关于软件需求中的不一致性管理*

朱雪峰[1,2+], 金 芝[1,3]

[1](中国科学院 计算技术研究所,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](中国科学院 数学与系统科学研究院,北京 100080)

Managing the Inconsistency of Software Requirements

ZHU Xue-Feng[1,2+], JIN Zhi[1,3]

[1](Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100080, China)

[3](Academy of Mathematics and System Sciences, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62565533 ext 5668, E-mail: zhuxuefeng@ict.ac.cn, http://knowledge_science.ict.ac.cn

Received 2004-10-10; Accepted 2005-02-03

Zhu XF, Jin Z. Managing the inconsistency of software requirements. *Journal of Software*, 2005,16(7): 1221–1231. DOI: 10.1360/jos161221

Abstract: Analyzing and handling the inconsistency in requirements is critical to the development of complex software systems. How to solve this problem directly influences the quality of requirements specification, as well as the final software production. Based on a common-recognized management framework of requirements inconsistency, this paper summarizes representative researches on this topic. The main aim is to get a comprehensive understanding of the current techniques and methods of inconsistency management. Finally, the paper also identifies some issues which are still open to further research.

Key words: requirements engineering; requirements specification; management of requirements inconsistency

摘 要: 复杂软件系统开发的一个关键问题是分析和处理可能存在的不一致的需求描述.这个问题解决得好坏直接影响到需求规格说明的质量,进而影响到最终软件产品的质量.在目前公认的一个不一致需求管理框架的基础上,就需求不一致性管理方面的有代表性的工作,进行了较为系统的分析,以期建立对当前需求工程中,关于不一致的需求管理方法和技术的全面认识.最后,对需求不一致性管理方面的研究进行了展望.

关键词: 需求工程;需求描述;需求不一致性的管理

中图法分类号: TP18 文献标识码: A

* Supported by the National Natural Science Key Foundation of China under Grant No.60233010 (国家自然科学重点基金); the Grand Project of the National Natural Science Foundation of China under Grant No.60496324 (国家自然科学基金重大项目); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312004 (国家重点基础研究发展规划(973)); the Knowledge Innovation Program of the Chinese Academy of Sciences (中国科学院知识创新工程项目)

作者简介: 朱雪峰(1973－),男,河南睢县人,博士生,主要研究领域为需求工程;金芝(1962－),女,博士,研究员,博士生导师,CCF高级会员,主要研究领域为人工智能,需求工程,基于知识的软件工程.

在人类社会中,知识和信息的不一致性是普遍存在的.而软件系统的需求来自于人类对现实社会的知识和期望[1],从有可能不一致的知识或期望中归结出来的软件需求也不可避免地存在不一致的可能性.不一致需求是普遍存在的这个观点目前已经得到广泛的共识[2−4].另一方面,人类社会允许矛盾的知识和信息以不同的形式出现在不同的人、甚至同一个人的身上,而不需要事先进行特别的处理,一般都是在矛盾发生时才予以考虑.但软件系统就其本质而言是一个形式系统,一旦确定后,机器只会照章办事,因此必须事先确保软件系统的需求描述是完备、确定和无矛盾的,否则会导致软件系统运行时错误,直接影响系统的安全性和可靠性.如何管理需求中的不一致性,已经成为软件工程,特别是需求工程中的一个重要问题.

目前,已经存在一些关于需求不一致性管理方面的工作,综合起来看,可以分为两大类.第 1 类是讨论一般意义上的需求不一致性管理框架,包括一般的管理原则和主要活动.第 2 类则结合具体的需求建模方法,讨论具体的需求不一致性分析和处理策略.

本文首先介绍一般意义上的需求不一致性管理框架,然后再结合具体的需求建模方法分析不同的需求不一致分析和处理策略,由于需求不一致产生的根本原因之一,是需求描述元素的指称域之间存在隐含的重叠关系,因此本文还在第 3 节讨论了当前关于描述元素指称域重叠的检测的工作.最后,文章分析了一些代表性方法的特征和优缺点,并探讨了需求不一致性管理的发展趋势.

1 不一致的需求管理框架

对需求不一致性的管理,还没有公认为系统性的解决方案.就目前的研究现状来看,Nuseibeh 等人的工作[5]可以说是具有代表性的,也是最为全面的.他们提出了一个需求不一致性的管理模式,如图 1 所示,其基本思路为,任何需求建模方法都隐含了一组维护其一致性的规则,当所获取的需求中包含了不满足这组一致性规则约束的命题时,即可认为出现了不一致性,需要进行进一步的处理.

在这个框架中,一致性规则[6]起到了至关重要的作用,因为是否出现了需求不一致性完全是相对于这些规则而言的.那么,什么可以作为一致性规则呢?这个管理框架并没有明确规定.采用什么作为一致性规则,完全依赖于系统分析员当前的关注点.例如文献[7]曾列举如下的一些可能的一致性规则:

1. 良构性规则:语言的合法性,即需求的表示要符合所采用的描述语言的语法;
2. 描述同一性规则:要求需求描述中完全重叠的不同描述元素必须是同一的;
3. 应用领域规则:说明在该软件的应用领域中个体之间应满足的关系;
4. 开发兼容性规则:要求软件开发不同阶段的模型要遵循相同的约束;
5. 开发过程柔性规则:项目所遵循的特定的软件工程实践或标准的模型具有柔性.

在上述规则实例中,有些是可以进行自动处理的,如规则 1 和规则 2;有些是不能进行自动处理的,如规则 4 和规则 5;而如规则 3 这样的规则是否能够进行自动处理,依赖于需求描述和一致性规则的形式化程度.

这个不一致性管理框架还明确指出了需求不一致性管理中的任务,主要包括:

(1) 需求不一致性的检测和诊断,包括不一致性的识别、定位和分类;

(2) 需求不一致性的处理,处理策略包括立即消除、暂时容忍和完全忽略;

(3) 需求不一致性的度量以及对需求不一致性的影响和风险评估.

这个框架可以说是普适的.但是它只给出了一个总体的结构和过程,其对不一致需求处理的能力,比如说能够管理哪种形式(如语法上或语义上)的不一致性以及管理到何种程度(如自动化程度等),则完全依赖于所预定的一致性规则以及需求的表示形式及其形式化程度.

另一类相关研究工作是多视点需求建模的不一致性管理[8],多视点方法从不同视点对系统进行了划分,它将系统整体需求划分为多个相互独立的局部视图,并使用不同的手段对多个视点分别进行建模,以期获得尽可能完备的系统需求.需求不一致性管理在多视点需求建模方法中格外重要,这是因为多视点方法用视点划分实现了系统的分解,要求从多个视点进行系统建模,在最后进行系统视点集成的时候,分别建模的多个视点模型之间常常蕴涵不一致的信息,必须考虑这些不一致信息的管理问题.多视点需求建模中的不一致性管理主要考虑的也是上述几个方面的问题.

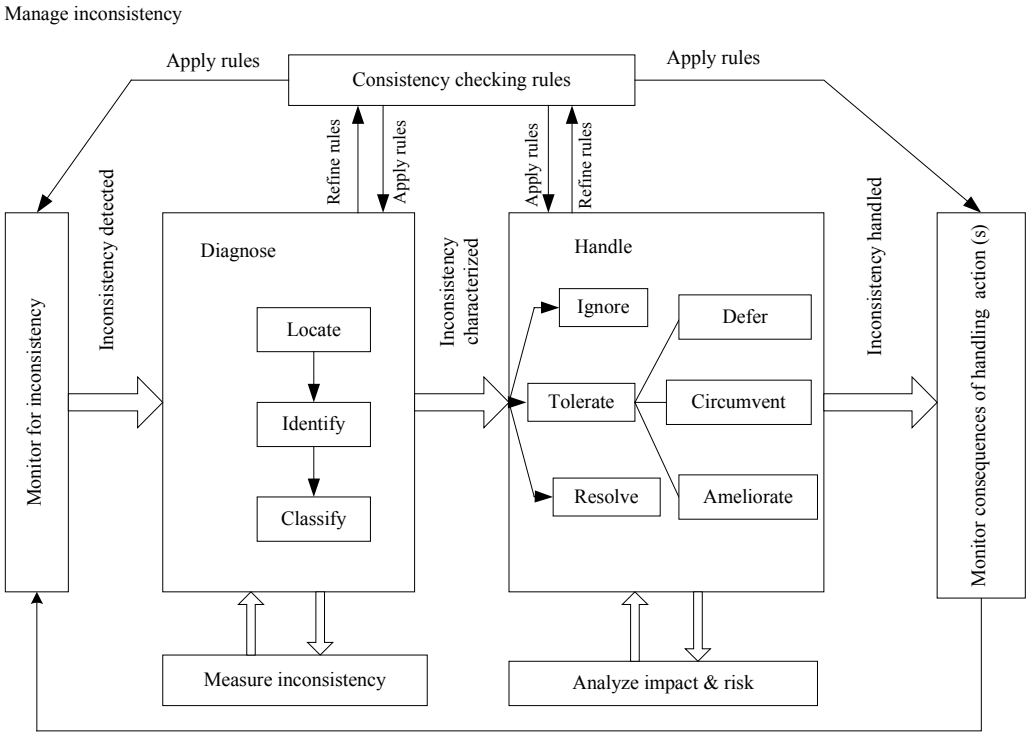

Fig.1 Rule-Based management framework of requirements inconsistency[5]

图 1 基于一致性规则的需求不一致性管理框架[5]

2 需求不一致性处理的主要方法

自动(或半自动)的需求不一致性分析和处理方法,与需求建模的原则和需求模型的表示形式密切相关.目前有代表性的研究工作可分为 3 类,第 1 类,采用经典逻辑或准经典逻辑作为需求表示形式,并利用定理证明技术进行需求不一致性的处理;第 2 类,采用状态变迁作为需求建模原则,并利用模型检验技术进行需求不一致性的处理;第 3 类,以系统目标这个元概念作为需求建模原则,并利用目标的语义模式和关于目标的启发式规则进行需求不一致性的处理.

2.1 基于定理证明的方法

此类方法的主要特征是:假设任何需求描述都可以表示为逻辑断言,采用经典逻辑、时态逻辑或其他非经典逻辑等作为需求描述的形式化表示形式,利用定理证明的手段,采用标准逻辑推理规则,如归结、联结、否定消除、全称量词的实例化,以及其他一些类似于封闭世界假设这样的规则,找出需求描述中蕴涵的矛盾.若检测出矛盾,即表明存在不一致的需求描述.这种方法的优点是能够支持自动的需求不一致性检测,并具有良定的检测过程和语义.但是定理证明在计算上的低效率妨碍了技术本身的可行性,此外,一阶逻辑的半可判定性也限制了这种技术的实用性.

按处理能力来分,这种方法又可以进一步分为基于经典逻辑的方法和基于准经典逻辑的方法两种.

2.1.1 基于经典逻辑的方法

早期采用经典逻辑进行需求不一致性检测的工作[9,10],典型的是以 CORE[11]作为需求获取框架的.在 CORE 框架中,需求分为两个部分:AS(Agent structuring)阶段产生的全局 Agent 层次(如文献[9]中的图 6)以及 TC(table collection)阶段产生的活动表(如文献[9]中的图 4)的集合.

在进行需求不一致性检测之前,首先需要为 CORE 框架定义一组一致性规则,例如:

• 视点内一致性规则:活动表中的任何“源(source)”和“目的地(destination)”都必须在 Agent 层次上作为一个叶子 Agent 出现.

• 视点间一致性规则:由 Agent(X)的一个活动表产生的到目的地(Y)的输出(Z),必须作为(Y)的某个活动表的来自源(X)的输入(Z).

具体处理过程是,首先将 CORE 活动表所表示的活动定义为如下形式的事实公式:

table(Source,Input,Action,Output,Destination)

构成活动事实库,再将所规定的一致性规则表示为对应的逻辑蕴涵式.然后,利用带封闭世界假设的经典逻辑定理证明器,来验证这组逻辑公式集合的一致性.若出现矛盾的推论,则表明这组需求描述中蕴涵矛盾.由于经典逻辑在出现不一致的情况下就无法再进行推理,因此需求一致性检测工作到此即结束了.对所存在的不一致需求的定位、诊断和处理都需要依靠人工完成.

2.1.2 基于带标记的准经典逻辑的方法

由于在存在矛盾的断言集中,经典逻辑会产生毫无意义的结论.因此,一旦检测到存在不一致的需求,不一致性的处理工作就无法继续进行下去.而越来越多的实践表明,很多情况下要求(暂时)容忍不一致的需求描述[5].为了能够暂时容忍不一致的需求描述,并跟踪不一致性产生的根源,文献[12]提出用带标记的准经典逻辑(labeled quasi-classical logic)来进行需求不一致的检测.

就表示形式而言,准经典逻辑与经典逻辑的区别,在于准经典逻辑严格区分了正原子公式和负原子公式,从而在推理过程中能够容忍经典逻辑意义下的矛盾.带标记的准经典逻辑与准经典逻辑具有同样的表达能力,其不同之处在于,带标记的准经典逻辑子句集中的每个子句前面都附带了一个标号的集合,即,若α为一个准经典逻辑的子句,而ξ为一原子符号集合(例如字母表),如果 $i\in\xi$,则 $i{:}\alpha$为一个带标记的准经典逻辑子句.

就推理能力而言,带标记的准经典逻辑与经典逻辑的主要区别在于:

(1) 不利用封闭世界假设,只推断在原子公式集中直接包含的原子公式,并允许同一原子公式的正负形式同时出现在一个模型中,阻止对形为$\alpha\wedge\neg\alpha$的公式进行推理,即禁止所有的平凡推理;

(2) 扩展可满足性关系,分别定义强可满足性和弱可满足性,其中强可满足性阻止在析取推理中引入无意义的原子公式;

(3) 对每个子句附加标号的集合,同一标号集合下子句的归结保持标号集合不变,不同标号的子句归结,除进行子句的归结以外尚需进行标号集合的合并.

以上 3 点措施使带标记的准经典逻辑不会产生无意义的结论,从而能够容忍经典逻辑意义下的矛盾(由上述(1)和(2)),同时,子句所带标号的集合记录了产生矛盾的根源(由(3)).

文献[12]进一步提出了对不一致的需求描述进行处理的方法.当检测到存在不一致的需求描述时,可以进行 3 类不产生变化的行为:

(1) 确定不一致的性质:从全体需求描述出发,构造最大一致断言集的集合,根据每个断言和最大一致断言集之间的关系,将断言划分为 3 类,即存在推断、全称推断和自由推断*;

(2) 识别不一致性的来源:由于需求描述采用带标记的逻辑公式,推理过程采用的也是带标记的准经典逻辑推理规则,推理过程中的任何结论都存在标记的集合作为该结论的标记,标记的集合即记录了需求不一致性的来源;

(3) 定义一些元级规则来给出需求不一致性处理的建议:如,不一致的需求出现在全称推断中必须要去除,出现在存在推断中则必须谨慎处理.

*由一个最大一致断言集推出的断言称为存在断言,由所有最大一致断言集推出的断言称为全称推断,由最大一致断言集的交集推出的断言称为自由断言.

2.2 基于模型检验的方法

模型检验是一种验证有限状态并发系统的形式化方法[13],这种系统由一组系统状态变量来刻画,这组系统状态变量的不同取值表示系统的一个可能的状态.系统的需求则表示为随着系统的执行,系统的可能状态之间的变迁关系.

形式化地说,一个状态变迁系统$\langle Q,R,I\rangle$由一组状态 Q、状态变迁关系 $R\subseteq Q\times Q$ 以及初始状态集 $I\subseteq Q$ 组成.路径是状态的无限序列,使得每个相邻状态对都属于 R.状态集合 Q 一般由一组状态变量编码,使得 Q 到变量值的映射是一对一的.目前最为常见的模型检测工具是 SMV[14]和 SPIN[15].

文献[16]是最早研究模型检验在检测需求规格说明中与应用相关的错误的工作,文献[17]则描述了一个形式分析技术,称为一致性检查技术,使用模型检验技术自动检测需求描述中的错误.文献[18]中给出了一个基于模型检验技术的需求不一致性处理框架,如图 2 所示.采用模型检验进行需求不一致检测的主要特征是:需要将系统的需求表示为特定的面向状态的语言,然后用面向这种语言的状态计算工具来验证各种性质.其优点是,具有与基于定理证明一样的良定的检测过程和语义;缺点是,可能会由于状态空间的指数性增长而影响算法的效率,同时只能检测出某些特定类型的不一致性.

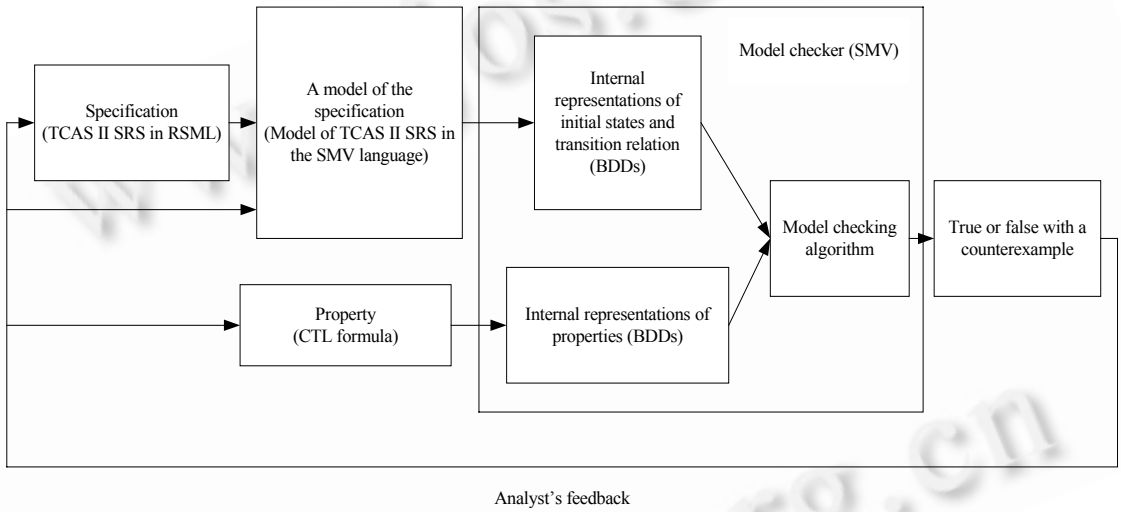

Fig.2 Inconsistency handling framework of requirements based on model checking[18]

图 2 基于模型检验技术的需求不一致性处理框架[18]

2.2.1 SCR 中的需求不一致性处理

这个方面的一个典型工作是 SCR(software cost reduction)方法[19],其主要思想是软件系统的四变量模型,即软件系统的行为可以由 4 个变量(即监测变量、控制变量、输入数据和输出数据)之间的一组关系精确刻画,其中监测变量和控制变量分别是影响系统行为的环境变量和系统所控制的环境变量.用两个关系集 *REQ* 和 *NAT* 描述环境,其中 *NAT* 包含关于监测变量和控制变量之间的约束;而 *REQ* 则定义要建立的系统的其他约束,即针对监测变量和控制变量,系统所必须维护的关系.另外还有两个关系集 *IN* 和 *OUT*,用来描述环境与系统的交互,*IN* 是从监测变量到输入数据的映射,而 *OUT* 是从输出数据到控制回来的变量的映射.

在上述思想的基础上,文献[17]建立了形式化需求模型,将软件系统定义为四元组$\langle E^m,S,S_0,T\rangle$,其中 E^m 为一组输入事件,S 为可能的系统状态集,S_0 为初始状态,T 为形为 $E^m\times S\rightarrow S$ 的部分函数,表示系统状态变迁.

在进行模型检验之前,首先需要将 SCR 描述转换为相应的工具(如 SMV)所使用的语言,SMV 计算出系统在任何状态下其变量值的合法组合(状态不变性),以及在状态变迁后变量值的合法变化(变迁不变性).可以检测出的需求不一致性包括:SCR 描述的语法和类型正确性、SCR 描述的无循环和无不确定性等,这都可以通过定义依赖于表示结构的过程来完成.

2.2.2 RSML 中的需求不一致性处理

另一类使用模型检验技术进行需求不一致性检测的工作,则以 RSML[20](requirements state machine language)作为其需求描述语言.RSML 是一种基于状态图的状态机语言,这种语言是传统状态图的一个扩展,它一方面建立了状态的层次,另一方面允许状态之间的广播式变迁.

文献[18]的工作就是使用 RSML 作为其需求描述语言的.它在进行模型检验之前同样需要将 RSML 的需求描述翻译成 SMV 的表示.使用 SMV 检测的特性包括:

(1) 绝大多数 RSML 规格说明中应该成立的特性:包括需求描述中是否出现不确定性变迁、功能定义是否一致、状态机的终止性等;

(2) 应用领域的特性:如应用领域对某些状态变量的取值范围约束或者带条件的约束是否满足,多个与现实相关的显示信息之间的对应性是否得到满足等.

2.3 基于目标的方法

与上述仅依赖需求表示形式的方法不同,KAOS 方法[21,22]的不一致需求管理策略依赖于其需求建模理念“目标”的语义.KAOS 方法认为任何软件系统都是用来实现或达到一个或一组系统目标的,其需求抽取过程表现为,首先确定一个目标,再通过目标的抽象、泛化和分解、求精,开发出一个目标的层次结构,然后以这个目标的层次结构为基础,通过一个元模型,关联上需要获取的其他概念,如任务、Agent、资源对象等,以及这些概念间的关联和约束.由于 KAOS 方法的需求建模元语(即目标)具有自身的语义,因此,对矛盾的处理更具语义特征.

2.3.1 矛盾的分类[21,22]

首先,KAOS 区分了需求不一致的 4 种程度,其中包括:

- 矛盾.在一个领域 Dom 中,一组断言 $A_1,\ldots,A_n$ 之间出现矛盾,当且仅当下列条件成立:

(1) $\{Dom,\wedge_{1\le i\le n}A_i\}$ □False(逻辑不一致)

(2) 对任意的 i:$\{Dom,\wedge_{j\ne i}A_j\}$ □False(最小性)

- 分歧.在一个领域 Dom 中,一组断言 $A_1,\ldots,A_n$ 之间出现分歧,当且仅当存在边界条件 B,使得:

(1) $\{Dom,B,\wedge_{1\le i\le n}A_i\}$ □False(逻辑不一致)

(2) 对任意的 i:$\{Dom,B,\wedge_{j\ne i}A_j\}$ □False(最小性)

(3) 存在一个场景 S 和时间点 i 使得:$(S,i)\vdash B$(可行性)

- 竞争.竞争是单个目标中分歧的一种特殊情况,由下列条件来刻画:

(1) 目标由形为$(\forall x{:}X)A[x]$的断言表示

(2) $\{Dom,B,\wedge_{i\in I}A[x_i]\}$ □False(对某个 I)

(3) $\{Dom,B,\wedge_{i\in J}A[x_i]\}$ □False(对任意 $J\subset I$)

(4) 存在一个场景 S 和时间点 i 使得:$(S,i)\vdash B$

- 障碍.障碍是在只涉及单个断言时分歧的一种边界情况:

(1) $\{Dom,B,A\}$ □False

(2) $\{Dom,B\}$ □False

(3) 存在一个场景 S 和时间点 i 使得:$(S,i)\vdash B$

不难看出,在 KAOS 方法中,目标之间可能存在的不一致性是分歧.矛盾是分歧在 B=True 时的一种特殊情况,而竞争和障碍则又是分歧的另外两种特殊情况,其中,障碍是在一般的分歧情况下取 n=1 时的情况.

2.3.2 分歧检测技术

KAOS 方法根据目标的语义定义了 4 种目标模式,其中采用了一些时态算子*:

Achieve: $P\Rightarrow\Diamond Q$; Cease: $P\Rightarrow\Diamond\neg Q$; Maintain: $P\Rightarrow Q\ \mathrm{W}\ R$; Avoid: $P\Rightarrow\neg Q\ \mathrm{W}\ R$.

KAOS 方法的目标分歧检测是找出使分歧出现的边界条件,有 3 种分歧检测技术:反向推理技术、基于分歧

*采用的时态算子包括:○(下一个状态)、●(前一个状态)、◊(将来某个时刻)、♦(过去某个时刻)、□(将来总是)、■(过去总是)、W(除非…,则将来总是)、U(直到…,则将来总是)等.

模式的技术以及基于启发式规则的技术,其中前两种为形式化技术,而第 3 种属于人工合作检测技术.

- 反向推理技术:假设系统目标和领域描述均采用形为 $A\Rightarrow C$ 的子句来表示,检测过程为

(1) 取 $B:=\neg A_i$;

(2) 令 $A\Rightarrow C$ 为所选择的规则,其中 C 与 B 中某个子公式 L 相匹配;

则有 $\mu:=mgu(L,C)^*$,$B:=B[L/A.\mu]$;

- 基于分歧模式的技术

(1) 定义目标模式,如前所示 4 种模式;

(2) 定义分歧模式,例如,Achieve-Avoid 分歧模式:由 $\{P\Rightarrow\Diamond Q,R\Rightarrow\Box\neg S,Q\Rightarrow S\}$ 得到边界条件为 $\Diamond(P\wedge R)$;

(3) 通过目标实例和分歧模式的匹配,得到分歧出现的边界条件.

- 人工合作检测技术:KAOS 方法还定义了目标的语义分类,包括 SatisfactionGoal,InformationGoal,SecurityGoal,SafetyGoal,AccuracyGoal 等,同时提供了一组启发式规则来识别分歧,包括:

(1) 若 SatisfactionGoal 目标与 SafetyGoal 目标涉及同一对象,则应考虑其间出现分歧的可能性.

(2) 若 ConfidentialityGoal 目标和 InformationGoal 目标涉及同一对象,则应考虑其间出现分歧的可能性.

(3) 若两个 OptimizeGoal 目标干预了同一个对象的属性,则应考虑其关注事件之间出现分歧的可能性.

(4) 若同一个 SatisfactionGoal 目标有关于多个 Agent 的实例,则应考虑出现竞争分歧的可能性.

2.3.3 分歧处理方法

对于检测出来的分歧,KAOS 方法提供相应的技术和启发式进行处理,主要策略包括:

(1) 避免边界条件:由于出现不一致的原因是边界条件的出现,因此最直接的策略是防止边界条件的出现,可以通过引入一个新目标 $P\Rightarrow\Box\neg B$ 即可达到此目的,其中 B 为要阻止的边界条件.

(2) 目标修补:当边界条件无法避免时则引入一个新的目标,表示如果边界条件 B 出现的话,有分歧的目标断言 A_i 在某个合理的将来就为真,即 $B\Rightarrow\Diamond_{\le d}\bigwedge_{1\le i\le n}A_i$.

(3) 矛盾预测:当发现存在一些持久条件 P,使得在某个上下文 C 中,如果 P 存留太久,则以后将会不可避免地陷入矛盾,即 $C\wedge\Box_{\le d}P\Leftrightarrow\Diamond_{\le d}\neg\bigwedge_{1\le i\le n}A_i$,则引入新目标避免预计的矛盾:$C\wedge P\Rightarrow\Diamond_{\le d}\neg P$.

(4) 目标弱化:通过弱化出现分歧的目标形式以使分歧消失,可使用下列目标弱化模式:

(i) 时态放松:如,弱化 $\Diamond_{\le d}A$ 为 $\Diamond_{\le c}A$ (其中 $c>d$)、弱化 $\Box_{\le d}A$ 为 $\Box_{\le c}A$(其中 $c<d$).

(ii) 转化 Achieve-Avoid 分歧模式 $P\Rightarrow\Diamond Q$ 或 $R\Rightarrow\Box\neg Q$ 为:(a) 弱化第 1 个断言:$P\wedge\neg R\Rightarrow\Diamond Q$;(b) 保留第 2 个断言:$R\Rightarrow\Box\neg Q$;(c) 通过增加新的目标来强化需求说明:$P\Rightarrow\Diamond(P\wedge\neg R)$.

(5) 选择其他的目标求精:若遇到上述策略无法解决的分歧,可尝试对目标的另一种求精,以期获得无分歧的子目标.

(6) 分歧消解启发式:提供一组处理分歧的启发式规则,帮助人工解决分歧.

2.4 基于图的需求描述

一些需求建模方法采用面向对象的图形形式来表达需求.针对这些方法,其需求不一致性检测的策略是,通过给定某些特定的图的性质,或者某些特定的图元的语义等,检测出一个需求描述内部、或一组需求描述间的不一致性.文献[23]针对面向对象的需求规格说明,提出用场景模型与类模型相结合,为软件系统的功能性需求建模,其中场景被定义为参与者之间(通常是系统与一组外部参与者之间)交互的有序集合,可以是具体的交互步骤的序列(实例场景),或者一组可能的交互步骤(类型场景).其需求不一致性处理的策略包括:

(1) 最小化描述之间的重叠,减少出现矛盾的可能;

(2) 系统地建立交叉引用的相关信息,检测交叉引用的不一致.所谓交叉引用是指:场景描述中创建了一个

---

*其中 $mgu(F_1,F_2)$为 F_1 和 F_2 的最一般一致化取代,$F.\mu$表示将一致性取代μ中的取代应用到 F 中产生的结果,$F[F_1/F_2]$表示用 F_2 替换公式 F 中 F_1 的所有出现.

响应,则要求类模型中存在一个类的对象有活动与此响应相对应(即实现这个响应).系统地建立交叉引用信息,即在场景描述中的每个交互上标记要涉及的类模型中的对象(或对象属性).

(3) 定义一组检测规则用特定工具进行自动检测,或由系统分析员进行系统的手工检测.

3 指称域重叠的检测

需求是现实世界一组现象的描述,是由一组符号构成的,这组符号称为需求描述的描述元素,每个描述元素都对应于现实世界中的一个或一组现象,某个特定符号的指称域则为该符号所对应的现实世界现象的集合.指称域的重叠就是一组符号所对应的现实世界现象集合间的重叠关系.文献[24]定义了两个描述元素指称域之间的 4 种关系,即无重叠(指称域不相交)、完全重叠(指称域完全相同)、包含性重叠(一个元素的指称域完全包含另外一个元素的指称域)和部分重叠(指称域相交,但既非完全重叠也非包含性重叠).只有当二元素的指称域出现后 3 种情况时,才有可能导致需求描述的不一致性.指称域重叠是出现需求不一致性的根本原因[7,24],指称域重叠的检测是需求不一致性管理中的一个主要问题.目前这方面的工作可以分为如下 4 类.

3.1 人工审查方法

由参与需求获取的系统分析员识别需求描述元素之间的指称域重叠关系,优点是所得结果的确定性,并可以采用不同的表示方法;缺点是效率低、主观性强、自动化支持程度不高以及不具备可扩展性等.

人工审查是目前应用最为广泛的方法.文献[25]提出需要识别不同需求描述之间的“强”对应性和“弱”对应性,分别相当于指称域的完全重叠关系和部分重叠关系;文献[26]提出需要说明需求描述元素之间的“相对物”关系,对应于指称域的完全重叠关系;文献[27]采用 *Z* 语言表示需求,提出需要提供所使用的变量间的“对应”关系,相当于指称域的完全重叠关系,并利用这些关系在 *Z* 框架组合时检测出需求不一致性.

文献[28]提出在建立需求描述之前,必须建立精确的“基项”集合,其中每个“基项”都要通过指派建立起与现实世界的关联,需求描述中出现的其他项则可以通过定义关联到“基项”或其他已定义的项上.这些“基项”的指派提供了需求描述与现实世界之间的桥梁,通过这个指派,需求描述中的所有项就能直接或间接地关联到现实世界的可观察现象上.

3.2 基于共享本体的方法

由系统分析员在所用到的描述元素上标记共享本体中的指定项.此标记指定了该描述元素在由共享本体给出的问题域中的解释,可以用来识别不同需求描述元素之间的关系.其优点是具有一定程度的自动判断能力,并可以采用不同的表达形式;缺点是结果的质量依赖于所有需求获取主体对共享本体的理解和认识的一致性程度,并且只能识别粗粒度的重叠关系.

文献[29]将领域模型作为共享本体,该本体给出了详尽的领域对象层次结构、对象间关系、希望达到的目标以及实现这些目标的操作子等,通过实例化本体表示的领域模型得到能为 Oz 系统[29]所处理的描述.该文假设实例化领域模型中同一个目标的描述元素是完全重叠的.

文献[30]同样采用共享领域本体作为需求描述的基础,但是与文献[29]不同的是,文献[30]中的本体是现实世界中现象的形式化,本体中元素作为软件需求描述所依据的类型体系,而软件需求描述则作为领域模型的实例.因此仅由领域本体无法完全确立指称域重叠关系,还必须同时用到下面的表示惯例方法.

3.3 基于表示惯例的方法

在需求表达完全形式化的基础上,最简单、最常用的表示惯例是名唯一假设,即同名描述元素完全相同,不同名描述元素无重叠.其优点是简单、直观,缺点是无法处理同义词和近义词问题、不能表示部分重叠关系、对模型的异构性敏感等.名唯一假设在常见的模型检验工具、定理证明技术等基于一致化算法[9,14]的方法中被普遍采用.

3.4 基于相似性度量的方法

此方法认为需求描述语言中的某些语言构造子实际上隐含了用该语言描述的现象之间的重叠关系.例如,

面向对象建模语言中的 is-a 关系,在集合包含的语义上就隐含了指称域包含性重叠或完全重叠关系.其优点是可自动识别重叠关系,并可采用不同的需求表达语言;缺点是不同模型之间需要通过元模型相关联,对模型的异构性比较敏感,并且所得结果并非总是精确的.存在一些方法实现基于相似性度量的重叠判断,文献[31]提出不同需求描述之间的结构和语义上的相似性判定方法,文献[32]中提出了构成部分领域特定本体的模型和抽象结构上的相似性判定方法.

4 分析比较和结束语

从需求抽取原则和表示形式、需求描述的粒度、描述元素指称域重叠的检测、需求不一致性的检测、需求不一致性的定位、需求不一致性的消解、实现方式等角度出发,通过深入分析可以得到现有的需求不一致性管理方法的特点:

- 基于经典逻辑的方法[9,10]:将需求描述为经典逻辑断言,描述粒度要细到能够刻画对象属性;指称域重叠的检测采用谓词和参数名的唯一原则;采用定理证明技术进行需求不一致性的检测,但是未提供需求不一致性的定位和消解手段;其实现借助于定理证明器,检测过程可以自动进行.
- 基于带标记的准经典逻辑的方法[12]:将需求描述为带标记的准经典逻辑断言,描述粒度要细到能够刻画对象属性;指称域重叠的检测采用谓词、参数和标记名的唯一原则;采用定理证明技术进行需求不一致性的检测,其中冲突源就是标号的集合,并用标记的集合来确定来源和极大一致子集;借助于定理证明器来实现,检测和定位过程可自动进行.
- 基于目标的方法[21,22]:将需求表示为系统的目标分解树,因此可以针对不同抽象程度的目标;采用目标名唯一假设进行指称域重叠的检测;用启发式规则进行分歧检测,用目标模式寻找分歧出现的边界条件,并通过分歧处理模式和启发式加以解决;但是目标检测和分析工具需要专门设计.
- 基于模型检验的方法[13−19]:将需求表示为状态变迁系统,描述粒度要细到能够刻画系统的状态变迁;采用状态名和状态取值的唯一进行指称域重叠的检测;若系统状态变迁之间存在冲突,则表明出现需求不一致,但是未提供需求不一致的定位和消解策略;其实现是,首先将系统状态模型转换为模型检验工具要求的表示形式,再借助于模型检验工具实现.
- 基于图形表示的方法[23]:需求用关联图的形式表示,可以针对不同的表示粒度;采用人工审查及表示惯例方法进行指称域重叠的检测;通过寻找图形间交叉引用的不一致性来检测和定位需求不一致性,通过人工处理来实现对需求不一致性的消解;通过定义交叉引用的规范规则,借助图形搜索算法来实现.

可以看出,所有方法都以需求获取和需求描述的原则为出发点,根据不同的需求表示形式,采用不同的处理策略.从某种程度上来说,除 KAOS 方法以外,其余方法都是在语法层次上处理了需求的不一致性.

在两种自动化方法中,基于逻辑的方法用逻辑断言表示软件需求,具有良定的判断规则,但对需求的形式化程度要求较高,同时,定理证明器的低效率也制约了需求描述的规模,一阶谓词逻辑的半可判定性也局限了这种技术的实用性.基于模型检验的方法,同样具有良定的检测过程和语义,但是要求整个系统表示成状态机的形式,需求的粒度也很细.虽然已经存在一些技术,如二分决策图 BDD[13]等,来提高模型检验的效率,但效率问题仍然是使用这种技术进行大型系统全面检测的障碍.这两种方法的另一个障碍是需求描述的难度,特别是在需求的早期阶段,很难给出所要求的精确需求描述.

基于图形的方法则是在自动化检测和易用性之间的一个折衷,此类方法用图形(如 UML 等)来描述需求,需求描述容易给出,也易于理解,可以在需求获取的不同阶段使用.但是,不一致性检测算法需要根据一致性规则来设计,这些规则是用户自定义的,有可能不正确或不全面,因而会影响检测的质量.

KAOS 方法引入目标作为需求的语义,并通过目标本身的矛盾的程度以及不同类型目标的矛盾场景,比上述方法要在更深的层次上区别并处理不一致的需求,可以说向语义级的需求不一致管理迈进了一大步.

虽然目前在需求不一致性管理方面已有很多工作,但还存在许多方面的问题需要解决.我们认为,未来的研究需要向需求的语义不一致方向发展.具体可以从以下几个方面展开:

基于环境交互的需求不一致处理[33].从语义上说,软件系统的需求就是对它将与环境发生的交互的约

束[34,35].在此意义上,需求的语义不一致性将体现为所交互环境的不一致性以及对环境作用的不一致性.

环境本体的构建.这是一个艰苦而且需要进行深入研究的工作.可以从两个方面考虑这个问题,一是从软件涉及的具体领域[36]出发,研究特定领域中软件的环境;另一方面是从软件所能解决的问题模式出发,深入研究不同问题模式中软件所处环境的特征,这对环境本体的构建有极大的推动作用.

需求不一致性度量.需求的不一致程度是控制需求进化的一个重要指标,准确的不一致性度量函数将非常关键.可以借鉴逻辑公式集合的打分函数和内聚函数[37],并结合需求的特征确立需求不一致性的度量.

矛盾的容忍、划分与封装机制.由于矛盾的普遍存在性,需求不一致性的存在是不可避免的,即存在某些需求不一致性是不可消除的,起码是在需求进化的某个阶段是不可消除的.可以从微理论[38]的思想出发,按不同的上下文或不同的场景分别建模,然后研究模型内以及模型间的一致性处理.

References:

[1] Nuseibeh B, Easterbrook S. Requirements engineering: A roadmap. In: Finkelstein A, ed. Proc. of the 22nd Int'l Conf. on Software Engineering, Future of Software Engineering Track. Limerick: IEEE Computer Press, 2000. 35−46.

[2] Balzer R. "Tolerating Inconsistency" revisited. In: Nuseibeh B, ed. Proc. of the 23rd Int'l Conf. on Software Engineering. Toronto: IEEE Computer Press, 2001. 665.

[3] Ghezzi C, Nuseibeh B. Guest editorial: Introduction to the special section. IEEE Trans. on Software Engineering, 1999,25(6): 782−783.

[4] Easterbrook S, Chechik M. Int'l workshop on living with inconsistency. In: Nuseibeh B, ed. Proc. of the 23rd Int'l Conf. on Software Engineering. Toronto: IEEE Computer Press, 2001. 749−750.

[5] Nuseibeh B, Easterbrook S, Russo A. Leveraging inconsistency in software development. IEEE Computer, 2000,33(4):24−29.

[6] Nuseibeh B. To be and not to be: On managing inconsistency in software development. In: Kramer J, Wolf A, eds. Proc. of the 8th Int'l Workshop on Software Specification and Design. Schloss Velen: IEEE Computer Press, 1996. 164−169.

[7] Spanoudakis G, Zisman A. Inconsistency management in software engineering: Survey and open research issues. In: Chang SK, ed. Handbook of Software Engineering and Knowledge Engineering. Singapore: World Scientific Publishing Co., 2001. 329−380.

[8] Nuseibeh B, Finkelstein A, Kramer J. Viewpoints: Meaningful relationships are difficult! In: Dillon L, Tichy W. eds. Proc. of the 27th Int'l Conf. on Software Engineering. Oregon: IEEE Computer Press, 2003. 676−683.

[9] Finkelstein A, Gabbay D, Hunter A, Kramer J, Nuseibeh B. Inconsistency handling in multiperspective specifications. IEEE Trans. on Software Engineering, 1994,20(8):569−578.

[10] Nuseibeh B, Kramer J, Hunter A. A framework for expressing the relationships between multiple views in requirements specification. IEEE Trans. on Software Engineering, 1994,20(10):760−773.

[11] Mully G. CORE—A method for controlled requirement specifications. In: Bauer F, Stucki L, Lehman M, eds. Proc. of the 4th Int'l Conf. on Software Engineering. Munich: IEEE Computer Press, 1979. 126−135.

[12] Hunter A, Nuseibeh B. Managing inconsistent specification: Reasoning, analysis and action. ACM Trans. on Software Engineering and Methodology, 1998,7(4):335−367.

[13] Clarke E, Grumberg O, Long D. Verification tools for finite-state concurrent systems. Lecture Notes in Computer Science 803, London: Springer-Verlag, 1994. 124−175.

[14] McMillan L. Symbolic model checking [Ph.D. Thesis]. Carnegie Mellon University, 1992.

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

[16] Atlee J, Gannon J. State-Based model checking of event-driven system requirements. IEEE Trans. on Software Engineering, 1993, 19(1):24−40.

[17] Heitmeyer C, Jeffords R, Kiskis D. Automated consistency checking requirements specifications. ACM Trans. on Software Engineering and Methodology, 1996,5(3):231−261.

[18] Chan W, Anderson R, Beame P, Burns S, Modugno F, Notkin D, Reese J. Model checking large software specifications. IEEE Trans. on Software Engineering, 1998,24(7):498−519.

[19] Heninger K. Specifying software requirements for complex systems: New techniques and their application. IEEE Trans. on Software Engineering, 1980,6(1):2−13.

[20] Leveson N, Heimdahl M, Hildreth H, Reese J. Requirements specification for process-control systems. IEEE Trans. on Software Engineering, 1994,20(9):684−707.

[21] Lamsweerde V, Darimont R, Letier E. Managing conflict in goal-driven requirements engineering. IEEE Trans. on Software Engineering, 1998,24(11):908−926.

[22] Lamsweerde V, Letier E. Handling obstacles in goal-oriented requirements engineering. IEEE Trans. on Software Engineering, 2000,26(10):978−1005.

[23] Glinz M. A lightweight approach to consistency of scenarios and class models. In: Cheng B, Weiss D, eds. Proc. of the 4th Int'l Conf. on Requirements Engineering. Schaumburg: IEEE Computer Press, 2000. 49−58.

[24] Spanoudakis G, Finkelstein A, Till D. Overlaps in requirements engineering. Automated Software Engineering, 1999,6(2):171−198.

[25] Easterbrook S. Handling conflict between domain descriptions with computer-supported negotiation. Knowledge Acquisition, 1991, 3(4):225−289.

[26] Delugach H. Analyzing multiple views of software requirements. In: Eklund P, Nagle T, Gerholz L, eds. Conceptual Structures: Current Research and Practice. New York: Ellis Horwood, 1992. 391−410.

[27] Boiten E, Derrichk J, Bowman H, Steen M. Constructive consistency checking for partial specification in Z. Science of Computer Programming, 1999,35(1):29−75.

[28] Jackson M. The meaning of requirements. Annuals of Software Engineering, 1997,3:5−21.

[29] Robinson W, Fickas S. Supporting multiple perspective requirements engineering. In: Lawrence B, ed. Proc. of the 1st Int'l Conf. on Requirements Engineering. Colorado Springs: IEEE Computer Press, 1994. 206−215.

[30] Jin Z, Lu R, Bell D. Automatically multi-paradigm requirements modeling and analyzing: An ontology-based approach. Science in China (Series F), 2003,46(4):279−297.

[31] Spanoudakis G, Finkelstein A. Reconciling requirements: A method for managing interference, inconsistency and conflict. Annuals of Software Engineering, 1997,3:433−457.

[32] Maiden N, Assenova P, Jarke M, Spanoudakis G. Computational mechanisms for distributed requirements engineering. In: Hurley WD, ed. Proc. of the 7th Int'l Conf. on Software Engineering and Knowledge Engineering. Pitsburg: IEEE Computer Press, 1995. 8−16.

[33] Zhu XF, Jin Z, Ontology-Based inconsistency management of software requirements specifications. LNCS 3381, Berlin, Heidelberg: Springer-Verlag, 2005. 340−349.

[34] Zave P, Jackson M. Four dark corners of requirements engineering. ACM Trans. on Software Engineering and Methodology, 1997,6(1):1−30.

[35] Jackson M. Problem Frame: Analyzing and Structuring Software Development Problem. Boston: Addison-Wesley Pub. Co., 2001.

[36] Lu R, Jin Z. Domain Modeling Based Software Engineering: A Formal Approach. Boston/Dordrecht/London: Kluwer Academic Publishers, 2000.

[37] Hunter A. Logical comparison of inconsistent perspectives using scoring functions. Knowledge and Information Systems Journal, 2004,6(5):528−543.

[38] Lenat D. Cyc: A large-scale investment in knowledge infrastructure. Communications of the ACM, 1995,38(11):33−38.