

## 软件产品族构件演化及其复杂度评价

张元鸣,肖刚,徐恭旭,陆佳炜

(浙江工业大学 计算机科学与技术学院,杭州 310023)

(zym@zjut.edu.cn)

**摘要:**基于现有构件以演化方式产生出新的构件是提高软件复用水平和满足用户不断变化需求的关键技术。首先,给出了一种基于多个代理的构件演化交互模型,该模型能够以自治方式实现演化一致性数据处理;其次,利用方面织入机制将新的功能代码准确织入现有构件内部,降低了构件不同功能代码的耦合度;然后,对构件演化的复杂度进行了讨论,给出了四项演化复杂度评价指标和一个复杂度计算模型,以对构件演化成本进行量化估算;最后,以数字化校园中各应用系统间数据交换构件演化为例,证明了方法的可行性和有效性。

**关键词:**软件产品族构件;构件演化;代码织入;演化复杂度

**中图分类号:** TP311.52 **文献标志码:** A

## Evolution of software product family component and its complexity evaluation

ZHANG Yuan-ming, XIAO Gang, XU Gong-xu, LU Jia-wei

(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou Zhejiang 310023, China)

**Abstract:** Evolving new software component based on previous software components is a key technique to improve software reusability and satisfy users' various demands. In this paper, an interactive evolution model was proposed based on multiple Agents, which could autonomously process consistent data. Then, the aspect weaving mechanism, which can effectively reduce the coupling degree of different function areas, was introduced in evolution to insert new codes into the exact places of target component. Furthermore, the evolution complexity was also discussed and several indicators and a model were given to calculate evolution cost. Finally, a data exchange component used in digital campus system was given to illustrate the effectiveness of above evolution methods.

**Key words:** software product family component; component evolution; code weaving; evolution complexity

### 0 引言

基于构件的软件开发方法(Component-Based Software Development, CBSD)通过软构件的组装开发软件系统<sup>[1-3]</sup>,它能够实现构件的即插即用,是提高软件复用水平的有效开发方法。然而随着用户功能需求的变化,构件功能也必然发生变化,如何基于现有构件以较低的成本和较短的周期产生出能够满足用户新功能需求的构件是一个重要课题。

软件产品族构件是指解决同一问题空间的一族功能、代码和接口相似构件的集合<sup>[4]</sup>,可为基于构件的开发提供不同功能版本的构件。同时,软件产品族构件也为开发具有新功能的构件提供了可复用资源。构件演化<sup>[5]</sup>是指基于现有软件产品族构件以演化方式产生能够满足用户新功能需求的软构件开发方法,其可分为两个方面:一是由于系统体系结构发生变化,原有的构件被新的构件所替换,从而导致构件之间连接的变化,被称为构件的外部演化;二是构件有了新的升级版本,从而导致构件实现的变化,被称为构件的内部变化。由于构件一般由接口、功能和信息三部分组成,因此构件的演化类型也分为接口演化、功能演化和信息演化三种类型。

构件演化主要有三种方式:继承方式、包装器方式和代理方式。前两种演化方式的实现在一定程度上是可行的,但存

在着问题:继承演化方式需要分析构件的源代码和详细了解构件的接口等信息,其应用比较难以掌握;包装器方式不能复用接口代码,而且被包装的构件是紧密耦合的。

本文对软件产品族构件演化交互方式、演化方法及其演化复杂度进行了较为系统的研究。首先对构件的演化交互过程进行了研究,给出了一种多代理(Agent)的演化交互模型,该模型能够以自治方式实现演化一致性数据处理,协同完成构件演化过程;然后对构件的演化机制进行了研究,并利用方面织入机制将新的功能准确织入现有构件内部,降低了构件不同功能代码的耦合度;此外,为量化估算构件演化的成本,还分析和讨论了构件演化复杂度评价指标体系并给出了一个复杂度计算模型。

### 1 软件产品族构件演化交互模型

一般地,软件产品族构件演化是基于现有构件的演化,即采用拷贝副本的方式,然后将演化后的构件作为一个新的产品族构件。

构件在演化过程中需要处理大量数据,主要包括演化前后的一致性数据以及演化过程中的备份数据等<sup>[6]</sup>。本文所提出的演化模型是一个基于代理(Agent)的构件演化方法,演化过程数据交互由相应的Agent以自治方式进行处理<sup>[7]</sup>,

收稿日期:2010-09-06;修回日期:2010-11-03。 基金项目:浙江省自然科学基金资助项目(Y106603)。

作者简介:张元鸣(1977-),男,河南濮阳人,讲师,博士,主要研究方向:软件体系结构、软构件;肖刚(1965-),男,浙江上虞人,教授,主要研究方向:软构件、软件产品族方法、智能信息系统;徐恭旭(1984-),男,浙江乐清人,助教,硕士,主要研究方向:软构件;陆佳炜(1981-),男,浙江湖州人,助理研究员,硕士,主要研究方向:Web服务、算法分析。

Agent 作为演化的处理中心,通过接收命令参数,完成各自规定的功能。

基于多代理的软件产品族构件演化交互模型如图 1 所示,它由表示层、演化层和数据层构成。表示层负责与用户进行信息交互,收集数据,是一个可视化界面。

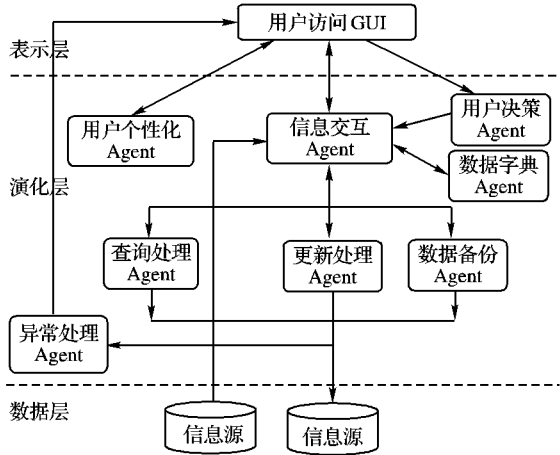


图1 基于多代理的构件演化交互模

演化层负责交互过程数据的处理,由多个 Agent 构成,包括用户个性化 Agent、信息交互 Agent、查询处理 Agent、更新处理 Agent、信息备份 Agent、动态数据字典 Agent 以及异常处理 Agent。这些 Agent 的功能相互独立,其自身行为和不受其他 Agent 的限制,它们相互服务、协同完成演化过程数据处理,各 Agent 的功能如下。

1) 用户个性化 Agent。记录与用户相关的数据,包括操作日志记录和用户喜好的操作行为。操作日志用做数据不一致时数据恢复的依据,操作行为为用户再次登录提供建议策略,提供智能化的人机交互界面。

2) 信息交互 Agent。负责转发命令,是信息处理过程的中转站。它是用户 GUI 界面可以发送命令参数的接口,将接收的命令发送到查询处理 Agent、更新处理 Agent 和数据备份 Agent。

3) 用户决策 Agent。参与用户决策,用户可以通过图形用户界面(Graphical User Interface, GUI),提供灵活全局配置文档参与决策,例如异构数据源之间数据的映射匹配规则,相关参数的输入等。

4) 动态数据字典 Agent。负责维护数据字典,包括数据表名及属性。当数据表结构类型发生变化时,该 Agent 动态捕获变化信息,及时更新数据字典中相应信息。

5) 查询处理 Agent。负责从信息源中提取数据,并处理数据的不一致性。

6) 更新处理 Agent。负责维护本地数据源或异构数据源的更新操作以及维护数据一致性与完整性。

7) 数据备份 Agent。负责备份本地数据源或异构数据源的数据交换,提供数据安全保证。

8) 异常 Agent。负责数据交互过程异常处理,保持数据交互的正确性。异常 Agent 采用数据的完整性约束,数据交互出现异常时,以事务方式进行处理,对交互的数据进行回滚,恢复到数据交互前的状态,并将异常情况反馈给用户,用户通过决策 Agent 提供解决的方案来解决异常,完成数据的

交互。

数据层包括两个数据源:信息源 1 是接收数据的接口,用来接收数据信息并进行保存;信息源 2 是输出数据的接口,用来存储处理结果。

在事物演化过程中引入代理的思想已经在其他领域出现。李强等人<sup>[14]</sup>在语言的演化过程中引入多 Agent 的方法,以语言的产生及其演化问题为研究对象,进行了基于多 Agent 的建模仿真实验。杨军等人<sup>[15]</sup>将 Agent 引入组织机构的演化,提出了一种基于面向结构的非自利型 Agent 组织形成和演化机制。詹剑峰等人<sup>[16]</sup>以软件体系结构的类型化理论为基础,探讨了基于子类型关系的 Agent 演化,提出了基于子类型关系的演化、扩充、迁移等方法,可以满足多 Agent 系统构造过程中 Agent 类型的多样性。而在软件产品族构件演化中如何利用 Agent 具体实现构件演化并未在相关文献中出现。本文将利用自治的 Agent 处理演化交互数据,能够根据需求的变化采取相应的智能策略,具有一定的灵活性和自适应性<sup>[8]</sup>。

## 2 基于方面织入的构件演化机制

构件演化交互模型给出了构件演化过程中数据处理的方式,使得能够以代理的方式完成交互数据处理,而如何将新的功能代码准确地加入到现有构件是构件演化必需解决的另一个关键技术。其基本原则是功能代码的添加应尽量与原有功能代码进行分离,必需的数据交互也要通过各 Agent 代理完成,这是以遵循功能的模块化和信息隐藏两大原则为目标,避免构件中不同功能成分之间不必要的耦合关系,使得构件中的代码可以相对独立地发生变化。

方面织入是面向方面编程(Asspect Oriented Programming, AOP)的方法,它采用横切分离关注点技术,将分离出的关注点(程序的各个功能)独立实现<sup>[9-11]</sup>,通过相应的机制将分离的关注点代码插入到原来的对象,可以大大降低各功能间的耦合度。

为准确地将新的功能代码加入原有构件,降低构件不同功能代码的耦合度,本文在构件功能演化机制中引入方面织入的思想,实现功能演化中代码的准确定位。其演化实现过程分为如下两步。

第一步 获取构件功能演化点,即在什么点上进行功能演化,这个点可以是函数层次或类层次,由查询处理 Agent 基于反射机制<sup>[12]</sup>获取原有构件功能列表及相关剖面信息,根据演化的功能需求确定功能的演化点。

第二步 确定新功能在功能演化点的织入时机,可以按以下三种规则织入。

1) 前通知(before)。新增功能在到达一个功能演化点运行之前织入。

2) 后通知(after)。新增功能在功能演化点之后织入。

3) 环绕通知(around)。新增功能包围功能演化点,如方法的调用,环绕通知在功能演化点前后可以完成自定义的行为,主要负责选择继续执行的连接点或此时返回其值。

上面三个织入规则中“前通知”和“后通知”在构件功能添加时使用,而“环绕通知”用于对构件原来某个功能的修改,加载新功能而短路原来的功能,这三个织入规则在产品族

构件演化过程中,将由用户决策 Agent 发送指令,确定织入的规则,同时根据查询 Agent 获取到的构件功能信息发送给用户决策 Agent,这样用户决策 Agent 可以明确织入的目标与时机,然后更新处理 Agent 生成织入规则文件,规则文件采用如下 XML( eXtensible Markup Language) 语言描述:

```
< DynamicWeaving >
  < TargetComponent >
    < ComponentId >
      ComponentId
    </ComponentId >
    < ComponentName >
      Target component name
    </ComponentName >
    < TargetMethod >
      Target method name
    </TargetMethod >
  </TargetComponent >
  < AspectComponent >
    < AspectComponentName >
      Aspect component name
    </AspectComponentName >
    < MethodName >
      Aspect method name
    </MethodName >
    < Advice >
      Advice type
    </Advice >
  </AspectComponent >
</DynamicWeaving >
```

织入规则配置信息的描述包括目标构件,目标功能演化点 <TargetMethod>。对于要演化的功能以方面构件描述 <AspectComponent>,也就是要添加或修改的功能构件,其子节点 <MethodName> 定义运行的起始点入口,<Advice> 节点定义方面构件织入到目标产品族构件的织入规则,可以是 before、after 和 around 类型。这里的织入规则主要分成了两部分,<TargetComponent> 节点是演化前产品族构件部分抽象描述,将其功能演化点映射到具体的类或函数上,<AspectComponent> 节点是要添加或修改功能的抽象描述,其映射到要添加的代码实体上,将两部分整合起来就是整个织入规则描述。软件产品族构件演化机制如图 2 所示。

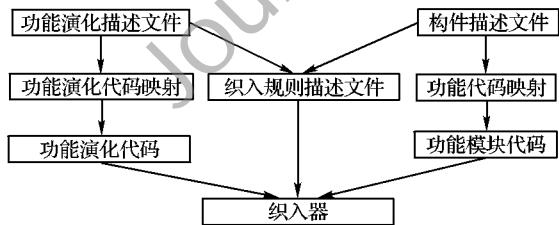


图 2 软件产品族构件演化机制

织入规则配置信息由更新处理 Agent 根据 Agent 间的传递信息自动生成,为产品族构件运行时功能的演化提供依据,根据实际应用需求的改变,只需要通过修改织入配置信息中相关的信息,织入到目标构件中执行。

3 构件演化复杂度评价

在演化过程中,某个构件的演化可能会导致其关联构件的同步演化,这些关联构件演化的总成本有可能会高于开发

新构件的成本。这样构件演化就失去了实际意义,因此有必要在构件演化前对其演化的复杂程度进行估算,为演化的必要性及成本给出参考。本章将对评价构件演化复杂度评价指标及计算方法做进一步探讨。

传统的软件开发成本估算通常是从投入软件开发的人数和开发的时间等方面进行考虑。由于构件演化是基于复用思想在现有构件基础上通过演化生成新的构件,故构件演化复杂度的计算方法应不同于传统软件开发成本的计算方法。本文设置了以下 4 个演化复杂度评价指标。

1) 演化关联度 (Evolution Related Components, ERC)。构件之间通过接口进行交互,在演化过程中,对某个构件的演化可能会涉及到其他构件。如果本构件的演化引起关联构件的演化,必将增加演化的成本,关联度是衡量其他关联构件一起进行演化复杂程度的一个评价指标,关联度的计算方式如下:

$$ERC = \sum_{i=1}^n f(C_i) \tag{1}$$

其中:  $n$  是与构件  $C$  关联的构件个数;  $f(C_i)$  是构件  $C_i$  与构件  $C$  的关联度计算因子,用于评价该关联构件演化复杂度。

2) 演化功能相关度 (Evolution Functional Mutuality, EFM)。功能相关度是指现有构件演化成目标构件的相关程度。若目标需求的产品族构件与提取的产品族构件完全符合,则此时的相关度为最小。该参数是为了评估演化前后的差异,若差异过大,必将对原有构件进行大量的修改,导致更大的演化成本。本文设计的产品族构件功能相关度估算方法如表 1 所示。

表 1 EFM 模糊量化值

描述	量化值
功能演化不用任何修改,完全符合需求构件	1
功能演化需少量修改,演化程度较小	3
功能演化有一定的工作量	5
功能演化需大量修改,演化程度很大	7

3) 演化构件粒度 (Evolution Component Granularity, ECG)。粒度是指产品族构件本身的复杂程度。大粒度的产品族构件,由多个原子构件或复合构件组成,功能较为复杂;小粒度的产品族构件由少数原子构件或复合构件组成,功能较为简单。粒度的大小反映了构件自身的复杂程度,而这直接影响了该构件的演化成本,因此设置该参数是为了考虑构件自身的复杂程度对演化成本的影响。本文设计的粒度模糊量化值如表 2 所示。

表 2 ECG 模糊量化值

描述	量化值
构件功能本身粒度很小,功能较单一	1
构件功能本身粒度一般,由少许原子构件组成,功能不复杂	3
构件功能本身粒度较大,由一些原子构件与复合构件组成,功能较复杂	5
构件功能本身粒度很大,由一些复合构件组成,功能很复杂	7

4) 演化代码量 (Evolution Code Lines, ECL)。演化代码量是指从提取的产品族构件到目标需求构件需要修改的代码数量,以行为单位进行量化。该指标主要用于评价将要实现的功能的复杂程度,作为构件演化复杂的一个重要指标。本

文将原产品族构件作为估量的参考对象,设置了演化代码的模糊量化值如表3所示,以提高估量的准确性。

表3 ECL 模糊量化值

描述	量化值
功能修改代码较少(小于现有构件的10%)	1
功能修改代码一般(现有构件的10%~20%)	3
功能修改代码较多(现有构件的20%~30%)	5
功能修改代码很多(大于现有构件的30%)	7

本文设置了以上四个评价指标,分别从构件内部(ECG)、与外部的联系(ERC)以及演化前后(EFM、ECL)三个方面对演化复杂度进行评价。依据这四个估算指标,按照层次分析法(Analytic Hierarchy Process, AHP)<sup>[13]</sup>对产品族构件演化复杂度进行实际估算,其计算方法是首先对这四个指标按其重要性优先顺序进行排队;然后确定各指标的量化值(权重),对于无法实际量化的指标,给出其模糊值;最后根据以下公式计算出演化复杂度(Evolution Complexity, EC):

$$EC = W^T \times (ERC, EFM, ECG, ECL) \quad (2)$$

其中  $W$  是各元素指标的权重值向量。对于计算出的  $EC$ , 若其值过大则说明对于提取出的目标构件需较大演化成本, 否则说明演化成本较低。该演化复杂度模型综合考虑了影响构件演化复杂程度的几个重要指标, 可以对演化复杂程度做出较为准确的评估, 对在构件演化前对其演化的必要性给出成本参考。

#### 4 应用实例

在数字化校园中,各应用系统往往需要频繁地进行数据交换,以保持各系统间的数据一致性。然而实际应用中各应用系统的数据交换对象,如数据表、字段和类型,是不同的。导致数据交换难以由现有构件完成,需要不断重新开发不同版本的数据交换构件。本章以该数据交换构件为例来说明以上所述软件产品族构件演化方法。

基于构件演化技术在现有数据交换构件的基础上演化出新的构件不需要重新编写程序代码,只需通过配置相关信息,即可完成数据交换任务,解决数据源变化引起的数据导入问题。

在演化过程中首先获取现有数据交换构件的功能演化点信息,包括其内部声明的方法和属性信息,这些信息由信息查询处理 Agent 来得到,其部分源代码如下:

```
Static void SearchAgent( String component) {
    Class cls = Class.forName( component);
    //获取构件内部声明方法
    Method methodlist = cls.getDeclaredMethods();
    for ( int i = 0; i < methodlist.length; i++) {
        Method m = methodlist[i];
        //逐个获取方法名称,添加到全局变量中
        NameList.add( m.getName());
        //获取对应方法,所有参数信息
        for( intj = 0; j < m.getParameterTypes().length; j++)
            { ...}
    }
}
```

在数据交换实现中,要求构件能够适应数据源的变化。

针对这一要求,数据交换构件采用动态配置导入的方法,通过解析配置文件来生成对应的数据库操作语句,这样若表结构发生变化,不用修改构件代码而仅需修改配置文件即可。数据交换的部分 XML 配置文件如下:

```
<?xml version = "1.0" encoding = "utf-8"? >
<list >
    <field id = "0" >
        //数据源 1, 数据来源地
        <datasource_from > FromDataBase
        </datasource_from >
        //数据源 2, 数据导入地
        <datasource_to > ToDataBase
        </datasource_to >
        //数据来源地对应表
        <table_from > FromTable </table_from >
        //数据导入地对应表
        <table_to > ToTable </table_to >
    </field >
    <field id = "1" >
        //数据来源地表字段 1
        <from_field > student_id </from_field >
        //数据导入地表字段 1
        <to_field > student_id </to_field >
        //数据类型匹配
        <type > string </type >
        //数据长度
        <length > 10 </length >
    </field >
    <field id = "2" >
        <from_field > student_name </from_field >
        <to_field > student_name </to_field >
        <type > string </type >
        <length > 15 </length >
    </field >
    ...
</list >
```

通过 XML 配置文件能够灵活地完成数据交换功能,主要表现为数据来源地、对应的表及其字段可以灵活地映射到目的数据库中,在数据导入时只需解析该配置文件,然后将对应关系传递到相应的数据库操作语句就可以,完成多个不同数据源的交互。这里  $\langle type \rangle$  节点是类型匹配信息,当数据导入过程中由于数据类型或数据长度的不匹配,其验证工作将交给异常处理 Agent 来处理,若发生异常时由异常处理 Agent 回滚事务保持数据的一致性。

新增功能由相应的更新处理 Agent 来生成相应的织入规则文件,将原有构件功能与新增构件联系起来,置入构件运行容器中,完成功能的演化。

设现有数据交换构件为 C, 则演化成目标数据交换构件的演化复杂度计算过程如下。

1) ERC。由于构件 C 调用了数据库连接构件和 SQL 语句生成构件, 所以  $ERC = 2$ 。

2) EFM。构件 C 功能的演化主要是添加数据灵活匹配的功能, 对原构件功能的修改较少, 属于添加部分功能, 所以  $EFM = 3$ 。

3) ECG。构件 C 本身完成的功能较单一, 先调用数据连接, 然后调用数据库操作语句生成构件, 所以  $ECG = 3$ 。

4) ECL。由于其需要添加部分功能,根据原构件的代码量,经估算,  $ECL = 5$ 。

确定以上各指标后,再用 AHP 方法确定各个指标的权重值  $W = (0.45, 0.18, 0.25, 0.12)$ , 得出演化复杂度  $EC = 2.25$ , 由于此值相对而言较小,故通过演化方法开发新的构件是经济可行的。

## 5 结语

软件产品族构件演化基于现有构件通过演化方式产生出满足新功能需求的构件,是提高软件复用水平和满足用户不断变化的功能需求的关键技术。本文对构件演化交互模型、演化机制及其演化复杂度评价进行了研究,设计了一个基于多代理的构件演化交互模型,该模型能够以自治方式协同处理演化一致性数据;然后通过利用方面织入机制将新的功能代码准确织入到现有构件内部,降低了构件不同功能代码的耦合度;此外,本文还设计了四个构件演化复杂度评价指标和一个评价模型,可以在构件演化前对其演化成本进行量化估算。

### 参考文献:

- [1] 杨美清. 软件工程技术发展思索[J]. 软件学报, 2005, 16(1): 1-7.
- [2] 梅宏, 陈锋, 冯耀东, 等. ABC: 基于体系结构、面向构件的软件开发方法[J]. 软件学报, 2003, 14(4): 721-732.
- [3] LAU K-K. Software component models [C]// Proceedings of the 28th International Conference on Software Engineering. New York, USA: [s. n.], 2006: 1081-1082.
- [4] 邹盛亨, 张伟, 赵海燕, 等. 面向软件产品家族的变化性建模方法[J]. 软件学报, 2005, 16(1): 37-49.
- [5] 符进强, 汪洋, 钱乐秋. 基于动态构件框架的构件演化[J]. 计算

机科学, 2001, 28(1): 21-24.

- [6] 罗毅, 李兴宇, 关连伟, 等. 构件演化中的系统行为一致性的研究[J]. 计算机科学, 2008, 35(1): 266-271.
- [7] 聂亚杰, 刘大昕. 面向 Agent 的软件工程[J]. 小型微型计算机系统, 2002, 23(4): 417-420.
- [8] 张琼妮, 肖刚, 张元鸣. 基于 XML 的自适应软构件模型研究[J]. 计算机工程, 2006, 32(17): 141-143.
- [9] SUVEE D, VANDERPERREN W, JONCKERS V. JAsCo: An aspect-oriented approach tailored for component based software development [C]// Proceedings of the 2nd International Conference on Aspect-oriented Software Development. Boston, USA: [s. n.], 2003: 21-29.
- [10] XIAO GANG, RUAN RONG-CHUN, LU JIA-WEI, et al. Research of reflective component model based on the separation of multidimensional concerns [C]// Proceedings of 2008 IEEE International Conference on E-Business Engineering. Washington, DC: IEEE Computer Society, 2008: 759-764.
- [11] 周颖颖. 基于 AOP 的横切关注点实现架构应用研究[D]. 大连: 大连海事大学, 2006.
- [12] 徐基旭, 肖刚, 陆佳伟, 等. 基于反射机制的多 Agent 数据交互模型研究[J]. 计算机应用, 2009, 29(10): 2830-2833.
- [13] 王莲芬, 许树柏. 层次分析法引论[M]. 北京: 中国人民大学出版社, 1999.
- [14] 李强, 毕贵红, 王华. 基于 Agent 的语言演化仿真研究[J]. 计算机仿真, 2009, 26(1): 287-291.
- [15] 杨军, 李中学. 基于组织效用的 Agent 组织形成和演化机制[J]. 计算机工程与设计, 2009, 30(5): 1285-1288.
- [16] 詹剑锋, 程虎. 基于软件体系结构的 Agent 规约和演化[J]. 计算机研究与发展, 2002, 39(12): 1543-1549.

(上接第 817 页)

## 4 结语

本文通过假定顾客到达服从泊松分布,且到达率随时间变化,还假定顾客对待购物品进行报价,它是私有信息,别人不知道其具体值,只知道它的分布函数,研究了供应方唯一,网站销售库存有限、商品具有时效性时如何制定最优的限制价策略问题,建立了线销售商的随机收益模型,用 Mathematic 5.0 对模型进行数值分析,得到如下结论:销售商对于不同初始库存、不同的销售时间以及不同的顾客到达率其最优限制价格策略是不同的,具体是:

1) 在其他条件不变的情况下,销售商初始库存愈高,销售商的最优限制价格应愈低,反而愈高;

2) 在其他条件不变的情况下,商品的销售时间愈长,销售商的最优限制价格应愈高,反而愈低;

3) 在其他条件不变的情况下,顾客到达率愈高,销售商的最优限制价格愈高,反而愈低。

### 参考文献:

- [1] 徐雅卿, 魏轶华, 胡奇英. 基于 Priceline 的买方/卖方定价收益管理问题[J]. 管理科学学报, 2008, 11(3): 63-69.
- [2] HANN I H, TERWIESCH C. Measuring frictional costs of online transactions: The case of a name-your-own-price-channel [J]. Man-

agement Science, 2003, 49(11): 1563-1579.

- [3] TERWIESCH C, HANN I H, SAVIN S. Online haggling at a name-your-own-price retailer: Theory and application [J]. Management Science, 2005, 51(3): 339-351.
- [4] FAY S. Partial-repeat-bidding in the name-your-own-price channel [J]. Marketing Science, 2004, 23(3): 407-418.
- [5] SPANN M, SKIERA B, SCHÄFFERS B. Measuring individual frictional costs and willingness-to-pay via name-your-own-price mechanisms [J]. Journal of Interactive Marketing, 2004, 18(4): 22-36.
- [6] DING M, ELIASHBERG J, HUBER J, et al. Emotional bidders - An analytical and experimental examination of consumers' behavior in a priceline-like reverse auction [J]. Management Science, 2005, 51(3): 352-364.
- [7] SPANN M, TELLIS G J. Does the Internet promote better consumer decisions? The case of name-your-own-price auctions [J]. Journal of Marketing, 2006, 70(1): 65-78.
- [8] 周振红, 黄深泽. 顾客网上讨价还价最优出价问题[J]. 系统管理学报, 2008, 17(6): 660-664.
- [9] 周振红, 黄深泽. 在线报价系统中销售商最优限制价的制定[J]. 统计与决策, 2008, 274(22): 59-61.
- [10] WILSON J G, ZHANG GUOREN. Optimal design of a name-your-own-price channel [J]. Journal of Revenue and Pricing Management, 2008, 7(3): 281-290.