维普资讯 http://www.cqvip.com

基于 EHA 的异常处理模型检验方法*)

杨厚群[1,2] 何中市[1] 陈 静[2]

(重庆大学计算机学院 重庆 400044)[1] (海南大学信息科学技术学院 海口 570228)[2]

摘 要 本文提出在整个 Java 程序开发链中通过使用 UML Statecharts 对异常处理建模,对 Statechart 进行模型检验,完成代码生成。首先将 Statecharts 转换为 EHA,然后给出其操作语义,根据操作语义映射到一个自动机。使用基于自动机理论的模型检验方法来验证基于 EHA 的异常处理模型是否满足某些关键性质,最后自动产生相关代码。

关键词 异常处理,模型检验,EHA

Approach of Model Checking for Exception Handling Based on EHA

YANG Hou-Qun[1,2] HE Zhong-Shi[1] CHEN Jing[2]

(College of Computer Science, Chongqing University, Chongqing 400044)[1]

(College of Information Science & Technology, Hainan University, Haikou 570228)[2]

Abstract This paper proposes an approach of handling exception throughout the entire development chain of Java programs by modeling exception handling in the UML statechart model, then model checking statecharts and utilizing automatic code generators for implementing the source. At first, the system statecharts model is converted into automaton by EHA operational semantics. The statecharts model is verified whether satisfies the some key properties. Lastly, source code are generated.

Keywords Exception handling, Model checking, EHA

自动机是由 UML Statecharts 的概念上发展而来的,一个状态图就是用一系列的状态展示一个事务的生命周期,事件驱动状态间的变迁过程,其中动作是随一套复杂的实际条件和当时的情况随机发生的,这套条件的抽象组成关口条件(guard condition),如果这个事件产生的结果通过了这个关口条件,事务的进程则进入到它的生命周期的下个阶段(即下个状态)。为满足创建这种类型的模型要求,自动机提供了许多抽象的类(class)用于事务的抽象工作。

模型检验是一种重要的自动验证技术,主要通过显式状态搜索或隐式 UML 模型可以作为测试该软件的依据,辅助测试数据的生成。一个直接的应用是将 UML Statecharts 所描述的对象状态机作为异常处理检验模型,但 UML 建模时通常不考虑 UML Statecharts 的状态和迁移的形式化定义。Statecharts 已经存在许多种语义解释。现有的 UML Statecharts 操作语义主要将 UML Statecharts [8]转换为适合于验证工具 SPIN、SMV 等的中间描述机制——层次自动机(EHA),以便进行形式化验证,这种转化摒弃了对象状态图本身的一些特性。

为此,本文首先对 EHA 的操作语义进行相应的改进,并根据 UML2.0 语义文档[1],通过 UML Statecharts 的建模元素和特征,尤其是层次结构、复合迁移、伪状态描述实例。

在定义语法和语义时,借鉴了文[2~4]的一些基本思想,同时引入了多个"事件队列"描述 UML Statechart 的单事件处理方式。通过 EHA 语义映射 Statechart,为实例的代码自动生成、模拟、测试和测试用例的重用奠定基础。

1 层次自动机及其语义

本文以生产监测系统为例,它是一个生产数据采集和管理的计算机系统,采集生产线上各个生产工艺过程中发生的数据,包括材料和在制品的收进数、投入数、作业数及合格数等,通过汇总、保存和计算,实现对产品产量和质量的管理和控制。系统数据采集有多种方式:关键工序的数据通过光电管采集(为了简化,这里只考虑关键工序的数据),每半分钟更新一次,工控机处理后每半小时更新一次,并在显示屏上显示,最后进入班数据,如图 1 所示。

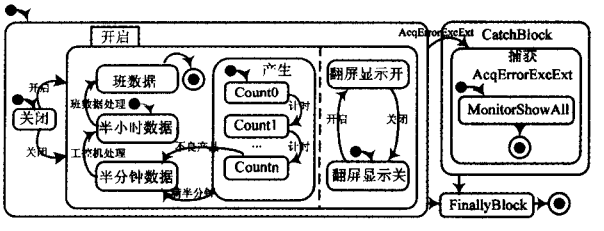

图 1 含异常事件处理模式的光电管数据采集的 Statechart

根据 UML Statecharts 的操作语义把它平面化为层次自动机,而各个层次自动机由若干个顺序自动机组成。

定义 1(顺序自动机) 顺序自动机 A 是一个四元组(σ_A, S_A^0, λ_A, δ_A),σ_A 是有穷状态集合,S_A^0 是初始状态,λ_A 是有穷的迁移标记集合,$\delta_A \subseteq \sigma_A \times \lambda_A \times \sigma_A$ 是迁移关系。

定义 2(扩展层次自动机) EHA H 是一个五元组(F, E, ρ, A_0, V),F 是有穷的顺序自动机集合,$\forall A_1, A_2 \in F, \sigma_{A1} \cap \sigma_{A2} = \phi$;$E$ 是有穷事件集合;V 是变量集合;$\rho: \bigcup_{A \in F} \sigma_A \to 2^F$ 是精化函数,给出 F 的一个树型结构满足:(1) 存在唯一的根自动机 $A_0 \in F$,使得不存在 $s \in \bigcup_{A \in F} \sigma_A, A_0 \in \rho(s)$;(2)每个非根自动机恰有一个父状态,对于 $\forall A \in F \setminus \{A_0\}, \exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} \sigma_{A'}, A \in \rho(s)$;(3) 不存在环路:$\forall S \subseteq \bigcup_{A \in F} \sigma_A, \exists s \in$

*)基金项目:海南省教育厅高校科研项目 Hjkj200603。杨厚群 博士研究生,研究方向:数据挖掘与机器学习、软件工程;何中市 博导,教授。

维普资讯 http://www.cqvip.com

S，$S\cap(_{A\in\rho(x)}\sigma_A)=\emptyset$。

在把 UML Statecharts 转换为 EHA 的过程中，层间迁移提升到它离开和进入的最高层状态之间，从而变为非层间迁移。为了不改变语义，需要对迁移标记进行扩充，添加受限源状态 sr 和确定目标状态 td 这两个集合。sr 将迁移的使能限制在源状态下的一个格局中，td 决定了迁移使系统进入目标状态时有哪些子状态也同时进入。事件集合 E 和变量集合 V 分别由 UML Statecharts 中所有事件和变量组成。

图 2 是图 1 中 Statecharts 对应的 EHA 表达。Statecharts 中的状态被映射为 EHA 状态。并行和非并行的抽取由自动机通过抽取适当的状态来表达。

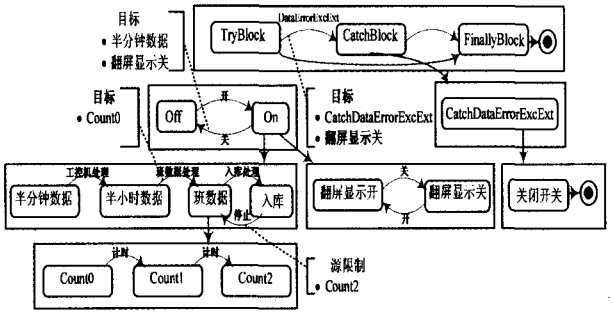

图 2　光电管数据采集的 EHA

层次自动机的全局状态由格局表示，它由包含的顺序自动机的某些局部状态组成。$Conf_H$ 表示由 EHA H 中的所有有效格局组成的集合。

定义 3(格局)　H 的一个格局是一个集合 $Conf\subseteq\cup_{A\in F}\sigma_A$ 使得：(1) $\exists_1 s\in\sigma_{A0}$ 满足 $s\in Conf$；(2) $\forall s,A$，若 $s\in Conf$ 且 $A\in\rho(s)$，则 $\exists_1 s'\in A, s'\in Conf$。

在 UML Statecharts 中，操作语义中的状态称为状况(status)，而迁移就代表 RTC 步。每个状况由格局和当前的环境组成，我们这里只考虑事件环境即当前事件队列中存在的事件。对于事件集合 X，用 ΘX 表示在 X 上某种类型事件队列的所有可能结构。下面用 LTS(标记转换系统)对经过扩充的 EHA 定义操作语义。

定义 4(EHA 的操作语义)　一个 EHA H 的操作语义是一个 LTS $T_S=(S, s_0, L, T)$，其中 $S=Conf_H\times\Theta X\times\Omega Y$ 是状况集合，$s_0=(C_0, E_0, V_0)$ 是初始状况，$L: S\to 2^{AP}$ 给每个状况标注一组原子命题，$\to\subseteq S\times S$ 是迁移关系。

T_S 中的一个迁移是 Statecharts 中通过选择最大无冲突迁移集完成的一个 RTC 步[1]。关系"→"通过一组演绎规则进行定义。本文基于文[5]的结果，针对开放模型、变量环境和出口、入口动作等，扩充定义了这些规则。

2　异常处理的模型检验

从模型设计和分析的角度看，基于 UML 的异常分析和建模可以运用层次化构造和抽象的方法来减小模型复杂度。而模型检验技术是分析复杂系统构造是否满足正确和可靠性需求的有效方法。异常处理的模型检验是对模型的可达性、安全和活动性能的分析，本文提出通过自动机代表 Statecharts(状态层、统一分解、内部状态变迁等)的关键模块，自动机具备相应的语法和语义，通过自动转换将 UML Statecharts 映射为自动机[7]，并最终实现模型检验代码自动生成。

一个状态自动机由包含简单状态和变迁的连续自动机组合而成，这些状态代表了 UML 模型的简单和复合状态，可被抽取为任意数量的统一的连续自动机操作。状态图中的复合状态可通过多个单区域自动机状态建模，而一个非并行的复合状态仅能抽取为一个自动机。

自动机状态变迁不能跨越不同的层。在自动机所表达的最底层的复合状态中，包含了变迁的所有源状态和目标状态，而 UML 模型的内部状态变迁将由标签传递所替代。这些标签所连接的源状态和目标状态是明确的。在源状态集中包含有 UML Statecharts 中变迁的源状态，而目标状态集则包含 UML Statecharts 中变迁的目标状态。如果源状态集合中的源态和所有中间态都处于激活状态，而且实际事件满足触发器和关口条件的要求，那么变迁是允许的。在状态变迁时，在目标状态集中的目标态和所有中间态都将被涉及。

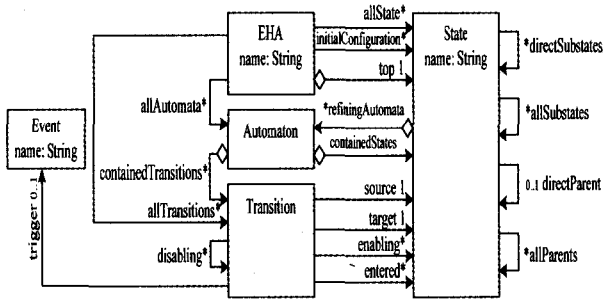

图 3　EHA 的元模型

EHA 作为 UML Statecharts 的中间模型，是 Statecharts 的抽象语法，抽取掉了语法细节，仅保留 Statecharts 的关键部分，并以一种结构化的方式表现出来，它是自动模型转换为模型检验工具的基础。由于本文的方法将异常引入 Statecharts 而不修改 Statecharts 句法或语义，即由基于 EHA 的模型检验使对异常感知的 Statecharts 模型检验成为可能，这将解决两个问题，第一是将决定如何在 EHA 模型中表现异常事件阶层。具体办法就是将事件元类的自协作引入到元模型中同时将概化特征引入到 EHA 事件概念中，但这将要求对 EHA 语义做相应修改并重实现基于 EHA 的模型检验。为了支持遗留模型检验，本文的方法如下：如果一个事件类 E 被细化为事件类 $R1, R2, \cdots, Rn$，那么将来自于 $\{E, R1, R2, \cdots, Rn\}$ 的非抽象事件类触发的变迁替代所有由 E 触发的变迁。这样唯一需要修改的是实现 UML Statecharts 到 EHA 转变中语法的替换。

第二是从异常情况中行为检验的观点来看，UML Statecharts 模型是开放的，因为事件的来源表明了运行时异常(由 Java 虚拟机抛出)和被检查型异常(由程序员编写的例程抛出)丢失了，检验器将相应地关闭模型，因为检验器带有一个异常事件运行时环境的模型。在详尽分析所有可能的 interleaving 和正常事件的情况后，扩展模型的生成就是一个将统一模块引入到 TryBlocks 中的、由异常事件引起的系统过程。这个扩展的自动代码生成是本文的工作目标。

3　代码自动生成

代码的生成模式是由 EHA 定义的行为模型的自动实现，本文的目标是实现一个能映射 EHA 行为到 Java 语言的方法。生成模式不要求任何接口或实现从任何具体基类派生的活动应用类对象。下面通过从 EHA 行为中剥离活动应用类来实现本文的目的。

附加动态行为到特定的 EHA 无状态类之内(包含对应

维普资讯 http://www.cqvip.com

的结构信息和实施动作的所有回调函数等),而应用类(如在例子中的 DataAcq 类) 作为动作的上下文,即应用类为行为类提供功能参数包括特殊用途的变量。这种方法需要应用类显式地存储状态配置信息,因为行为类是无状态的。

表 1 dynamic_behavior 包中实施模式对应的 Java 类

EHA 类	-collectEnabled(Event, Configuation Object) : Transition[] -collectFireable(Transition[]) : Transition[] -RestartTransition(Configuration Object, Transition) : void +dispatchEvent(Event, Configuration Object) : void +initialize(Object) : Configuration
Transition 类	triggerEventClass: Class aasociatedAction(Object) : void guard(Object) : bool
Configuration 类	isActive(State) : bool markActive(State) : void markinactive(State) : void
State 类	entryAction(Object) : void exitAction(Object) : void
Automaton 类	……
Event 类	……

实现事件处理引擎的模式必须包括两部分的转化:基本的支持类是在动态包内,而自动产生的类则是在生成包里,如表 1 所示。

从结构化观点看,如何在 Java 中表示 EHA、自动机、状态、迁移和事件类,上述的类都是 EHA 元模型的对应 Java 等值体,它们的实例及之间的关联都是元模型在 Java 语言上的映射。

从如何插入 Java 编码安全动作等动态应用的观点上看,这些类方法是抽象概念上的点(actions, guards 等),最后被具体编程语言级别上的实现所替代:状态入口和出口动作是状态类的方法(State. entryAction, State. exitAction),而动作与变迁相关联,关口是变迁类的方法(Transition. associatedAction, Transition. guard)。因为动作和关口是对具体状态和变迁的定义,这些类是抽象的,它们的方法将在派生类中实现。当前状态配置由与当前活动状态相关联的配置类表达。所有基于 EHA 行为的活动状态都应该包含在这个类的一个实例中。

转化由 EHA 类实现,dispatchEvent 和 initialize 方法是入口。如同上面讨论到的,活动应用类充当方法的普通参数。initialize 方法用于初始化活动对象的配置。真正的事件派发由 dispatchEvent 功能实现。这个函数作为功能参数被调用,由应用类负责传递事件、配置和自身引用。dispatchEvent 方法首先调用 collectEnabled 方法,并透过 Transitions 实例的关联(充当 allTransitions 角色)收集变迁,然后通过调用 collectFireable 方法从这个集合中去除由优先权关联造成的失活变迁(Transition. disabling 关联)。在执行对 RestartTransition 方法相应的调用期间,因掉电重起而选择的变迁将被执行;这里包括(1) 进行状态退出动作(即调用 State. exit 方法);(2) 进行动作关联变迁;(3) 进行状态入口动作(即调用 State. entry);(4) 更新配置。

在编程人员实现的方法中捕捉可能发生的异常并将其转换为异常事件。将 Java 语言定义的结构(即在 Java 中的 try-catch 块)用于整合表 1 中的功能,转化方法不需要任何涉及到实际行为模式的修改,真正重要和易出错的部分,如建立对象结构表达 EHA 等,由编码发生器自动创建。

结论 本文根据 EHA 结构化地描述 UML Statecharts 及其操作语义,将 Java 异常处理 Statecharts 映射为 EHA 自动机,然后基于自动机理论的模型检验方法以类似 Java 语言异常处理的方式(try-catch-finally 结构)对可能发生的异常能够做出响应,产生相应的代码。由于本文的提出方法不要求对 UML Statecharts 语义做修改,因此遗留模型检验甚至在异常出现时都可用于检查系统的行为,为基于异常感知的状态图源代码自动发生器的实现提供了一个良好的基础。

参 考 文 献

1 Object Management Group. Unified Modeling Language Document Set. Version2.0, OMG document formal /05-07-04. 2005
2 Harel D. Statecharts: a visual formalism for complex system. Science of Computer Programming, 1987, 8 (3) : 231～274
3 Mikk E, Lakhnech Y, Petersohn C. On formal semantics of statecharts as supported by STATEMATE. In: The 2nd BCS-FACS Northern Formal Methods Work shop. Spring-Verlag, 1997
4 Petersohn C. Data and control flow diagram s, statecharts and Z: their formalization, integration and real-time extension [Ph. D. Thesis]. Germany, 1997
5 Latella D, Majzik I, Massink M. Towards a formal operational semantics of UML Statechart diagrams. In: Ciancarini P, Fantechi A, Gorrieri R, eds. Proceedings of the 3rd International Conference on Formal Methods for Open Object-Oriented Distributed Systems. Boston: Kluwer Academic Publishers, 1999. 15～18
6 董威,王戟,齐治昌. UML Statecharts 的模型检验方法[J]. 软件学报, 2003, 14 (4) : 750～755
7 VarróD, VarróG, Patericza A. Checking General Safety Criteria on UML Statecharts. In Lecture Notes in Computer Science, number 2187. Springer Verlag, 2003. 46～55

(上接第 275 页)

参 考 文 献

1 Papazoglou M P, Heuvel W. Service oriented architectures: approaches, technologies and research issues[J]. Very Large Database Journal, 2007, 16(3): 389～415
2 Erl T. SOA Principles of Service Design[M]. Prentice Hall, 2007. 25～66
3 Canfora G, Fasolino A R, Frattolillo G, et al. Migrating interactive legacy systems to Web services[C]. In: Proceedings of the 10th European Conference on Software Maintenance and Reengineering, 2006
4 Reynaud C, Sirot J P, Vodislav D. Semantic integration of XML heterogeneous data sources[C]. In: Proceedings of the 2001 International Symposium on Database Engineering & Applications, 2001. 199～208
5 Abdalla K F. A model for semantic interoperability using XML [C]. In: Proceedings of Systems and Information Engineering Design Symposium, 2003. 107～111
6 Dan A, Johnson R, Arsanjani A. Information as a Service: Modeling and Realization[C]. In: Proceedings of SDSOA, 2007. 2
7 Wang Z, Zhao Z, Fang J. A Service-oriented Approach for Flexible Information Resource Integration [C]. In: Proceedings of COMPSAC, 2007. 573～578
8 Lewis G, Morris E, Smith D, et al. Service-Oriented Migration and Reuse Technique (SMART) [C]. In: Proceedings of 13th IEEE International Workshop on Software Technology and engineering Practice, 2005. 222～229
9 Wang X F, Shawn H, Enamul H, et al. Integrating Legacy Systems within The Service-oriented Architecture[C]. In: Proceedings of Power engineering Society General Meeting, 2007. 1～7
10 Jennex M E. Modeling Emergency Response Systems [C]. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, 2007. 22
11 Turoff M, Chumer M, Walle B V, et al. The design of a dynamic emergency response management information system [J]. Journal of Information Technology Theory and Applications, 2004, 5 (4): 1～36
12 Mendonca D, Beroggi G E G, Wallace W A. Decision support for improvisation during emergency response operations [J]. International Journey emergency Management, 2001, 1(1): 30～38