一种用况模型中的相似事件流检测算法

刘 辉, 麻志毅, 和云峰, 邵维忠
(北京大学信息科学技术学院, 北京 100871)

摘 要: 用况是捕获需求的主要工具之一. 随着系统复杂性的增长, 系统涉众数目急剧增加, 需求捕获变得更为困难. 解决系统复杂性问题的常用方法是采用分而治之的策略, 从不同涉众中得到系统需求的不同方面, 然后将不同方面的需求合并为完整需求模型. 随之而来的问题是不同用况之间的重叠, 就是相同或相似的事件流片断出现在多个用况中. 重复事件流片断会降低模型的可维护性、可理解性, 增加系统开发成本. 虽然人们已经意识到了这个问题, 但缺乏相应的检测算法. 本文给出基于信息检索技术的启发式检索算法以检测相似事件流片断. 实验结果表明算法是有效的.

关键词: 需求模型; UML; 用况; 场景; 事件流; 信息检索
中图分类号: TP311 文献标识码: A 文章编号: 0372-2112 (2006) 12A-2366-05

Detecting Duplicate Event Flows in Use Case Models

LIU Hui, MA Zhi-yi, HE Yun-feng, SHAO Wei-zhong
(*School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*)

Abstract: In order to deal with the complexity of large systems, the divide-and-conquer policy is adopted in requirements engineering: collect requirements from different groups of stakeholders, and then compose them together as a complete requirements specification. However, the policy brings forward the problem of overlapping use cases: similar event flows appear in more than one use case. Duplicate event flows lower the readability and maintainability of use case diagrams. The problem has been recognized, but no detecting approach is available to find out duplicate event flows. The paper proposes a heuristic algorithm which is based on information retrieval technologies. Evaluation results suggest that the algorithm is efficient and effective.

Key words: requirements specification; unified modeling language (UML); use case; scenario

1 引言

自从 Jacobson 提出用况(Use Case)之后, 用况图逐渐成为描述需求的最常用的方式之一. UML 将用况定义为参与者(Actor)与系统(System)之间的交互, 这个交互产生一个对参与者有意义的可观察的结果[4]. 这个交互使用文本形式的事件流来描述, 通常也叫做脚本. 它从用户的角度来捕获系统的功能需求, 以用户的目标(Goal)为核心描述了用户为达到目标所需要的系统能力. 它直观地解决了"系统如何满足用户需求"的问题. 同时, 因为用况描述采用格式化文本的形式描述用户与系统的交互过程(而不是系统的输入与输出之间的形式化的映射关系), 所以用况图直观易懂, 大大方便了用户与系统分析人员的交流. 格式化的文本形式也有效防止了杂乱无章的描述, 不但有利于阅读, 也有利于避免描述上的重复和遗漏. 正是因为这些优点, 用况被 UML 所采纳成为 UML 描述需求模型的主要手段. 同时, 用况驱动也是主流软件开发过程 RUP(Rational Unified Process)的三大特色之一. 国内的许多相关标准, 比如中国电子商务建模语言 cnXAML, 也都采用了用况模型. 主流的 OO 建模工具, 包括 Rose、Visio、RSA 以及国内的 JBOO[9] 都支持用况建模.

信息系统正变得越来越复杂, 涉及的人员(也称涉众——stakeholders)也越来越广. 数量众多的涉众都有可能提出他们对新系统的期望, 而这些期望正是软件需求的来源. 需求分析人员需要与这些涉众交流、记录他们的期望、整理和分析系统需求并给出正式的需求文档. 但是因为系统的复杂性以及涉众数量巨大, 一个需求分析人员往往无法应付. 为了应对系统的复杂性, 常用的一种策略是"分而治之": 将涉众和需求分析人员分组, 每组需求分析人员负责从一组涉众中提取系统的需求信息并提交一份需求文档. 最后将这些需求文档合并为一份完整的系统需求文档. 这种策略成功地解决了系统的复杂性问题, 但是也不可避免地带来了一些问题. 其中的一个问题就是在与涉众交流的阶段难以发现不同涉众小组之间的一些重叠的需求. 这些重叠的需求最后转化为用况图之后, 表现为出现在不同的用况描述中的相同或相似事件流片断. 即使

收稿日期: 2006-09-22; 修回日期: 2006-11-28
基金项目: 国家 973 重点基础研究发展规划(No. 2005CB321805); 国家自然科学基金(No. 60473064); 国家科技攻关计划(No. 2003BA904B02); 国家 863 高技术研究发展计划(No. 2005AA112030)

是同一个人提出的不同用况之间, 也会因为用况粒度的问题而存在或长或短的重复事件流片段. Michael 和 James[3] 认为不同场景之间不可避免地会出现重复片断.

重复事件流片段严重影响需求文档的质量. Xu[7] 以及 Yu[8] 等人认为, 分散在多个用况描述中的相同或相似的事件流片断会严重影响文档的质量. 其影响主要体现在: 不利于文档维护、增加需求文档的复杂性以及增加系统的实现成本. Steve Adolph 和 Paul Bramble[1] 认为, 重复编写公共事件流片断是多余的, 而且会增加模型的不准确与不一致的风险. 其次, 如果修改多处出现的事件流片断, 冗余可能导致不一致性.

针对可能出现的公共事件流片段, UML 规范提供了用况之间的包含(Include)和扩展(Extend)关系以复用公共事件流片段[4]. 其目的是要通过复用的方式减少甚至避免重复事件流的出现. 包含用况(Including Use Case)包含被包含用况(Included Use Case)的行为, 也就是事件流. 而用况之间的扩展关系则说明了在什么情况下扩展用况(Extending Use Case)的行为(事件流)如何插入到被扩展(Extended Use Case)的行为(事件流)中. 可以把出现在多个用况事件流中的事件流片断提取为一个单独的用况, 然后通过包含或者扩展关系将这个公共事件流插入到原来的事件流中. 这样就避免了多次重复描述某段完全相同或相似的事件流片断.

但是正如上文所述, 当人们采用分而治之的策略处理复杂系统的时候, 重复事件流片段是无法完全避免的. 所以对于已经出现的重复事件流片段, 也需要有方法解决. Xu[7] 以及 Yu[8] 等人提出的解决方法是用况重构. 重构是在不改变软件行为特性的前提下提高软件质量的一种有效手段. 用况重构的是通过 UML 的包含或者扩展机制, 把重复事件流片断抽取为公共用况, 从而简化了需求文档, 也方便了需求文档的维护和系统实现. 虽然用况重构提供了消除重复事件流片段的解决方案, 而且 UML 也提供了相应的机制, 但是实现用况重构需要一个前提: 找出分散在不用用况内的重复事件流片断. 随着信息系统规模的急剧增长, 软件需求文档也迅速变得庞大而复杂. 对于需求分析人员来说, 要在庞大复杂的文档中找出重复出现的相同或相似事件流片断绝非易事. 所以需求分析工具需要提供自动或半自动的检测功能, 并在此基础上辅助需求分析人员重构用况模型. 本文为解决这一问题而提出了一个自动检测相似事件流片断的启发式算法. 在 CASE 工具 JBOO[9] 中实现并验证了该算法的有效性.

第 2 节给出检测相似事件流片断的启发式算法, 第 3 节是算法评测. 第 4 节是结论和进一步的工作.

2　检测相似事件流片断的算法

本节首先给出用况的抽象模型, 然后基于这个抽象模型给出事件相似度模型. 最后利用事件相似度模型进行启发式搜索. 对检测到的相似事件流片断利用 UML 的包含和扩展关系进行模型重构.

2.1　用况抽象模型与特征追踪关系

一个用况是参与者为实现某个目的而与系统发生的一个交互序列. 这个交互通常使用格式化文本的方式描述. 这个交互序列中的每一步通常称为一个事件(Event). 每个事件包括两部分: 施动者和事件描述. 施动者是发出这个事件(或实施这个动作)的主体. 因为交互只能在参与者和系统之间进行, 所以施动者要么是参与者(Actor)要么是系统(System)[2]. 事件描述一般是一句或多句自然语言描述. 将用况形式化描述为:

$$U:=\langle e_1, e_2, \cdots, e_n\rangle \tag{1}$$

U 表示一个用况, 它由一序列的有序事件组成 $e_1, e_2, \cdots, e_n$. 事件流片断其实就是用况事件序列的一个子序列 $s=\langle e_k, e_{k+1}, \cdots, e_{k+m}\rangle$. 本文提出的抽取公共用况的算法, 其目的就是要找出各个用况$\{U_1, U_2, \cdots, U_x\}$所包含的事件流中是否有相同或相似的事件流片断 s.

特征(Feature)表示系统的能力[2], 而用况是用户所需要实现的目标以及达到目标所需要的交互(如何使用系统能力以达到目标). 所以用况和特征之间存在依赖关系. 通常使用追踪矩阵表示这种关系. 重复或相似事件流往往使用相同的功能(特征), 所以使用相同功能的用况显然更有可能出现相似事件流片断. 所以追踪矩阵也是检测公共事件流的有效信息. 用况在特征追踪上的相关度表示如下:

$$S_f(u_1, u_2)=\frac{|F_{u_1}\cap F_{u_2}|}{|F_{u_1}\cup F_{u_2}|} \tag{2}$$

S_f 为用况在特征追踪上的相关度, F_{u_i} 表示和用况 u_i 有追踪关系的特征集, $||$ 计算特征集的模.

2.2　关键词抽取与事件相似度

通过计算两个事件流片断之间的相似度可以找出相似或相同的事件流片断. 而计算两个事件流片断的相似度, 可以通过事件相似度计算得到. 一个事件就是一个文本(描述), 所以计算两个事件之间的相似度可以借鉴信息检索中用于计算两个文本相似度的技术.

算法的第一步是抽取关键词. 其目的是希望通过比较两个文本所包含的关键词来比较文本的相似性. 根据有关自动文本摘要的研究[5], 有利于文本比较的的主要是名词和动词. 借助现有的文本检索工具可以自动抽取关键词. 以关键词形式标识的事件可表示为:

$$E:=\langle X, KL\rangle \tag{3}$$

$$KL:=\{k_1, k_2, \cdots, k_n\} \tag{4}$$

E 表示一个事件, 它由两部分进行标识: 施动者(X)和关键词表(KL). 关键词是从事件描述中抽取的所有名词和动词的集合. 在传统的信息检索领域, 一般还要记录每个关键词出现在某个文本(事件)中的次数(即频率 $f_{e,k}$, 也称项频). 但是考虑到事件的描述一般也就一两句话(比起常规信息检索领域中的网页等大批量文本来说, 事件描述文本是极其短小的), 绝大部分关键词的项频都为 1, 所以本算法没有记录每个关键词的项频. 这样可以节省大量的存储空间和计算代价(后面的相似度计算会因此而得到简化).

如上文所述(第 2.1 节), 在事件描述中施动者要么是参与者要么是系统[2]. 而参与者事件是系统外部事件, 而系统动作(对参与者事件的响应)属于系统内部, 所以施动者是系统的事件与施动者是参与者的事件不应该具有可比性(或者说

相似度为零). 参与者是某类用户参与用况时所扮演的角色. 参与者之间的相似度可以通过他们共同参与的用况来反映.

$$S_a(a_1, a_2) = \frac{|U_{a_1} \cap U_{a_2}|}{|U_{a_1} \cup U_{a_2}|} \tag{5}$$

U_{a_i}表示参与者a_i所参与的所有用况的集合, $S_a(a_1, a_2)$是参与者 a_1, a_2 在"参与"关系上的相似度.

算法的第二步是为每个关键词生成事件倒排表. 对每个关键词, 生成一个列表记录所有包含该关键词的事件.

$$K \rightarrow \{e_1, e_2, \cdots, e_m\} \tag{6}$$

K 为一个关键词, $\{e_1, e_2, \cdots, e_m\}$表示包含了该关键词的事件. m 表示这个关键词一共出现在几个事件中, 在信息检索中通常称之为词频. 词频的倒数为反网页频率, 记为 idf_K.

算法的第三步: 去掉无效关键词. 根据信息检索的经验[6], 词频过低或过高的关键词对找出相似文本并没有多大帮助. Gerard Salton 和 Michael McGill[6] 发现其所抽取的关键词中有一半左右只出现一次(也就是词频为 1). 对于事件流描述也一样, 抽取出的关键词中存在大量的低频词. 这样的关键词对于抽取相似事件流意义不大. 只要阈值 η 设置得当, 去掉词频低于阈值 η 的关键词不会对抽取结果造成多大的影响, 但却可以大大减小计算量, 同时也减小存储空间需求. 去掉的低频词也必须从事件的关键词列表中移除. 对于高频词的处理与低频词类似. 如果一个关键词在几乎所有事件中都出现, 那么它对区分不同事件也没有多大意义. 可以设置最大词频阈值为 μ.

去掉无效关键词之后, 可以开始比较两个事件的相似度. 假定事件 e_1 和 e_2 为:

$$e_1 = \langle x_1, KL_1 \rangle, \ e_2 = \langle x_2, KL_2 \rangle$$

$$KL_1 = \{k_{1,1}, k_{1,2}, \cdots, k_{1,n}, k_{c,1}, k_{c,2}, \cdots, k_{c,L}\}$$

$$KL_2 = \{k_{2,1}, k_{2,2}, \cdots, k_{2,m}, k_{c,1}, k_{c,2}, \cdots, k_{c,L}\}$$

关键词集合KL_1, KL_2 包含公共元素(关键词): $\{k_{c,1}, k_{c,2}, Lk_{c,L}\}$. 如果以关键词集合 KL_1 和 KL_2 的并集 $KL_1 \cup KL_2$ 作为向量空间, 事件 e_1 和 e_2 对应的向量分别为 $\boldsymbol{w}_1$ 和 $\boldsymbol{w}_2$. 关键词 k 在事件e 的特征向量空间对应项的权值 $w_{e,k} = tf_{e,k} \times idf_k$ (项频 $tf_{e,k}$为布尔值, 关键词出现在相应事件中则为 1, 否则为 0). 事件 e_1 和 e_2 的余弦相似度为:

$$\text{Cos } Sim(e_1, e_2) = \frac{\boldsymbol{w}_1 \cdot \boldsymbol{w}_2}{|\boldsymbol{w}_1| \times |\boldsymbol{w}_2|} = \frac{\sum_{i=1}^{L} idf_{k_{c,i}}^2}{\sqrt{\sum_{i=1}^{n} idf_{k_{1,i}}^2 + \sum_{i=1}^{L} idf_{k_{c,i}}^2} \times \sqrt{\sum_{i=1}^{m} idf_{k_{2,i}}^2 + \sum_{i=1}^{L} idf_{k_{c,i}}^2}} \tag{7}$$

综合考虑事件关键词相似度与施动者相似度, 得到最终的事件相似度如下:

$$Sim(e_1, e_2) = \text{Cos } Sim(e_1, e_2) * \xi + S_a(x_1, x_2) * (1 - \xi) - \beta \tag{8}$$

ξ 是关键词相似度权重. 为了后面启发式搜索的需要, 将事件的相似度减去相似度阈值 β. 如果相似度大于零, 即为相似; 如果相似度小于零为不相似. 如果 e_1 和 e_2 中的某一个是参与者事件, 而另一个是系统响应, 那么这两个事件的相似度就应该为 $-\beta$(相似度最小值), 表示完全不同.

$$Sim(e_1, e_2) = -\beta \quad \text{if } x_1 = System \wedge x_2 \neq System \vee x_1 \neq System \wedge x_2 = System \tag{9}$$

2.3 事件流片断相似度

事件流片段由一序列的事件组成, 所以计算事件流片段相似的最直接的方法是依次匹配两个事件流片段内的事件序列并累加每对事件的相似度(具体算法参见 2.4 节). 但事件相似度考虑的主要还是文本相似度, 而没有考虑更多的用况模型信息. 如 2.1 节所述, 如果需求模型中包含特征模型而且把特征模型与用况模型相关联, 那么用况在特征追踪上的相似度也是检测相似事件流片断的有效信息, 必须综合考虑. 假定事件流片断 ef_1, ef_2 分别来自用况 u_1 和 u_2, 则事件流片断 ef_1, ef_2 相似度定义如下:

$$Sim_{ef}(ef_1, ef_2) = S_{ef}(ef_1, ef_2) * \omega + S_f(u_1, u_2) * (1 - \omega) \tag{10}$$

$S_{ef}(ef_1, ef_2)$是根据事件相似度累加得到事件流片断相似度, ω 为事件相似度权重. $S_f(u_1, u_2)$为用况 u_1 和 u_2 在特征追踪上的相关度. 本文 2.1 节的公式 2 给出了 S_f 的详细定义.

2.4 启发式搜索

将关键词按词频从低到高排序, 依次处理每个关键词. 对其事件列表中的每一对事件 e_1 和 e_2 开始启发式搜索. 假定 e_1 和 e_2 分别属于事件流 ef_1 和 ef_2

```
If Sim(e1, e2) > σ then
    往前扩展相似事件流片断
    往后扩展相似事件流片断
End If
```

如果当前事件 e_1 和 e_2 的相似度大于阈值 σ, 那么以事件 e_1 和 e_2 为起点, 顺着事件流 ef_1 和 ef_2 往前和往后扩展很可能找到两个相同或相似的事件流. 下面以前向扩展为例阐述如何扩展相似事件流片断.

```
While (未到事件流的末尾 and 事件流片断相似度 Sim_ef > Φ
       and 连续失配事件对 n 小于 λ)
  e1 = e1•NextEvent
  e2 = e2•NextEvent
  If Sim(e1, e2) > 0
    S_ef += Sim(e1, e2)
    n = 0
  Else //往前一步试图交叉匹配
      e3 = e1•NextEvent
      e4 = e2•NextEvent
If Sim(e1, e4) > 0 and Sim(e2, e3) > 0
      //交叉匹配成功
      S_ef += [Sim(e1, e4) + Sim(e2, e3)] * θ
      e1 = e1•NextEvent
      e2 = e2•NextEvent
      n = 0
    Else //交叉匹配失败
```

S_{sf} += $Sim(e_1, e_2)$

n ++

End If

End If

End While

其中 Sim_{sf} 和 S_{sf} 的关系如式(10)所示. 扩展算法并不复杂. 如果当前事件对匹配, 那就顺着事件流分别往前扩展一步; 如果当前事件对失配, 就往前扩展一步并试图交叉匹配. 交叉匹配的背景是当两个相邻事件之间不存在强制的顺序约束的时候, 不同的参与者可能会采用不同的操作顺序. 算法还以连续失配事件数作为判定是否结束扩展的一个条件. 如果连续出现 λ 对不匹配事件(采用交叉匹配也无法消除失配), 那么就意味着事件流片断的扩展已经超出了公共事件流片断的范围了. 此时需要回溯到失配前的事件对上, 并停止在该方向上的扩展. 为保证相似事件流片段的相似度, 算法还以片段的总体相似度 φ 作为判定是否继续扩展的临界值. 扩展算法的关键是几个阈值(σ、φ、λ 和 θ)的设定以及交叉匹配.

如果抽取的公共事件流过短(比如只包含一个或两个事件), 那么将它作为一个公共用况意义不大. 所以, 对抽取的相似事件流需要根据事件流长度进行一次过滤: 在结果集中删除所有长度小于 ℓ_m 的事件流片断.

2.5 用况重构

因为基于相似度的检测无法保证 100% 的查准率, 所以检测到的相似事件流片段需要人工确认它们是否可以抽取为一个公共用况. 因为被抽取的各个事件流片段未必完全一致, 所以在创立公共用况之前必须先修改相应用况的事件流, 并给出一个可以替代这些相似事件流片段的公共事件流. 最后使用包含(include)或者扩展(extend)关系将提取出来的公共事件流片段插回到原事件流的合适位置. 如果被抽取的事件流是原用况的 Alternative course(包括 Optional flow、Alternative Flow 和 Exception Flow)则使用扩展关系[2], 否则使用包含.

3 算法评测

3.1 评价方法

算法评测的主要评测指标为算法的复杂度和有效性. 算法复杂度包括时间复杂度和空间复杂度. 算法的空间需求(除了存放作为输入的用况描述之外)主要是为每个事件存放一个关键词列表以及为每个关键词存放一个事件倒排表. 用况模型(纯文本模式)的大小一般在几千字节到几兆字节之间, 而关键词列表和事件倒排表要远小于用况描述本身的大小, 所以一般的 PC 机内存就可以装载关键词列表和事件倒排表. 算法的时间复杂度则与关键词的分布相关. 因为无法给出通用的关键词分布曲线, 从而也难以在理论上给出算法的时间复杂度. 本文将通过实验运行的方式给出几个实验案例的运行时间.

算法的有效性主要包括查全率(Recall)和查准率(Precision).

查全率= [检出相似事件流的数目/用况描述中存在的相似事件流的数目] × 100%

查准率= [检出相似事件流的数目/检出的事件流的数目] × 100%

查全率和查准率的计算都需要人工的参与. 首先需要确认查出的事件流之间是否真的相同或相似. 其次, 计算查全率还需要知道整个用况描述中到底有多少相似事件流.

3.2 实验结果

相似事件流片段检测算法实现为 UML 建模工具 JBOO[9] 的一个插件, 并以此作为实验工具. 在 JBOO 中建立的用况模型可以使用这个插件检测重复或相似事件流片段, 从而为用况重构提供了契机.

实验对象是一个实际开发项目的用况需求文档. 这个项目是某知名电子商务公司开发的供公司内部使用的办公用品分发与采购管理系统, 同时负责办公室预订等内部资源管理. 实验所使用的机器是 DELL PC 机, Intel Pentium 4_2.26GHz, 内存 512MB, 硬盘 40GB, 操作系统为 Windows XP sp2.

设定参数如下: η= 2, μ= 事件数目* 10%, ξ= 70%, β= 0.5, ω= 50%, σ= 1- β, φ= 0.3, λ= 3, θ= 0.5, l_m= 4. 然后分别对两个项目的需求文档进行实验, 得到结果如表 1 所示.

表 1 实验结果

项目	需求文档(KB)	事件总数	查全率	查准率	消耗时间(s)
内部资源管理系统	21.4	225	85.8%	95%	34

这里所说的需求文档是指作为需求文档主体的用况模型. 其主体是事件流(场景)描述. 从实验结果来看, 查全率、查准率和时间复杂度都在可接受的范围内.

很显然, 查全率、查准率和时间复杂度会随着参数设置的变化而变化. 其中影响较大的是 λ 和 β. 下面的实验试图分析 λ 和 β 对查全率与查准率的影响, 并试图寻找合适的 λ、β 参数设置.

λ 表示连续失配事件对的最大值. λ 越大, 对事件失配(误差)的容忍度也就越高, 所以查全率上升而查准率下降. 图 1 给出是在实际项目中检测的 λ 对查准率与查全率的影响曲线. 可以看出, 实验结果与理论分析基本一致. 从实验结果来看, 当 λ 小于 3 的时候, 查全率急剧下降; 而当 λ 大于 4 的时候, 查准率开始显著降低. 所以 λ 比较合适的取值应该在 3 到 4 之间.

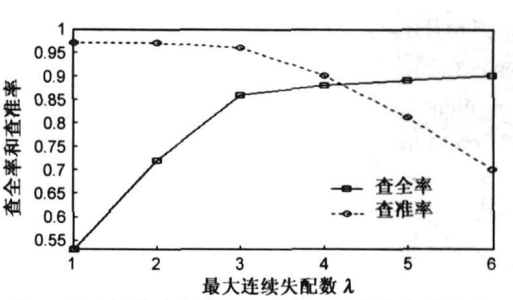

图 1 最大连续失配数 λ 对查准率与查全率的影响

图 2 是 β 对查准率与查全率的影响曲线(虚线表示查准率, 实线表示查全率). β 是通过余弦相似度区分事件相似或不相似(即失配与否)的阈值. β 越大, 事件相似度就越小, 所

以查全率越小. 而 β 越小, 不同事件之间的相似度就越大, 所以查准率开始下降. 从图 2 的曲线来看, β 比较合适的取值范围是 0.4 到 0.6 之间.

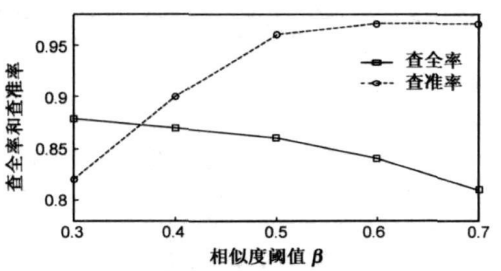

图 2 相似度阈值 β对查准率与查全率的影响

需要注意的是, 实验都是在固定其它参数的前提下得到的. 当其它参数发生改变的时候, λ和 β 的设置也会变化. 而且, 对不同的项目其理想参数配置也未必相同.

4 结论和进一步的工作

用况模型已经成为捕获用户需求的最有效手段之一, 并为主流建模语言所采纳. 用况驱动的软件开发也变得越来越流行. 但日益复杂的信息系统使得需求建模的难度越来越大. 数量庞大的涉众以及分而治之的策略决定了重复事件流片断的必然性. 这种重复事件流片断大大降低了需求模型的可读性、可维护性, 增加了系统实现的成本, 同时增加了系统不一致性的风险. 虽然已经有许多学者认识到了这个问题, UML 也提供了解决这个问题的机制, 但是解决这个问题的前提是要先找出重复事件流片断. 遗憾的是目前尚没有出现检测重复事件流片断的算法. 本文基于信息检索技术以及启发式搜索技术提出了一个检测相似事件流片断的算法, 并在 UML 建模工具 JBOO[9] 中实现了这个算法. 实验证明该算法的时间和空间复杂度都不高, 同时算法的查全率和查准率却相当不错.

目前还没有处理近义词和多义词. 在信息检索领域有许多技术可以解决这些问题(包括中文处理), 这些方法都值得借鉴. 另外, 算法还需要在不同领域的大型或者超大型复杂信息系统中进一步验证, 同时也为算法中诸多参数的调配寻找一个较为理想的设置.

参考文献:

[1] Steve Adolph, and Paul Bramble. Patterns for Effective Use Cases[M]. Boston, MA, USA: Addison Wesley Longman Publishing Co., Inc., 2002

[2] Kurt Bittner, Ian Spence. Use Case Modeling[M]. Boston, MA, USA: Addison Wesley, 2000.

[3] Michael Jesse Chonoles, James. A Schardt. UML 2 for Dummies[M]. Indianapolis, Indiana: Wiley Publishing, Inc, 2003.

[4] UML2 Superstructure[S]. OMG Document: formal/05-07-04.

[5] Gerard Salton. Syntactic approaches to automatic book indexing [A]. In Proceeding of the 26th Annual Meeting of the Association for Computational Linguistics[C]. Morristown, NJ, USA: Association for Computational Linguistics, 1988. 204- 210.

[6] Gerard Salton, Michael McGill J. Introduction to Modern Information Retrieval[M]. New York: McGraw-Hill, Inc., 1983.

[7] Xu J, Yu W, Rui K, Butler G. Use case refactoring: a tool and a case study[A]. In Proceedings of the 11th Asia-Pacific Software Engineering Conference[C]. Washington, DC, USA: IEEE Computer Society, 2004. 484- 491.

[8] Yu W, Li J, Butler G. Refactoring use case models on episodes [A]. In Proceedings of the 19th International Conference on Automated Software Engineering[C]. Washington, DC, USA: IEEE Computer Society, 2004. 328- 331.

[9] 麻志毅, 赵俊峰, 孟祥文, 张文娟. 青鸟面向对象软件建模工具的研究与实现[J]. 软件学报, 2003, 14(1): 97-102.
Ma Zhi-Yi, Zhao Jun-Feng, Meng Xiang-Wen, Zhang Wen-Juan. Research and implementation of Jade Bird object-oriented software modeling[J]. Journal of Software, 2003, 14(1): 97- 102. (in Chinese)

作者简介:

刘 辉 男, 1978 年 11 月出生于福建省长汀县, 现为北京大学计算机软件与理论专业博士研究生. 研究领域为面相对象技术、模型驱动的体系结构、软件重构以及形式化软件工程方法. E-mail: liuhui04@sei.pku.edu.cn

麻志毅 男, 1963 年 7 月生于内蒙古赤峰市, 博士, 副教授, 主要研究方向为软件工程与软件工程环境、面向对象技术和构件技术等. E-mail: mzy@sei.pku.edu.cn