程序模型检查器综述

林梦香[1] 吴国仕[2]
(北京航空航天大学软件开发环境国家重点实验室 北京 100083)[1]
(北京邮电大学软件学院 北京 100875)[2]

摘 要 模型检查实际程序设计语言编写的程序是近年来程序验证领域的研究热点之一,出现了一批针对C,C++或Java语言的程序模型检查器原型。总结了程序模型检查中的主要问题及相关技术,以是否使用中间建模语言为标准,对现有程序模型检查器进行了分类,并具体地介绍了一些代表性工具中的模型获取及化简技术,最后展望了程序模型检查器未来的研究方向。

关键词 模型检查,程序模型检查,模型抽取
中图法分类号 TP311.5 **文献标识码** A

Survey on Model Checker for Programs

LIN Meng-xiang[1] WU Guo-shi[2]
(State Key Lab. of Software Development Environment, Beihang University, Beijing 100083, China)[1]
(School of Software Engineering, Beijing University of Posts & Telecommunications, Beijing 100875, China)[2]

Abstract Model checking real programs coded with modern programming languages get more and more attention. Several program model checkers for C, C++ or Java programming languages were developed. Model checking programs related main problems and basic methods were analyzed. Model checkers for programs were classified based on whether an intermediate modeling language is adopted. Methods of model extraction and reduction in some typical tools were introduced. Finally, the future trends of program model checking were discussed.

Keywords Model checking, Program model checking, Model extraction

模型检查[1]是一种有限状态系统的形式化验证方法,它通过算法穷尽地搜索系统的有限状态空间,检查系统(模型)的每个状态是否满足预期性质。整个过程是自动的。模型检查技术能够处理并发情况,在系统满足预期性质时能给出证明,反之则提供反例。经过近30年的发展,模型检查理论已基本成熟,实现这些理论的模型检查器成功地应用于数字硬件和协议的设计验证,并作为工业界数字硬件开发质量保证体系的一部分。类似地,人们将模型检查技术应用于软件需求规约及设计的验证中。但是,在这类软件模型检查中,需要手工构造验证系统的模型。建模需要验证专家和设计人员的合作,构造过程繁琐,并可能出现遗漏或引入新错误,随着系统的演化,验证模型也要随之演化。这个复杂的建模过程妨碍了模型检查技术在工程中的应用。将传统的模型检查技术直接应用于实际程序设计语言(如C、C++、JAVA等)编写的程序,可以避开复杂而易错的建模过程。为了与以前的软件模型检查区别,一般称之为"程序模型检查"。

程序模型检查是近年来程序验证领域的研究热点之一,出现了一批程序模型检查器原型。本文总结了程序模型检查中的主要问题及相关技术,从是否使用中间建模语言的角度对现有程序模型检查工具进行了分类,并详细介绍了其中有代表性的工具。

1 程序模型检查技术

一个模型检查理论包括3个方面的内容:验证模型、描述时序性质的形式系统以及模型检查算法。转移系统(transition systems)是验证模型的基本形式,不同模型检查理论框架中的验证模型是转移系统的变型。

经典模型检查理论已基本成熟,实现这些理论的经典模型检查器有SMV[2]和SPIN[3]等。经典模型检查器的建模语言通常是卫式命令语言。与这类语言相比,实际程序设计语言的结构和语义更为复杂,如何从程序文本中导出所需的验证模型是程序模型检查面临的首要问题。一种最直接的方法是用一组程序输入的具体值实际(模拟)地执行程序,通过实际执行得到程序的具体状态空间。实现这种方法的模型检查器被称为具体模型检查器。如果将程序输入的具体值用符号代替,符号执行给定程序,就得到程序的符号状态空间。另外,还可以根据程序语言的等式语义,将程序编码为等式系统,并利用约束求解技术计算出程序的状态空间。

模型检查中固有的状态空间爆炸问题在程序模型检查中更为严重。一般地,程序的状态空间极其巨大甚至是无限的,

到稿日期:2008-05-05 本文受863国家重点项目(2007AA010301)资助。
林梦香(1968-),女,讲师,博士生,主要研究方向为模型检查技术及应用、自动测试数据生成等,E-mail:mxlin@nlsde.buaa.edu.cn;吴国仕(1957-),男,教授,主要研究方向为智能信息处理等。

检查整个状态空间是不可能的，需要将程序状态空间化简为一个实际可处理的有限空间，简化后的空间必须包含检查性质所需的足够的信息。除了传统模型检查中的状态化简技术外，对程序状态空间的化简更多地结合了程序分析中的相关技术，主要方法有：基于各种程序抽象技术计算源程序的抽象程序；利用程序切片减小程序规模；通过限定程序中不确定性的类型及其可能性的数量来获得程序状态空间的一个有限子集。抽象技术最早应用于程序分析中，它通过程序变换对程序变量的数据域进行化简，抽象解释[4]建立了抽象的形式系统。在程序模型检查中，程序中的数据变量是导致状态空间爆炸原因之一。在保持系统的相关行为的前提下，应用抽象技术将实际系统的状态集映射到一个抽象的较小集合，可以有效地缓解状态空间爆炸问题。不确定性带来了多种可能性，是产生状态空间爆炸的另一个原因。程序中不确定性包括：输入和运行环境的不确定、并发环境下并发进程执行顺序的不确定。限定程序不确定性的类型及其可能性的数量，可以获得有限的检查空间。

2 程序模型检查器

长期以来，形式化验证方法因实用性差而备受诟病。研究针对实际程序语言的模型检查器，不仅可以探索如何应用形式化验证方法提高程序的质量，而且可以推动模型检查理论和技术的进一步发展。程序模型检查的首要目标是证明程序满足某些重要的性质，由于可以提供反例，它也可以用于错误检测，并且后者具有更广的应用前景。

关于程序模型检查器的介绍散落在众多的研究文献中，缺乏系统性的介绍。文献[33]从嵌入式系统验证角度，分析了若干面向C语言的模型检查器。本文以是否使用中间建模语言为标准，对现有程序模型检查器进行分类，具体分析了一些代表性工具的模型获取和化简技术。由于篇幅限制，文中对涉及的技术未作深入讨论，对某些技术感兴趣的读者可以进一步阅读相关文献。

2.1 基于中间建模语言的程序模型检查

对实际程序语言各种结构的处理和模型检查算法都是高度复杂的。设计一个中间建模语言，将一个实际程序语言编写的程序变换为一个中间模型，可以重用模型检查算法，使程序模型检查框架具有更好的扩展性。为保证模型的有限性，程序变换中常常结合程序切片和抽象等技术。早期的研究多数使用已有经典模型检查器的建模语言作为中间语言，实现这种方法的工具有：NASA Ames 的 JavaPathFinder1[7]，KSU 的 Bandera 和 H. Holzmann 的 Feaver/modex 等。后续的研究又提出了一些新的中间建模语言，其中代表性的工具有：MSR 的 Slam 和 ZING，Berkeley 的 Blast 等。

2.1.1 Feaver/MODEX 与 Spin

Feaver/MODEX[9]的目标是将程序的实现级描述（C语言）转换到经典模型检查器 Spin 的验证模型，模型检查器 Spin 的建模语言是 Promela。与早期的机械翻译方法不同，Feaver/MODEX 将源程序的控制流结构表示为 Promela 语言的控制流结构，用 Promela 新定义的嵌入原语将源程序中数据声明和基本语句嵌入到验证模型中。为了将嵌入语句转换为 Promela 的语句，同时进行模型的化简，需要用户提供一个规则表，它具体地给出了语法转换规则和抽象规则，抽象规则提供了数据隐藏和谓词抽象等抽象能力。Feaver/MODEX 按照规则表逐句进行语句转换，一些数据域抽象的导出需要借助于某些判定过程。利用规则表还可以限制检查模型的规模。用户在进行程序模型检查时，除了需要给出待检查的性质列表和测试驱动（a test driver）外，还要提供一张用于模型抽取的规则表（a lookup table）。FeaVer/MODEX 中将这 3 个部分组合起来，称为 test harness。

SPIN 及其模型抽取器 FeaVer/MODEX 的验证对象是分布式软件系统中的交互问题，其局限性在于验证模型范围受到用户输入参数的限制[10]，如提供程序输入或每次检查允许的具体特征集。这个问题在其它具体程序模型工具中同样存在，与传统测试中遇到的问题类似。

2.1.2 Slam/Slam/ZING

MSR 的 Slam[14]和 Berkeley 的 Blast[23]中提出了用布尔程序作为 C 程序的验证模型。布尔程序具有与 C 程序相同的控制流结构，但只包含布尔类型的变量。给定一个 C 程序 P 和一个谓词集 E，C2BP 工具通过自动谓词抽象[5,16]，生成一个布尔程序 BP(P,E)。布尔程序构造过程中要反复调用定理证明器。

Bebop[15]是一个（顺序）布尔程序的模型检查器，其中时序安全性质 Φ 表示为有限自动机，用 C 函数实现。将待检查 C 程序 P 与性质自动机函数同步得到一个构造程序 P′。程序 P 满足性质 Φ 当且仅当程序 P′中某个语句标号 L 是可达的，这样程序的时序性质检查就简化为程序 P′中语句是否可达问题。Bebop 使用过程间数据流分析算法来检查布尔程序中语句的可达性，其中，程序的控制流直接表示为控制流图，程序的可达状态集和语句的转换函数间接表示为 BDDs。布尔程序是源程序的过近似（over-approximation），布尔程序中一个出错迹可能不是源程序的实际运行迹，需要通过路径模拟技术分析其可行性。若出错迹可行，表明源程序违反性质 Φ，反之则说明出错迹是抽象导致的假反例（spurious counter-examples）。出现假反例后，需要以某种方式调整谓词集，以便从抽象程序中消除这个反例。重复这个过程，直至没有假反例或性质成立。这个过程称为反例指导的抽象精化（CEGAR）[6]。BLAST 用懒抽象（lazy abstraction）方法[22]优化了 Slam 中简单的抽象-检查-精化迭代过程。BoPPo[25]结合基于 SAT 的符号模型检查和偏序化简，实现了异步多线程布尔程序的模型检查。

ZING[24]的目标与 Slam 类似，但它的模型不是简单的布尔程序，而是一种具有共享内存和消息传递机制的异步交替的并发模型。ZING 建模语言除了包含顺序流、分支和循环，还支持具有调用栈的过程调用、动态分配对象、动态创建进程、共享内存和消息传递的进程间通信。ZING 的基础结构包括模型抽取和模型检查两个部分，其中模型抽取分为两个阶段：从实际程序中抽取 ZING 语言程序，并将 ZING 程序转换为一个标识转移系统的 ZING 对象模型（ZOM）。模型检查在对象模型 ZOM 上进行。

2.2 直接生成验证模型的程序模型检查

程序语言与中间建模语言的语义差异往往限制了对实际程序的处理能力。同时，程序模型检查中，对代码的任何修改、变换都可能导致假报错，或者错误遗漏；模型检查中触发的实际代码越多，可以发现的错误越多[26]。从被检查的源程

序中直接生成验证模型，可以有效地避免上述问题。在具体实现策略上，NASA Ames 的 JPF2 直接处理程序的中间码而不是源程序本身，绕开了复杂的源语言结构；CMU 的 MAGIC 在生成模型之前对源程序进行了抽象等变换，以减小检查程序的规模；斯坦福大学的 CMC 用实际（模拟）执行方法得到所需的验证空间；NASA Ames 的 JPF-SE 扩展 JP2 实现符号执行；CMU 的 CBMC 将源程序直接编码为一个等式系统。

2.2.1 JPF2

JavaPathFinder2（JPF2）[11] 是 NASA Ames 研制的第二代 Java 程序模型检查器，它直接检查 Java 的字节码。JPF2 的核心是一个定制的执行字节码的 Java 虚拟机（JVMJPF）以及遍历程序状态图的深度优先算法，JVMJPF 支持所有 Java 字节码指令，因此 JPF2 可以处理 Java 语言的所有特征。与 SPIN 一样，JPF2 是一个具体状态模型检查器。为了减少模型检查空间，JPF2 中综合应用了对称化简、偏序化简、静态切片、程序抽象和运行时分析多种技术[12]。具体实现上，解决了 Java 程序中类加载、对象分配和垃圾搜集中的对称化简；通过静态偏序计算获得偏序化简所需的信息；对于用户定义的抽象谓词，调用一个判定过程生成源程序的一个抽象程序；利用 JVMJPF 模拟器搜集数据竞争和锁顺序冲突的信息，用这些信息指导模型检查器的搜索。

BANDERA[8] 是一个将 Java 源程序翻译为经典模型检查器（如 SMV 和 Spin）语言程序的工具集，它实现了并发 Java 程序的静态切片和数据抽象。BANDERA 规约语言（BSL）可以表示线性时序逻辑（LTL）性质以及方法的前条件和后条件。为了提高可用性，JPF2 实现了与 BANDERA 的集成，集成后的 JPF2 通过 BANDERA 前端输入检查性质，显示错误路径，同时利用 BANDERA 中的切片和抽象工具减小被检查程序的规模。JPF2 可以检查死锁、不变量、代码中用户定义断言以及线性时序逻辑性质（LTL）。为了处理 LTL 性质，JPF2 还实现了一个从 LTL 到 Büchi 自动机的前端翻译器。

2.2.2 MAGIC

CMU 的 MAGIC[13] 采用谓词抽象从 C 程序的控制流图中抽取标识转移系统（Labeled Transition systems）LTS。LTS 与 Kripke 结构类似，不同之处在于用动作标识转移。一个谓词定义了程序变量（集）上的一个约束，给定一个程序和一组谓词，谓词抽象算法通过跟踪这些谓词来抽象数据。在抽象程序中，每个谓词表示为一个布尔变量，源程序中的数据变量被消除。与 Slam 一样，抽象程序是源程序的一个过度近似。这样，为了证明没有出错状态是可达的，只需证明在抽象模型中出错状态不可达。MAGIC 同样遵循 CEGAR 模式消除假反例，如果规约不满足抽象模型，就检查反例的有效性，对于假反例构造其解释并精化抽象模型。MAGIC 中用弱模拟（weak simulation）作为规约与抽象模型一致性概念，并将弱模拟简化成一个判定布尔公式可满足问题。

2.2.3 CBMC

当系统具有无穷多个状态转移或状态转移数目非常大时，计算并访问所有状态几乎是不可能的。有界模型检查（bounded model checking）[18] 通过有限次展开转移来解决这个问题，转移展开次数称为界。当展开界足够大且趋于无穷时，系统的有界模型等于无界模型。CMU 的 CBMC[19] 将有界模型检查的方法应用到顺序 C 程序的模型检查。

通过程序的静态单赋值形式（SSA），CBMC 巧妙地用布尔公式编码程序的（有限的）所有可能运行迹。由于无法静态确定程序中循环的执行次数，CBMC 将循环进行有限次展开，展开次数可以设定为 0 或 1，也可由用户给定。程序到布尔公式的主要变换过程如下：

• 程序预处理。用等价赋值替换副作用（side effects），用等价的 goto 语句替换 break 和 continue 语句，用等价的 while 循环替换 for 和 do while 循环语句，用等价的 if 和 goto 语言替换 switch 语句。将循环结构进行有限次展开，展开界为 n。当展开次数小于 n 时，加入 if 语句，if 语句中条件与结构原条件相同。最后一次展开后，加入断言语句，断言为原条件的否定。

• 变量改名。经过预处理后，程序中只包含赋值、if 语句、前向 goto 语句和标号及断言语句。对于程序某个位置处变量 V，如果在该位置前变量 V 的赋值次数为 m，将该变量改名为 Vm。一个变量赋值后，其赋值次数加 1。改名时同时将 goto 语句替换为等价的 if 语句。改名后的程序是一个静态单一赋值程序，即每个变量最多赋值一次。

• 对于仅由赋值和条件语句组成的程序，根据等式语义，将每个赋值语句变换为一个位向量等式。程序语句得到位向量等式 C，检查性质得到另一个位向量等式 P，将形如 $C \wedge \neg P$ 的布尔公式转换为 CNF，输入到一个 SAT Solver。公式 $C \wedge \neg P$ 是可满足的，当且仅当存在一个违反给定性质 P 的运行迹，该运行迹就是一个反例。

CBMC 支持几乎所有 ANSI-C 语言特征，包括指针、动态内存分配、浮点和双精数据类型、递归。有界模型检查中的展开界由用户指定。如果存在比展开界长的反例，这时它只是作为一个错误发现的工具，而不能证明正确性。

2.2.4 CMC

基于实际执行的程序模型检查通过实际（模拟）执行实际代码的方法生成程序的状态空间，避免了因使用抽象技术对源程序的改变和限制。Verisoft[20] 最早应用这个方法检查实际程序设计语言编写的程序。为了避免状态空间爆炸问题，Verisoft 在程序执行过程中不保存状态，这样就难以避免重复搜索。CMC[27] 是一个面向 C 或 C++ 程序实现级的模型检查器，最初用于网络协议实现中的错误检测，类似于一个网络模拟器。应用 CMC 之前要建立用户模型和环境模型，它们分别建模外部输入和环境。CMC 从一个初始状态开始，通过系统地模拟执行转移（语句）递归地生成和检查后续状态。可以检查的性质包括一般性质（如内存泄漏、访问不合法内存）和应用相关性质，应用相关性质可以用断言或布尔函数（C 实现）描述。与 Verisoft 不同的是，CMC 保存检查过的状态以避免重复搜索，同时在搜索中加入回溯机制，处理环境中的不确定性。基于 CMC、面向文件系统的模型检查器 FiSC[28] 检查了三个被广泛应用和测试过的文件系统，共发现了 32 个严重错误。

2.2.5 JPF-SE

符号执行由 KING[29] 在程序测试研究中引入。符号执行中用符号值而不是具体值表示程序输入，符号值代表了输入变量取值域中的任意一个。符号执行中，程序变量的值是一个关于符号输入的表达式，程序状态因而是一个符号状态。

一个符号状态是一个具体状态的集合，对一个巨大甚至是无限的系统状态空间，符号地表示状态集合可以缓解状态空间爆炸的问题。

NASA Ames 的 JPF-SE[32] 扩展了具体模型检查器 JPF2，使之可以进行 Java 程序的符号执行。具体方法上，JPF-SE 要求对源程序插装[30]：具体类型被相应的符号类型代替，具体操作被相应的符号表达式上的操作代替。符号执行中，只要路径条件被更新，就调用一个合适的判定过程，检查其可满足性。如果 PC 不可满足，则搜索回溯。为了有效求解符号数值约束，JPF-SE 实现了与 STP 等著名判定过程的接口。

对于可能导致无限多个符号状态的循环结构，JPF-SE 采用两种技术解决这个问题：一是设定程序输入或搜索深度的界；二是自动抽象和比较符号状态。如果确定一个符号状态已被访问过，则搜索回溯[31]。

JPF-SE 用程序的符号状态代替具体状态，在实现上借助于具体模型检查器实现程序的符号执行，充分利用了标准模型检查的各种搜索技术和状态化简技术，代价是要对源程序进行转换。

结束语 将模型检查等形式化验证技术应用于实际程序面临巨大的挑战，根源在于实际程序设计语言的复杂性及程序规模。本文分析了一些代表性程序模型检查器在解决这些问题中采用的策略及具体技术。尽管现有的程序模型检查工具在实际应用中取得了一些成果，但距离工程中的应用还有相当的距离。未来的研究方向有：

• 处理更多的语言特征和一般类型的程序。

从经典模型检查器的卫式命令语言到实际程序设计语言的某个子集，模型检查器能够处理的语言越来越复杂。目前程序模型检查器主要应用于事件驱动等控制依赖的程序。今后的程序模型检查工具应能处理更多的实际语言特征和一般类型的程序。

• 多种技术的进一步融合。

为了减小检查模型的规模，程序模型检查中结合程序切片、抽象、运行时分析等多种技术。从程序分析与验证领域看，出现了来自测试、程序分析、形式验证等不同领域的技术相融合的趋势[17]，最新的研究[21]综合地应用了不同领域的技术。

• 进一步提高处理程序的规模。

目前程序模型检查器能够处理的程序规模远远不能满足实际要求，需进一步研究程序模型化简技术，并结合分布存储和并行处理等技术，提高模型检查器处理程序的规模。

参 考 文 献

[1] Clarke E, Grumberg O, Peled D. Model Checking. MIT Press, 1999

[2] McMillan K L. Symbolic Model Checking. Kluwer Academic Publishers, 1993

[3] Holzmann G J. The model checker SPIN. IEEE Trans. on Software Engineering, 1997, 23(5): 279-295

[4] Cousot P, Cousot R. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints// Proc. POPL'77. 1977: 238-252

[5] Graf S, Saidi H. Construction of abstract state graphs with PVS //Grumberg O. ed. CAV, Springer Verlag, volume 1254, 1997: 72-83

[6] Clarke E M, Grumberg O, Jha S, et al. Counterexample-guided abstraction refinement// Proc. CAV'00. 2000: 154-169

[7] Havelund K, Pressburger T. Model Checking JAVA Programs Using JAVA PathFinder. International Journal on Software Tools for Technology Transfer, 2000, 2(4): 366-381

[8] Corbett J C, Dwyer M B, Hatcliff J, et al. Bandera: Extracting finite-state models from Java source code// Proc. ICSE'00. ACM, 2000: 439-448

[9] Holzmann G. Logic Verification of ANSI - C Code with Spin // Proc. of the 7th International SPIN Workshop. Springer Verlag, LNCS1885, 2000: 131-147

[10] Holzmann G, Smith M. An automated verification method for distributed systems software based on model extraction. IEEE Trans. on Software Engineering, 2002: 364-377

[11] Brat M, Havelund K, Park S, et al. Java PathFinder -A second generation of a Java model checker// Proc. of the Workshop on Advances in Verification. Chicago. July 2000

[12] Brat G, Havelund K, Park S, et al. Model checking programs// Proc. ASE'00. Grenoble, France, 2000

[13] Chaki S, Clarke E M, Groce A, et al. Modular Verification of Software Components in C. ACM Trans. Computer Systems, 2004, 30(6): 388-402

[14] Ball T, Rajamani S K. The Slam project: Debugging system software via static analysis// Proc. POPL'02. ACM, 2002: 1-3

[15] Ball T, Rajamani S K. Bebop: A symbolic model checker for Boolean programs// Proc. SPIN'00. 2000: 113-130

[16] Ball T, Majumdar R, Millstein T, et al. Automatic predicate abstraction of C programs// Proc. PLDI'01. 2001: 203-213

[17] Yorsh G, Ball T, Sagiv M. Testing, abstraction, theorem proving: Better together// Proc. ISSTA'06. 2006

[18] Biere A, Cimatti A, Clarke E M, et al. Symbolic model checking without BDDs// Proc. TACAS'99. Springer-Verlag, LNCS1579, 1999: 193-207

[19] Clarke E, Kroening D, Lerda F. A tool for checking ANSI-C programs// Proc. TACAS'04. Springer, 2004: 168-176

[20] Godefroid P. Model Checking for Programming Languages using VeriSoft// Proc. POPL'97. ACM, 1997: 174-186

[21] Godefroid P, Klarlund N, Sen K. Dart: Directed automated random testing// Proc. PLDI'05. ACM, 2005: 213-223

[22] Henzinger T A, Jhala R, Majumdar R, et al. Lazy abstraction// Proc. of POPL'02. ACM, 2002: 58-70

[23] Beyer D, Henzinger T A, Jhala R, et al. The software model checker BLAST applications to software engineering. International Journal on Software Tools for Technology Transfer, 2007, 9: 505-525

[24] Andrews T, Qadeer S, Rajamani i S K, et al. Zing: A model checker for concurrent software // Proc. CAV'04. Springer, LNCS 3114, 2004: 484-487

[25] Cook B, Kroening D, Sharygina N. Symbolic model checking for asynchronous Boolean programs// Proc. SPIN'05. 2005: 75-90

[26] Engler D, Musuvathi M. Static analysis versus software model checking for bug finding// VMCAI'04. Springer, LNCS 2937, 2004: 405-427

(下转第 41 页)

(1)更新监听

建立更新监听器,监听局部智能主题地图的动态变化。

(2)更新信息的传输

监听器监听到局部主题图发生变化后,采用 SOAP 实现分布式系统中主题图同步数据的传输。SOAP(Simple Object Access Protocol),简单对象访问协议,是分布式环境中交换信息的简单的协议,是基于 XML 的协议,包括 4 个部分:

SOAP 封装(Envelop):定义一个描述消息中的内容。

SOAP 编码规则(Encoding Rules):表示应用程序需要使用的数据类型的实例。

SOAP RPC 表示(RPC Representation):表示远程过程调用和应答的协定。

SOAP 绑定(Binding):使用底层协议交换信息。

主题地图动态更新的数据传输如图 4 所示。

图 4 动态更新的数据传输示意图

SOAP 信封(SOAP Envelope)是一个包含 Header 和 Body 内容的 XML 文档。

信息结构如下所示:

```
〈soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"〉
    〈soap-env:Header/〉
        〈soap-env:Body〉
            〈students:GetAllStudents xmlns:students="http://ctec.xjtu.edu.cn"〉
                〈students:name〉王飞〈/students:name〉
            〈/students:GetAllStudents〉
        〈/soap-env:Body〉
〈/soap-env:Envelope〉
```

(3) 全局主题图的更新

当接收到更新信息后,对全局主题图进行动态更新。通过描述源主题图的指定更新部分和目标主题图的拟更新部分,使用源主题图的 XML 描述以及转化的定义描述(convert.xslt)来产生更新以后的主题图,如图 5 所示。

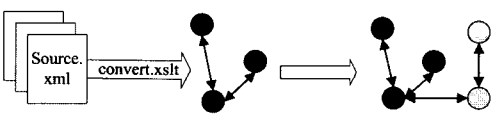

图 5 主题图更新示例

结束语 主题图融合是将主题图融入 Web 2.0 的关键技术之一,但主题图融合技术尚处于理论研究阶段,要将其实际应用到 Web,需要解决的问题还很多,如基于语义主题的相似性算法、分布式环境下主题图的逻辑融合、融合冲突的检测和消除以及主题图动态更新的协议和更新算法等等,有待我们去深入的探索和研究,主题图融合技术可以广泛应用到 Web 信息组织、信息管理、信息查询与搜索、信息集成以及构建信息导航地图等,有着极其广阔的应用前景。

参 考 文 献

[1] 朱良兵. Topic Maps:撬动 Web2.0 的语义杠杆[J]. 图书馆杂志,2007,26(5):48-51

[2] Pepper S. The TAO of Topic Maps [EB/OL]. http://www.gca.org/papers/xmleurope2000/,2000-06-12

[3] 朱良兵,纪希禹. 基于 Topic Maps 的叙词表再工程[J]. 现代图书情报技术,2006,141:81-84

[4] 田磊,覃征,衡星辰,等. 基于本体的多源异构 XML 数据近似查询方法[J]. 西安交通大学学报,2007,41(6):702-706

[5] ISO/IEC 13250 Topic Maps:Information Technology Document Description and Processing Languages. 2nd ed. 2002[S]. http://www.y12.doe.gov/sgml/sc34/document/0322files/iso13250-2nd-ed-v2.pdf,2002-06-16

[6] Pepper S. OASIS Published Subjects:Introduction and Basic Requirements[EB/OI]. http//www.oasis-open.org/committees/geolang,2005-03-23

[7] Garshol L M, Moore G. Topic Maps-Data Model [EB/OL]. http://www.isotopicmaps.org/sam/sam-model/,2004-02-16

[8] Vatant B. Published subjects:from Confucius to topic maps and beyond [J]. Interchange,2002,39(1):25-32

[9] Maicher L, Witschel H F. Merging of Distributed Topic Maps based on the Subject Identity Measure (SIM) Approach. Department of Information Sciences, University of Leipzig, Chair of NLP, Augustusplatz 10-11, 04109 Leipzig, Germany, Sept. 2004

[10] 吴笑凡,周良,张磊,等. 分布式主题图合并中的 TOM 算法[J]. 武汉大学学报,2006,39(5):131-136

[11] Jung M K, Hyopil S, Hyoung J K. Schema and constraints-based matching and merging of Topic Maps[J]. Information Processing and Management,2007,43(4):930-945

[12] 吴克河,马应龙,林鹏程,等. 一种基于本体的语义冲突处理方法[J]. 计算机工程与应用,2007,43(13):182-185

[13] 聂志强. 语义冲突及冲突处理模型的设计[J]. Science & Technology Information,2007,2:491

(上接第 15 页)

[27] Musuvathi M, Park D Y, Chou A, et al. CMC: A pragmatic approach to model checking real code // Proc. OSDI'02. ACM, 2002:75-88

[28] Yang J, Twohey P, Engler D, et al. Using model checking to find serious file system errors // Proc. OSDI'04. San Francisco, 2004

[29] King J C. Symbolic execution and testing. Comm. of the ACM, 1976,19(7):385-394

[30] Khurshid S, Pasareanu C S, Visser W. Generalized symbolic execution for model checking and testing // Proc. TACAS'03. Springer, LNCS 2619, 2003:553-568

[31] Anand S, Pasareanu C, Visser W. Symbolic execution with abstract subsumption checking // Proc. SPIN'06. LNCS 3925, 2006:163-181

[32] Anand S, Pasareanu C S, Visser W. JPF-SE: A Symbolic Execution Extension to Java PathFinder // Proc. of TACAS'07. Springer-Verlag, LNCS 4424, 2007:134-138

[33] Schlich B, Kowalewski S. Model Checking C Source Code for Embedded Systems // IsoLA'05. Columbia, MD, USA, 2005