前述的 CSP||B 方法, 采用自底向上的方式, 用 B 模型表达计算部件, 而用 CSP 表达系统的执行流程, 成功的开发了一个车辆自组织系统的形式化模型, 证明了编写和检查 CSP||B 规范可以帮助消除程序集及其通信协议中的错误和歧义. 由于 CSP 和经典 B 的失效-发散语义类似, 因此 CSP 也被用来作为 B 的行为精化框架[21],

以保障在 B 模型的精化过程中能够保持所需的行为属性.

在 Event-B 逐渐推广之后, 研究人员的关注点转移到了如何结合 CSP 和 Event-B 来开发系统的形式化模型. Schneider 和 Treharne 使用 CSP 作为 Event-B 的控制流建模语言, 从而使 Event-B 模型中的控制流变得明确和清晰, 从而增强可读性, 也便于分析. 同时, 这种方法也简化了 Event-B 模型中的大量的涉及控制流的规约代码[22]. 更重要的是, Schneider 等提出了一个行为组合框架, 证明了独立的 CSP||B 组件的精化模型在组合之后, 得到的组合模型行为也是原来的抽象模型的行为的精化. Schneider 等用一个超时重传协议的模型的案例证明了这种方法的优势[23]. 与经典 B 一样, Event-B 也可以用 CSP 作为它的行为语义模型, 从而保证 Event-B 的精化链中的行为属性 (如安全性属性) 不被违背[24]. Schneider 等证明了, 只要 CSP 控制流模型是无死锁的, 而 Event-B 功能模型是非发散的, 则这两个模型的组合获得的模型是无死锁的[25].

CSP||B 方法的优势在于它直接使用形式化的行为建模语言 CSP 来建立 Event-B 模型的控制流, 使得 Event-B 模型的行为属性验证更加简单.

CSP||B 方法的局限在于, 它要求建模人员必须精通两种不同的形式系统, 提升了学习成本, 对建模人员的理论水平要求较高.

在各种 Event-B 控制流建模方法中, 边精化/节点精化方法、事件精化结构合流语言有一些共性的内容, 都将顺序、选择、循环和并发结构加入到了 Event-B 模型中. 这些方法的优点在于采用了各种 (图形化的或非图形化的) 方式让 Event-B 模型的控制流变得更加清晰, 其自动生成的控制流代码也免去了手工编写模型的负担, 减少了人为引入的错误. 但是这些方法都有一个重大的缺陷, 即, 它们都没有给 Event-B 模型的控制流赋予行为语义. 因此, 对 Event-B 模型的行为属性验证仍然需要大量的、手工的交互式证明. CSP||B 方法最大的优点是为 Event-B 模型赋予了行为语义. 因此, CSP||B 方法不仅能够更清晰的表达 Event-B 模型的控制流、并根据 CSP 模型生成 Event-B 的控制流代码, 支持组合精化, 而且还能够为 Event-B 模型的精化策略提供指导. 让 Event-B 模型在精化过程中保证其行为属性, 如安全性、活性以及其他各种线性时态逻辑 LTL (Linear Temporal Logic) 属性[26–28]. 而这种能力是其它 3 种方法无法保证的.

Event-B 对控制结构提供支持的方法的总结如表 1 所示.

表 1 Event-B 模型的控制结构方法总结

方法	特点	适用场景	优点	缺点
边精化/节点精化	精化事件或精化变量	—	为其它几种控制结构提供了理论指导	无行为语义、无工具支持
事件精化结构	用树状图表示分层精化关系	需要从宏观上观察和评估系统的整体分解过程时	自动生成控制流代码、精化层次清晰	控制流不清晰、不支持行为属性验证
流语言	用使能关系表达事件次序	系统规模较小时	自动生成控制流代码、表达了事件次序	精化层次不清晰、不支持行为属性验证
CSP||B	使用CSP表达事件的次序	系统控制流可以和事件动作分离时	自动生成控制流代码、表达了事件次序、支持行为属性验证、支持精化链的属性保持	学习成本高

2.2 面向对象的 Event-B

面向对象方法一直是结构化设计和增强软件模块内聚性的公认的方法, 用于体现面向对象的系统分析与设计思想的统一建模语言 UML 也已被广大工程人员所接受. Event-B 的研究人员为推进 Event-B 在工程方面的应用, 提出了一系列的综合 UML 和 Event-B 的方法, 为 Event-B 模型的结构化做出了贡献.

南安普顿大学的 Snook 等在 Event-B 的 UML 表达方面所做的工作最为杰出. Snook 等首先用 UML Profile [29,30] 构造了从 UML 的类图和状态图到经典 B 的翻译器; 由于 UML 本身非常复杂, 所以 Snook 等仅针对 UML 中的类图和状态机设计了相应的翻译器, 可以将类和类之间的各种关联关系, 例如继承、引用等映射到经典 B 模型中的集合、变量和关系上; 同时, UML-B 可以将状态机映射到经典 B 语言中的状态变化上. 为了表达程序的控制流和并发执行, UML-B 还使用了 UML 活动图中的一些建模元素, 例如 Fork、Join, 伪状态节点等, 这些节点的添加也会影响到所生成的经典 B 的代码.

随着 Event-B 取代经典 B 成为主流的软件系统

建模语言, Snook 等又对 UML-B 做了改进, 定义了 UML-B 的正式的元模型[31]. 这使得 UML-B 能够更好地为 Event-B 服务, 而不是迎合 UML 的各种建模符号, 也使得 UML-B 变成了一种“类似于 UML”的独立的形式化建模语言. 这一变化带来的最重要的影响是, 在此以后, UML-B 模型的结构、精化就代表了 (或者说, 就是) 整个 Event-B 模型的结构和精化, 而不是像 Profile 机制那样让 Event-B 作为 UML 精化的附属品. 而 UML-B 的结构和精化问题也就成为了一个独立的问题, 和 UML 中的类图和状态图的精化区别开来. 当然, UML-B 的类图和状态机形式的图形化表达方式和 Event-B 的文本形式的模型在结构化和精化方面还是存在直觉上的差距. 为此, Snook 等专注于研究如何用 UML-B 的精化来表达 Event-B 模型的精化, 他和 Hallerstede 一起研究了如何用 UML-B 状态机来表达状态转换图的边精化和节点精化问题, 认为可以用层次化的状态机来模拟添加到状态空间和相应事件的细节 (即节点精化), 并用 choice 伪状态节点来表达事件的分解 (即边精化). Snook 等将这两种精化分别称为数据精化和事件精化[32]. Said 等在 Snook 工作的基础上, 提出 UML-B 应该支持 5 种精化方式: 在精化类中添加新的属性和关联、在精化 Machine 中添加新的类、状态精化、转换精化和事件在类之间的移动, 并对 UML-B 进行了扩展, 使之支持这 5 种精化方式[33,34].

UML-B 的优点体现在: 首先, 它的出现和发展对基于 Event 的软件工程形式化方法提供了强有力的支撑. 由于 UML-B 能够自动生成文本形式的 Event-B 模型, 所以模型生产速度更快, 因此对问题领域的探索更加方便, 这一点非常有助于熟悉 UML 的工程人员尽快建立问题域的抽象的形式化模型, 而不用纠结于形式化的数学语言[35]. 其次, UML-B 采用层次化的状态机逐步完成模型的精化, 在一定程度上隐藏了底层的实现细节, 符合软件开发人员的思维习惯; 最后, 通过其建模案例的研究, Said 等证明了[36,37], 使用 UML-B 精化比使用 Event-B 精化更容易完成精化的正确性证明, 这充分体现了 Event-B 模型的结构化能够在一定程度上降低形式化模型建模和验证的复杂性.

UML-B 的局限性包括: 首先, 类图和状态图的粒度过细, 对于复杂系统来说, 会有过多的交互和控制流需要表达. 其次, UML-B 模型无法与已有的 Event-B machine 集成; 最后, UML-B 状态机表达方式无法表达并行的状态机.

为解决 UML-B 的缺点, 南安普顿大学对 UML-B 持续地改进, 使之能够嵌入到传统的 Event-B 模型代码中, 并将现存的 Event-B 模型中的事件与状态机中的事件“链接”起来, 从而达到控制事件次序的效果. 目前, UML-B 已经发展为“集成的 UML-B”(integrated UML-B, iUML-B), 并已应用于各领域的系统建模[38]. iUML-B 状态机的另一个重要的优势就是引入了并发状态机, 为直观地建模并发系统提供了支持.

2.3 可重用的 Event-B 模型

形式化模型, 尤其是以定理证明方法为基础的形式化模型, 通常需要建模人员手工完成大量的证明, 以保证模型的正确性和精化的一致性. 因此, 如何减小手动证明和交互式证明的工作量, 提升形式化模型的可重用性, 也是软件工程形式化方法的一个重要研究领域.

(1) 基于接口的 Event-B 模型

依赖于接口而不是依赖于实现, 是系统分层设计、独立演化的基本策略, 也是增强模型的复用性和扩展性的基本原则. 纽卡索大学的 Iliasov 和奥博学术大学的 Troubitsyna 等提出了基于接口的 Event-B 模型设计策略, 称之为 Modularisation 方法[39], 其思路是将 Event-B 模型分解为一个个的模块, 每个模块由模块接口和模块实现构成. 模块接口包含了与环境交互的外部变量定义、不变式定义和操作定义; 模块的实现则是一个 Event-B machine. 模块实现用事件来完成模块接口中所定义的操作, 而模块的使用者调用模块接口来使用模块的服务. Modularisation 方法被应用于欧盟第七框架的卫星姿态和轨道控制系统的建模和验证[40,41]. 进一步的, Modularisation 方法已被用于契约式的形式化模型的设计[42], 从系统的形式化的 Event-B 模型导出每个组件的契约, 从而保证组件实现的互操作性.

基于接口的 Event-B 模型的优势是分离了 Event-B 模型的接口描述和接口实现, 达到了形式化模型的接口与实现分离. 由此而带来的好处包括: 支持 Event-B 模型的架构设计和推理; 各模块的实现可以独立完成, 而不用关注其他模块的变化; 提升了 Event-B 模型的可扩展性等. 同时, 基于接口的 Event-B 模型也解决了多层、多模态的控制系统中模式一致性难以验证的问题, 并利用架构分解的方法降低了大型复杂系统的形式化模型的复杂度.

基于接口的 Event-B 模型的局限性在于, 它对形

式化模型的设计者的经验有着更高的要求. 接口一旦定义, 就无法在后续的过程中随意修改. 因为后续的精化都必须符合接口的需求. 这从某种程度上限制了形式化模型的设计者对设计空间的探索.

(2) 基于组件的 Event-B 模型

无论是原子分解、UML-B 还是基于接口的设计, 都是为自顶向下的形式化建模提供支持. 而在自底向上的形式化模型重用方面, 还很少有人开展工作. 芬兰图尔库计算机科学中心的 Edmunds 和 Snook 等合作, 提出了使用已有的形式化组件库来逐步构建系统形式化模型的方法[43–45]. Edmunds 扩展了 iUML-B 的类图, 提出了接口类, 并用组件实例图来表达组件之间的组合. 组件实例图方法借鉴了 CSP 的组合语义, 将 Event-B 的事件分为输入事件、输出事件这种与外部产生交互的接口事件以及不会与外部交互的内部事件, 并沿用了 CSP 的组合规则, 即两个组件在共享事件上同步组合, 组合后的事件对外部不可见, 所形成的新的较大的组件就隐藏了共享事件, 这样, 对于更高的层级来说, 共享事件就变成了一个静默的事件 (τ 事件).

图尔库计算科学中心的 Ostroumov 等也在 Edmunds 的工作的基础上提出了一套可视化的 Event-B 组件库[46], 设计了通用的 Event-B 组件模型和连接器模型. 其中, Event-B 组件模型包括环境事件和功能性事件, 环境事件负责和外部环境交互, 又被分为输入事件和输出事件, 而功能事件负责将输入事件读入的数据进行处理, 并传递给输出事件. Event-B 连接器模型包括顺序连接器模型和平行组合连接器模型. 顺序连接器负责两个组件的数据交互, 而平行组合连接器则完成两个组件的行为同步. Ostroumov 等最终设计和实现了一个可视化的形式化组件库[47], 让用户可以直接用“drag-and-drop”的方式直接使用已有的形式化组件库来构建系统的形式化 Event-B 模型, 并用该组件库搭建了一个液压传动系统和一个轨道交通控制系统的 Event-B 模型.

基于组件的 Event-B 模型的优势在于它实现了大粒度的形式化模型的重用, 让系统建模人员能够采用快速地搭建大型系统的形式化模型, 也达到了证明过程的重用. 其局限性在于, 领域组件库中的形式化组件的专用性太强, 目前只开发了液压传动系统和轨道交通控制系统的形式化组件库. 其他领域的建模人员必须自行开发新的组件库, 才能实现自底向上的系统建模过程.

(3) Event-B 设计模式

设计模式是软件工程中实现软件设计思想重用的重要方法, 可以被视为是微型的、可重用的软件体系结构模型. 为提高 Event-B 模型的可重用性, 研究人员也提出了 Event-B 设计模式的概念[48]. Silva 等[49,50]提出了设计模式和“Generic Instantiation”的方法, 并用这种技术完成了安全关键的地铁系统的 Event-B 模型的开发. 在设计模式的实例化过程中, Generic Instantiation 方法可以使用重命名插件来完成设计模式的实例化, 在设计模式中已经完成的证明责任是无需重复证明的. Yeganefard 等[51] 在将 Monitored, Controlled, Mode and Commanded (MCMC) 方法应用于 Event-B 模型的过程中, 提出了 Event-B 模型的 monitor 模式 control 模式, mode 模式 和 command 模式, 在构建实际的控制系统的 Event-B 模型时, 只需要将这 4 种事件模式实例化为实际的系统事件即可. Yeganefard 等使用 MCMC 的 4 种模式开发了巡航控制系统 [52]、汽车车道偏离预警系统[53]、车道对中控制器 (LCC)[54] 的 Event-B 模型. 在 Yeganefard 等的工作中, 提出了模式组合的方法, 将简单模式组合后形成组合模式. 但是模式的组合需要工具的支持. 除了以上典型的 Event-B 设计模式之外, Gondal 等[55,56] 提出了一些针对 Event-B 的精化模式和分解/组合模式, 用于面向特征的控制系统的产品线的形式化建模. Intana[57] 使用 Event-B 设计模式建模了无线传感器网络中的精化和组合模式.

Event-B 设计模式的优势在于, 它抽象了某一个领域内的共性问题并将其表述为抽象的形式化模型, 然后证明该模型的正确性. 这样建模人员就能够采用重命名的方式直接将抽象模型实例化为具体问题的形式化模型, 不仅避免了“重复的制造轮子”, 也节省了重复的精化和正确性证明. 但是 Event-B 设计模式也存在一些局限. 和基于组件的 Event-B 一样, 设计模式本身都是针对某个领域的特定问题的共性抽象, 因此每个设计模式所适用的领域也较为固定, 难以直接应用于其他领域.

(4) Event-B 抽象数据类型

作为一种支持代码生成的形式化建模语言, Event-B 必然要支持对各种抽象数据类型, 如线性表、堆栈、队列和树的自定义和可重用技术. Abrial 和 Butler 很早就提出了对 Event-B 进行数学扩展的思路[58], 而后由

Butler 在 Rodin 平台上实现为 Theory 插件[59], 并提供了 Array、Stack、Sequence、Tree、B-Tree 等一系列的标准抽象数据类型库. 建模人员既可以根据需要定义自己所需要的新的抽象数据类型, 也可以直接使用 Rodin 中的标准库. 由于抽象数据类型也可以被视为一种可重用的模式, Basin 等将 Theory 方法和 Generic Instantiation 方法结合起来[60], 使其能够应用于大型复杂系统的形式化建模及验证. Fürst 和 Hoang 用这种综合性的方法构造了复杂的列控系统的形式化模型[61,62], 证明了该方法能够减少手工证明的工作量.

Event-B 抽象数据类型的优势在于它能够大幅提升自动定理证明的程度, 而传统的基于精化的方式则需要大量手动证明. 而其局限性在于, 它没有提供参数化的抽象数据类型的 Event-B 模型, 即通用的抽象数据类型 (例如, 一个与元素类型无关的通用的堆栈类型). 因此, 在实际使用时还无法像面向对象语言的模板类那样通用.

对各种可重用的 Event-B 模型的总结如表 2 所示.

表 2 可重用的 Event-B 模型方法总结

方法	特点	适用场景	优点	缺点
基于接口的Event-B模型	接口与实现分离、各模块独立精化	多模态的复杂系统的分层设计	降低了大型复杂系统的形式化建模的复杂度	限制了形式化模型的设计者对设计空间的探索
基于组件的Event-B模型	组件的建模和正确性证明过程的重用	专用领域的形式化模型构建	加大了重用的粒度、支持自底向上快速搭建系统模型	领域组件库的专用性太强
Event-B设计模式	设计思想和证明过程的重用	专用领域的形式化模型构建	节省了重复的精化和正确性证明	设计模式的领域专用性太强
Event-B抽象数据类型	数据结构和算法以及证明过程的重用	数据处理类系统的形式化模型构建	提升自动定理证明的程度	没有提供参数化的抽象数据类型

2.4 实时 Event-B 模型

实时并发系统的复杂性使得其验证过程必须使用形式化方法来完成, 它使得研究人员能够使用严格的数学模型来描述系统的需求以及系统的行为, 并验证给定的系统或系统模型是否满足所要求的行为属性. 然而, Event-B 本身不支持建模时间概念, 在表达时间方面的能力有一定的局限, 因此也难以完成实时系统的时间属性的验证.

为解决 Event-B 对实时系统的建模问题, 研究人员已经做了一些前期的工作. 其主要方法是使用 Event-B 建模语言本身的能力来建模时间, 通常是使用离散的时钟滴答事件 (tick_tock) 来建模时间的流逝, 并建模各种时间间隔模式; Butler 等[63] 最早提出了在经典 B 中建模离散时间的方法, 这种方法使用一个自然数变量 (名叫时钟变量) 来表达当前时间, 通过增加这个变量的值来表示时间的流逝. 他和经典的时间系统的最大区别在于, 如果当前时间等于截止时间, 则使用一定的操作来阻止时间的流逝. 这种建模思想被后续的所有工作所采纳. 是后续所有实时 Event-B 模型的理论基础.

Cansell 等[64] 首先提出了 Event-B 的时间约束模式的概念, 并专门采用 tick_tock 事件来表达时间的流逝. 时间约束模式首先将时间的流逝与 Event-B 中的事件关联起来, 并将事件发生的次序从定性的“先后次序”精确到了用变量表达的时间段上. 但是 Cansell 并没有进一步对各种时间模式进行分类.

Rehm 等[65−67] 提出了“持续时间模式”(duration pattern) 来表达一个事件的持续时间, 让建模人员可以在 Event-B 框架中对实时属性进行建模和推理. 但是同样没有对各种时间模式分类.

Sarshogha 等[68,69] 提出了 3 种时间模式: Delay 模式、Expiry 模式和 Deadline 模式, 用来表示实时系统的延迟、超时和截止期的概念. 这 3 种模式的提出, 为实时系统的 Event-B 建模提供了通用的参考模型. 但是其模式没有考虑时间间隔问题, 且模式所考虑的场景较少.

Sulskus 等[70,71] 在 Sarshogha 的工作基础上提出了时间间隔方法, 用 iUML-B 状态机的单个状态表示时间间隔, 并分别用状态节点的入边和出边来表示该时间间隔的触发事件 (trigger) 和响应事件 (response). Sulskus 开发了一个称为 tiGen (time interval Generator) 的工具来支持其时间建模的思想.

时间间隔方法的优势在于其使用 JSD 风格和 iUML-B 状态机共同表达了各种时间间隔模式, 为实时系统的形式化建模及正确性证明提供了丰富的可重用的实时模式, 并节省了大量的时间属性的证明工作. 这种方

法的局限在于, 它依然是在 Event-B 建模框架中描述实时系统, 仍然需要大量的手工证明工作. 另外, 这些模式也无法方便地转换为时间转换系统模型, 因此也难以使用模型检测方法来减轻时间属性验证的负担.

在表 3 中对各种实时 Event-B 建模方法的优缺点做出了总结.

表 3 实时 Event-B 方法总结

方法名称	特点	适用场景	优点	缺点
时钟变量	用时钟变量建模时间	仅需要记录事件发生的时刻的场景	—	—
时间约束模式	用tick_tock事件表达时间流逝	需要表达时间流逝的场景	提供了一个表达时间流逝的触发事件	未对时间模式分类
持续时间模式	用持续时间建模事件的持续时间	需要定量地表达事件持续时间的场景	提升了自动证明的比率	未对时间模式分类
3种时间模式	表达了实时系统的延迟、超时和截止期的概念	操作系统任务调度、实时操作系统建模	供了通用的实时Event-B参考模型	模式所考虑的场景较少
时间间隔模式	用iUML-B状态机的节点表达时间间隔	有各种复杂场景的实时系统	提供了丰富的可重用的时间模式、节省了时间属性的证明工作	难以转换为时间转换系统、不支持时间属性验证

2.5 国内外研究现状小结

可以看出, 自从 Event-B 建模方法被提出以来, 研究人员已经提出了大量的新方法、新思路和支持工具来帮助其支持软件开发过程. 经过近十几年的努力, 这些方法及其所提出的插件被集成到基于 Eclipse 开发的 Rodin 平台上, 存在很好的互操作性. 表 4 总结了近年来 Event-B 为支持软件工程和系统工程的全寿命周期所提出的方法和插件.

表 4 Event-B 为支持软件工程和系统工程所提出的方法和插件

软件寿命周期	插件/方法名称	支持的能力
需求分析	ProR[72]	需求管理
概要设计	CODA[73]	定义软件体系结构
概要设计	Modularisation	定义系统接口
概要设计	Component Diagram	定义系统构件
概要/详细设计	UML-B、iUML-B statemachine	定义对象及对象内部状态
详细设计	Pattern、Generic Instantiation	定义可重用的设计思路
详细设计	Flow、ERS	定义系统控制流
详细设计	Theory	定义抽象数据类型
编码	Code Generation (Java、C、C++、VHDL)[74]	自动代码生成

3 发展的挑战和方向

3.1 挑战

(1) 随着“软件定义”时代的到来, 软件的应用领域已扩充到人类认识所能达到的所有领域, 软件的范型已经从结构化、构件化范型变为服务化、网络化、云化范型. 软件范型的转变必然对软件工程形式化方法带来新的挑战. 这种挑战对基于 Event-B 的“构造即正确”的方法尤为明显. 系统正确性的关注点已不再局限于某个模块、组件内部的正确性, 还需要考虑系统整体的大量的对象交互、行为互操作等特点.

(2) 软件的规模、复杂性呈现爆炸式增长, 传统的模型检测方法因其探索空间有限, 无法完成大型复杂软件系统的各种行为属性验证. 而基于 Event-B 的形式化方法也面临着证明工作繁杂、证明效率低等问题.

(3) 软件制造的泛在化对软件形式化建模工具提出了更多的易用性、易理解性的要求, 也为软件工程形式化提出了更多的挑战. 大量的非计算机专业的系统开发人员需要更加贴近用户需求的非形式化的“前端”来描述问题域、进行需求分析, 并能在工具的支持下将这些非形式化的领域模型转换为 Event-B 模型.

3.2 发展的方向

对于以上挑战, 本文提出以下几点应对方案, 也是对未来基于 Event-B 的软件工程形式化方法的趋势的预测:

(1) 增加对软件体系结构建模工具的支持, 让其能够表达大粒度的系统构件之间的控制流和数据流关系并映射为 Event-B 模型. 这种方法使系统构建者能够从更加宏观和抽象的视角研究系统的“涌现”特性, 而不是关注于某些方面的实现细节的正确性. 现有的对 Event-B 和 BIP[75] 共同进行组合建模的工作以及将 SysML[76,77] 和 UML 活动图[78] 转换为 Event-B 模型的工作是一个很好解决方案, de Sousa 和 Snook 等提出了将 IOD 也归入到 UML-B 中的建议[79] 也能让 Event-B 更加符合软件工程人员的习惯, 但是还缺少在 Rodin 环境中的工具支持.

(2) 增加对新的编程范型的半形式化建模工具的支持以及这些建模工具到 Event-B 语言的转换工具. 在这些领域, 研究人员正在开展的工作包括使用 Event-B 建立面向服务的架构[80]、微服务架构[81]、云计算[82]、物联网应用[83]、区块链[84]、自适应软件[85] 等, 但所提出的方法还没有形成系统的、可用的插件和工具.

(3) 提出更多的、更加适合最终用户的领域编程语言和领域建模工具以及相应的模型转换工具, 实现各种非形式化的领域建模语言到 Event-B 模型的转换, 为问题域到解空间的映射提供更好的桥梁. 由法国国家研究局 ANR 支持的 IMPEX 工程正在研究如何将本体语言描述的领域本体模型转换为 Event-B 的本体结构[86,87], 是一个可以参考的方案.

(4) 通用软件的安全性和可靠性需求会催生更多将 Event-B 模型转换为行为模型的模型转换工具. Event-B 没有行为语义, 也没有时间语义, 这也使得我们可以将其映射到任何所需的论域, 例如 Labeled Transition System[88] 及其各种变体, 如 Timed Transition system[89] 等, 为其赋予行为语义和时间语义, 从而完成各种行为属性和时间属性的验证. Peng 等[90] 在 Event-B 的 LTS 语义方面所做的研究可以作为一个参考的思路.

4 结束语

当前计算的泛在化趋势、信息系统向物理世界迁移以及人机物融合的趋势将形式化方法的应用从安全关键领域渗透到各种通用软件领域, 对各种新方法、新工具以及集成开发环境的要求日益迫切. Event-B 作为一种支持“correct-by-construction”思想的定理证明语言, 为软件工程形式化提供了重要支撑.

本文首次以软件工程对形式化方法所提出的需求为出发点, 以一种形式化建模语言对软件工程全寿命周期的支持程度来研究软件工程形式化方法, 对 Event-B 建模语言为软件工程所提供的支撑进行分类阐述, 对软件工程形式化方法的推广和应用具有一定的借鉴和参考价值.

参考文献

1 Alagar VS, Periyasamy K. Specification of Software Systems. London: Springer, 2011.

2 Baier C, Katoen JP. Principles of Model Checking. Cambridge: The MIT Press, 2008.

3 Abrial JR. Modeling in Event-B: System and Software Engineering. Cambridge: Cambridge University Press, 2010.

4 Hallerstede S. Structured Event-B models and proofs. Proceedings of the 2nd International Conference on Abstract State Machines, Alloy, B and Z. Orford, QC, Canada. 2010. 273–286.

5 Hallerstede S, Snook C. Refining nodes and edges of state machines. Proceedings of the 13th International Conference on Formal Engineering Methods. Durham, UK. 2011. 569–584.

6 Butler M. Decomposition structures for Event-B. Proceedings of the 7th International Conference on Integrated Formal Methods. Düsseldorf, Germany. 2009. 20–38.

7 Fathabadi AS, Butler M. Applying Event-B atomicity decomposition to a multi media protocol. Proceedings of the 8th International Symposium on Formal Methods for Components and Objects. Eindhoven, the Netherlands. 2010. 89–104.

8 Fathabadi AS, Rezazadeh A, Butler M. Applying atomicity and model decomposition to a space craft system in Event-B. Proceedings of the 3rd International Conference on NASA Formal Methods. Pasadena, CA, USA. 2011. 328–342.

9 Fathabadi AS, Butler M, Rezazadeh A. A systematic approach to atomicity decomposition in Event-B. Proceedings of the 10th International Conference on Software Engineering and Formal Methods. Thessaloniki, Greek. 2012. 78–93.

10 Fathabadi AS. An approach to atomicity decomposition in the Event-B formal method [Ph. D. thesis]. Southampton: University of Southampton, 2012.

11 Alkhammash E, Butler M, Fathabadi AS, *et al*. Building traceable Event-B models from requirements. Science of Computer Programming, 2015, 111: 318–338. [doi: 10.1016/j.scico.2015.06.002]

12 Fathabadi AS, Butler M, Rezazadeh A. Language and tool support for event refinement structures in Event-B. Formal Aspects of Computing, 2015, 27(3): 499–523. [doi: 10.1007/

s00165-014-0311-1]
13 Iliasov A. Use case scenarios as verification conditions: Event-B/flow approach. Proceedings of the 3rd Software Engineering for Resilient Systems. Geneva, Switzerland. 2011. 9–23.
14 Treharne H, Schneider S. Using a process algebra to control B OPERATIONS. Proceedings of the 1st International Conference on Integrated Formal Methods. York, UK. 1999. 437–456.
15 Butler M. csp2B: A practical approach to combining CSP and B. Formal Aspects of Computing, 2000, 12(3): 182–198. [doi: 10.1007/PL00003930]
16 Treharne H, Schneider S. How to drive a B machine. Proceedings of the 1st International Conference of B and Z Users on Formal Specification and Development in Z and B. York, UK. 2000. 188–208.
17 Butler M, Leuschel M. Combining CSP and B for specification and property verification. International Symposium on Formal Methods. Newcastle, UK. 2005. 221–236.
18 Schneider S, Treharne H. CSP theorems for communicating B machines. Formal Aspects of Computing, 2005, 17(4): 390–422. [doi: 10.1007/s00165-005-0076-7]
19 Schneider S, Treharne H, Evans N. Chunks: Component verification in CSP||B. Proceedings of the 5th International Conference on Integrated Formal Methods. Eindhoven, the Netherlands. 2005. 89–108.
20 Colin S, Lanoix A, Kouchnarenko O, *et al*. Towards validating a platoon of cristal vehicles using CSP||B. Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology. Urbana, IL, USA. 2008. 139–144.
21 Schneider S, Treharne H. Changing system interfaces consistently: A new refinement strategy for CSP||B. Science of Computer Programming, 2011, 76(10): 837–860. [doi: 10.1016/j.scico.2010.08.001]
22 Schneider S, Treharne H, Wehrheim H. A CSP approach to control in Event-B. Proceedings of the 8th International Conference on Integrated Formal Methods. Nancy, France. 2010. 260–274.
23 Schneider S, Treharne H, Wehrheim H. Bounded retransmission in Event-B||CSP: A case study. Electronic Notes in Theoretical Computer Science, 2011, 280: 69–80. [doi: 10.1016/j.entcs.2011.11.019]
24 Schneider S, Treharne H, Wehrheim H. Stepwise refinement in event-B CSP. Part 1: Safety. Department of Computing, University of Surrey, 2011.
25 Schneider SA, Treharne H, Wehrheim H. A CSP account of event-B refinement. Proceedings of 15th International Refinement Workshop. Limerick, Ireland. 2011. 139–154.
26 Schneider S, Treharne H, Wehrheim H. The behavioural semantics of Event-B refinement. Formal Aspects of Computing, 2014, 26(2): 251–280. [doi: 10.1007/s00165-012-0265-0]
27 Schneider S, Treharne H, Wehrheim H, *et al*. Managing LTL properties in Event-B refinement. Proceedings of the 11th International Conference on Integrated Formal Methods. Bertinoro, Italy. 2014. 221–237.
28 Hoang TS, Schneider S, Treharne H, *et al*. Foundations for using linear temporal logic in Event-B refinement. Formal Aspects of Computing, 2016, 28(6): 909–935. [doi: 10.1007/s00165-016-0376-0]
29 Snook C, Oliver I, Butler M. The UML-B profile for formal systems modelling in UML. In: Bernin F, Butler M, Cansell D, *et al*, eds. UML-B Specification for Proven Embedded Systems Design. Boston: Springer, 2004. 69–84.
30 Snook C, Butler M. UML-B: Formal modeling and design aided by UML. ACM Transactions on Software Engineering and Methodology, 2006, 15(1): 92–122. [doi: 10.1145/1125808.1125811]
31 Snook C, Butler M. UML-B and Event-B: An integration of languages and tools. Proceedings of the IASTED International Conference on Software Engineering. Anaheim, CA, USA. 2008. 336–341.
32 Snook C, Waldén M. Refinement of statemachines using Event B semantics. Proceedings of the 7th International Conference of B Users. Besançon, France. 2007. 171–185.
33 Said MY, Butler M, Snook C. Class and state machine refinement in UML-B. Proceedings of Workshop on Integration of Model-based Formal Methods and Tools. 2009.
34 Said MY, Butler M, Snook C. Language and tool support for class and state machine refinement in UML-B. Proceedings of the 2nd World Congress on FM 2009: Formal Methods. Eindhoven, the Netherlands. 2009. 579–595.
35 Snook C, Butler M. UML-B: A plug-in for the Event-B tool set. Proceedings of the International Conference on Abstract State Machines, B and Z. London, UK. 2008. 344–344.
36 Said MY. Methodology of refinement and decomposition in UML-B [Ph.D. thesis]. Southampton: University of Southampton, 2010.
37 Said MY, Butler M, Snook C. A method of refinement in UML-B. Software & Systems Modeling, 2015, 14(4): 1557–1580.
38 Hoang TS, Snook C, Ladenberger L, *et al*. Validating the requirements and design of a hemodialysis machine using iUML-B, BMotion Studio, and Co-Simulation. Proceedings of the 5th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Linz, Austria. 2016. 360–375.

39 Iliasov A, Troubitsyna E, Laibinis L, *et al*. Supporting reuse in Event B development: Modularisation approach. Proceedings of the 2nd International Conference on Abstract State Machines, Alloy, B and Z. Orford, QC, Canada. 2010. 174–188.

40 Iliasov A, Troubitsyna E, Laibinis L, *et al*. Verifying mode consistency for on-board satellite software. Proceedings of the 29th International Conference on Computer Safety, Reliability, and Security. Vienna, Austria. 2010. 126–141.

41 Iliasov A, Troubitsyna E, Laibinis L, *et al*. Developing mode-rich satellite software by refinement in Event-B. Science of Computer Programming, 2013, 78(7): 884–905. [doi: 10.1016/j.scico.2012.04.010]

42 Laibinis L, Troubitsyna E. A contract-based approach to ensuring component interoperability in Event-B. From Action Systems to Distributed Systems. New York: Chapman and Hall/CRC. 2016. 81–96.

43 Edmunds A, Walden M, Snook C. Towards component-based reuse for Event-B. Proceedings of the 27th Nordic Workshop on Programming Theory. Iceland. 2015. 3.

44 Edmunds A, Walden M. Modelling "Operation-Calls" in Event-B with shared-event composition. Proceedings of the 19th Brazilian Symposium on Formal Methods. Natal, Brazil. 2016. 97–111.

45 Edmunds A, Snook C, Walden M. On component-based reuse for Event-B. Proceedings of the 5th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Linz, Austria. 2016. 151–166.

46 Ostroumov S, Waldén M. Facilitating formal Event-B development by visual component-based design. Turku: Turku Centre for Computer Science, 2015.

47 Ostroumov S, Waldén M. Formal library of visual components. Turku: Turku Centre for Computer Science, 2015.

48 Hoang TS, Fürst A, Abrial JR. Event-B patterns and their tool support. Software & Systems Modeling, 2013, 12(2): 229–244.

49 Silva R, Butler M. Supporting reuse of Event-B developments through generic instantiation. Proceedings of the 11th International Conference on Formal Engineering Methods on Formal Methods and Software Engineering. Rio de Janeiro, Brazil. 2009. 466–484.

50 Silva R. Application of decomposition and generic instantiation to a metro system in Event-B. Environmental Modelling & Software, 2011, 47(2): 138–147.

51 Yeganefard S, Butler M, Rezazadeh A. Evaluation of a guideline by formal modelling of cruise control system in Event-B. Proceedings of the 2nd NASA Formal Methods Symposium. Washington, DC, USA. 2010. 182–191.

52 Yeganefard S, Butler M. Problem decomposition and sub-model reconciliation of control systems in Event-B. Proceedings of 2013 IEEE 14th International Conference on Information Reuse & Integration. San Francisco, CA, USA. 2013. 528–535.

53 Yeganefard S, Butler M. Structuring functional requirements of control systems to facilitate refinement-based formalisation. Electronic Communications of the EASST, 2011, 46.

54 Yeganefard S, Butler M. Control systems: Phenomena and structuring functional requirement documents. Proceedings of 2012 IEEE 17th International Conference on Engineering of Complex Computer Systems. Paris, France. 2012. 39–48.

55 Gondal A, Poppleton M, Butler M. Composing Event-B specifications-case-study experience. Proceedings of the 10th International Conference on Software Composition. Zurich, Switzerland. 2011. 100–115.

56 Gondal A. Feature-oriented reuse with Event-B and Rodin [Ph. D. thesis]. Southampton: University of Southampton, 2013.

57 Intana A. Formal engineering methodologies for wireless sensor network development with simulation [Ph.D. thesis]. Southampton: University of Southampton, 2015.

58 Abrial JR, Butler M, Hallerstede S, *et al*. Proposals for mathematical extensions for Event-B [Technical Report]. 2009. http://deploy-eprints.ecs.soton.ac.uk/80/. (2010-04-22).

59 Butler M, Maamria I. Mathematical extension in Event-B through the Rodin theory component. Deploy Project, 2010.

60 Basin D, Fürst A, Hoang TS, *et al*. Abstract data types in Event-B —An application of generic instantiation. arXiv: 1210.7283, 2012.

61 Fürst A, Hoang TS, Basin D, *et al*. Large-scale system development using abstract data types and refinement. Science of Computer Programming, 2016, 131: 59–75. [doi: 10.1016/j.scico.2016.04.010]

62 Fürst A, Hoang TS, Basin D, *et al*. Formal system modelling using abstract data types in Event-B. Proceedings of the 4th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Toulouse, France. 2014. 222–237.

63 Butler M, Falampin J. An approach to modelling and refining timing properties in B. Refinement of Critical Systems, 2002.

64 Cansell D, Méry D, Rehm J. Time constraint patterns for Event B development. Proceedings of the 7th International Conference of B Users. Besançon, France. 2007. 140–154.

65 Rehm J. A method to refine time constraints in Event B framework. Automatic Verification of Critical Systems-AVoCS 2006. Nancy, France. 2006. 173–177.

66 Rehm J. A duration pattern for event-B method. Junior Researcher Workshop on Real-Time Computing-JRWRTC 2008. Rennes, France. 2008.

67 Rehm J. Proved development of the real-time properties of

the IEEE 1394 Root Contention Protocol with the event-B method. International Journal on Software Tools for Technology Transfer, 2010, 12(1): 39–51. [doi: 10.1007/s10009-009-0130-5]

68 Sarshogh MR. Extending Event-B with discrete timing properties [Ph.D. thesis]. Southampton: University of Southampton, 2013.

69 Sarshogh MR, Butler M. Specification and refinement of discrete timing properties in Event-B. Electronic Communications of the EASST, 2011, 46.

70 Sulskus G. An investigation into Event-B methodologies and timing constraint modelling [Ph.D. thesis]. Southampton: University of Southampton, 2017.

71 Sulskus G, Poppleton M, Rezazadeh A. An interval-based approach to modelling time in Event-B. Proceedings of the 6th International Conference on Fundamentals of Software Engineering. Tehran, Iran. 2015. 292–307.

72 Hallerstede S, Jastram M, Ladenberger L. A method and tool for tracing requirements into specifications. Science of Computer Programming, 2014, 82: 2–21. [doi: 10.1016/j.scico.2013.03.008]

73 Butler M, Colley J, Edmunds A, *et al*. Modelling and refinement in CODA. arXiv: 1305. 6112, 2013.

74 Fürst A, Hoang TS, Basin D, *et al*. Code generation for Event-B. Proceedings of the 11th International Conference on Integrated Formal Methods. Bertinoro, Italy. 2014. 323–338.

75 Siala B, Bodeveix JP, Filali M, *et al*. An Event-B development process for the distributed BIP Framework. In: Ait-Ameur Y, Nakajima S, Méry D, eds. Implicit and Explicit Semantics Integration in Proof-Based Developments of Discrete Systems. Singapore: Springer, 2021. 283–307.

76 Fotso SJT, Mammar A, Laleau R, *et al*. Event-B expression and verification of translation rules between SysML/KAOS domain models and B system specifications. Proceedings of the 6th International Conference on Abstract State Machines. Southampton, UK. 2018. 55–70.

77 Zhang Q, Huang ZQ, Xie J. Distributed system model using SysML and Event-B. Proceedings of the 2nd International Conference on Machine Learning and Intelligent Communications. Weihai, China. 2018. 326–336.

78 Achouri A, Hlaoui YB, Ayed LJB. Institution-based UML activity diagram transformation with semantic preservation. International Journal of Computational Science and Engineering, 2019, 18(3): 240–251.

79 de Sousa TC, Snook CF, Silva PSM. A proposal for extending UML-B to support a conceptual model. Innovations in Systems and Software Engineering, 2011, 7(4): 293–301. [doi: 10.1007/s11334-011-0169-9]

80 Tounsi I, Kacem MH, Kacem AH, *et al*. Transformation of compound SOA design patterns. Procedia Computer Science, 2017, 109: 408–415. [doi: 10.1016/j.procs.2017.05.410]

81 Vergara S, González L, Ruggia R. Towards formalizing microservices architectural patterns with Event-B. Proceedings of 2020 IEEE International Conference on Software Architecture Companion. Salvador. 2020. 71–74.

82 Lahouij A, Hamel L, Graiet M, *et al*. An Event-B based approach for cloud composite services verification. Formal Aspects of Computing, 2020, 32(4): 361–393.

83 Kacem MH, Tounsi I, Khalfi N. Modeling and specification of bootstrapping and registration design patterns for IoT applications. Proceedings of the 18th International Conference on Smart Homes and Health Telematics. Hammamet, Tunisia. 2020. 55–66.

84 Lahbib A, Wakrime AA, Laouiti A, *et al*. An Event-B based approach for formal modelling and verification of smart contracts. Proceedings of the 34th International Conference on Advanced Information Networking and Applications. Caserta, Italy. 2020. 1303–1318.

85 Han DS, Yang QL, Xing JC, *et al*. EasyModel: A refinement-based modeling and verification approach for self-adaptive software. Journal of Computer Science and Technology, 2020, 35(5): 1016–1046. [doi: 10.1007/s11390-020-0499-x]

86 Ait-Sadoune I. Mohand-Oussaid L. Building formal semantic domain model: An Event-B based approach. Proceedings of the 9th International Conference on Model and Data Engineering. Toulouse, France. 2019. 140–155.

87 Ait-Ameur Y, Ait-Sadoune I, Casteran P, *et al*. On the importance of explicit domain modelling in refinement-based modelling design. Proceedings of the 6th International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z. Southampton, UK. 2018. 425–430.

88 Gorrieri R. Labeled transition systems. Gorrieri R. Process Algebras for Petri Nets: The Alphabetization of Distributed Systems. Cham: Springer, 2017. 15–34.

89 Cimatti A, Griggio A, Magnago E, *et al*. Extending NUXMV with timed transition systems and timed temporal properties. Proceedings of the 31st International Conference on Computer Aided Verification. New York, NY, USA. 2019. 376–386.

90 Peng H, Du CL, Rao L, *et al*. A LTS approach to control in Event-B. Scientific Programming, 2018, 2018: 8765186.