软件 Agent 的继承性研究*

樊晓聪 徐殿祥 侯建民 郑国梁

(南京大学计算机软件新技术国家重点实验室 南京 210093)
(南京大学计算机科学与技术系 南京 210093)

摘要 Agent 作为一种受限的智能对象,对 Agent 的继承特性进行深入研究并将继承机制嵌入到 AOP(agent-oriented programming)中则具有重要意义.文章基于 BDI Agent 模型,对软件 Agent 的继承性和复制行为进行了研究.从单继承和多继承两个方面给出了 Agent 继承的语义,将 Agent 实例的动态复制机制形式地划分为功能分割、逻辑分割、择优分割和返祖分割这 4 类,分析了每类分割方式的作用,并基于电子市场系统应用背景给出了相应的实例.

关键词 软件 Agent,继承,复制行为,BDI 模型,规划.

中图法分类号 TP311

AOP(agent-oriented programming)是一种特殊的 OOP(object-oriented programming)[1~3].Agent 的心智状态、通信机制和能力分别对应于对象的状态、消息收发和操作.Wooldridge[4]认为,利用言语行为协议(speech act theory)进行协作的 Agent 既是推理系统,又是认知实体,它为 AOP 提供了自治和认知两层抽象.自治抽象揭示了 Agent 与主动对象之间的关系.在自治性、并发性和应激性上,两者具有近似对应关系;认知抽象与人们通常利用大众心理学来解释和预测复杂系统行为的方法相耦合.

但是,Agent 在自治程度上优于主动对象.虽然主动对象可以根据它所处的环境来决定自身行为,从而实现控制的局部化,但其行为集是预先确定的,并且缺乏协商和学习能力;Agent 的预动性(proactive)赋予其独有的规划特征,使其可以通过对底层原动作的动态规划控制自身行为,能对不可预知的事件作出合理反应.因而,Agent 能更自然、更真实地模拟和解决客观世界中的问题.

另一方面,OOP 的日益成熟也为 AOP 的发展提供了契机.AOP 能否普及,并像 OOP 那样焕发强大的生命力,取决于其能否提供 OOP 目前所具有的特性,并在某些方面(诸如易理解性、易维护性等)能否超越 OOP 而提供独具的优点.由于继承性是 OOP 的基石,AOP 能否具有继承性很自然地便成为首先要解决的问题.Crnogorac[5]意识到这一点,并初步探讨了 AOP 中的继承问题,使 Agent 的对象属性研究作为 Agent 方法学的一个重要方面重新得到重视,从而在一定程度上缓解了 Agent 研究领域两极分化的现象[6].但 Crnogorac 仅局限于其中的行为保持和继承异常问题,对 Agent 继承中的许多细节没有充分考虑.

本文试图从单继承和多继承两方面对 Agent 的静态继承进行研究,并深入探讨 Agent 的动态继承-动态复制行为(cloning).第 1 节介绍改进的 BDI 模型.第 2 和第 3 节分别讨论 Agent 的静态继承和动态复制行为.第 4 节对与 Agent 继承有关的问题进行分析.最后概括本文的主要工作.

1 BDI 模型与规划表示

在 Agent 理论研究领域,BDI 模型[7]是最富有成果的 Agent 模型.处于动态环境中的 BDI Agent 通过对外

---

* 本文研究得到国家自然科学基金和国家 863 高科技项目基金资助.作者**樊晓聪**,1971 年生,博士,主要研究领域为软件工程,Agent 方法学,多 Agent 系统.**徐殿祥**,1967 年生,博士,副教授,主要研究领域为软件工程,软件 Agent.**侯建民**,1967 年生,博士,主要研究领域为实时系统,模型验证.**郑国梁**,1937 年生,教授,博士生导师,主要研究领域为软件工程,知识工程.

本文通讯联系人:樊晓聪,南京 210093,南京大学计算机科学与技术系

本文 1998-06-12 收到原稿,1998-12-15 收到修改稿

界变化的不断感知，基于自身的信念产生目标，对某些目标作出承诺从而产生意念．通常将一个 BDI Agent 表示为$(\mathscr{K},\mathscr{B}^I,\mathscr{G}^I,\mathscr{P}^S)$，其中 $\mathscr{K}$，$\mathscr{B}^I$，$\mathscr{G}^I$ 和 $\mathscr{P}^S$ 分别表示知识集(为真的信念)、初始信念集、初始目标集和静态规划集，它们由特定的 Agent 语言描述，用来规范 Agent 的静态心智特征．

在语法上，规划由激发事件集、语境条件集和规划体组成．可将规划 p 表示为元组(p_E,p_C,p_G)，p_E 和 p_C 分别对应于 p 的激发事件集和语境条件集，p_G 为规划图．规划图的节点表示状态，弧表示状态转换，弧上的标记为一个三元组(E,C,A)，分别表示激发转换的事件、转换发生的条件和转换伴随的动作．规划图中有 3 类节点：1 个初始节点，3 种叶节点("成功"、"失败"和"放弃")和内节点．内节点可以是具体节点(原动作、通信语句、不确定语句等)，也可以是代表子规划体的抽象节点．抽象节点有一指针指向其扩展子图．

定义 1.1(服务簇)．Agent 的一个规划图与其中所有抽象节点的扩展子图共同构成该 Agent 的一个服务簇．服务簇实例化后得到具体的服务．

定义 1.2(规划等价类≈)．规划 p 和 q 属于同一规划等价类(记作 $p\approx q$)当且仅当 $p_E=q_E$，$p_C=q_C$．

属于同一规划等价类的规划可以合并表示为一个类规划图，规定其中最左子规划图的优先级最高，向右依次降低，并且优先级可以动态调整(规划的成功执行使自身的优先级增加，反之则减少)．

2　静态继承

当通过静态继承从 Agent 父类产生子类时，子类可以增加新的或取消父类中已有的信念、目标和规划，也可以修改或重定义父类的规划．但新增加的信念和目标可能与父类的信念和目标存在矛盾，新定义的规划若与父类的规划同属一个等价类，也存在尝试执行的先后次序问题．按照信念修正的观点，由于新知识获取的途径已经过严格审视，从而是可以确信的．因此，当新旧知识存在矛盾时，总是放弃那些旧知识中与新知识矛盾的部分，这与开放逻辑[6]中信息的非单调更新思想也相吻合．据此可以解决父类与子类之间信念和目标的矛盾．而对属于同一等价类的父子规划的优先级问题存在不同观点．Crnogorac 认为，父类的行为总能经过严格证明是正确的(规划总能正确执行)，而子类的行为尚未确定，如果优先采用子类规划可能导致失败(在高安全实时系统中不能回溯)．但是，通过继承后，父类中经过验证为可靠的行为在子类的环境中能否可靠运行？文献[9]指出，继承并不能简化验证，反而使验证工作复杂化．因此，对高安全实时系统中能否赋予 Agent 继承特性存在置疑．而在允许失败重试的应用中，赋予子类规划以高优先级在实现上是可行的，在效率上也是必要的，因为与父类规划相比，实现同一行为的子类规划在算法上可以针对子类的新特性进行优化．为此，我们采用赋予子类规划高优先级的方法．

定义 2.1(静态单继承的语义)．设$(\mathscr{K}_B,\mathscr{B}_B^I,\mathscr{G}_B^I,\mathscr{P}_B^S)$是$(\mathscr{K}_A,\mathscr{B}_A^I,\mathscr{G}_A^I,\mathscr{P}_A^S)$的子类，在类 B 的规范中，若其知识新定义集为 $\mathscr{K}_B^{\mathscr{N}}$；信念取消集和新定义集为 $\mathscr{B}_B^{\mathscr{A}}$ 和 $\mathscr{B}_B^{\mathscr{N}}$；目标取消集和新定义集为 $\mathscr{G}_B^{\mathscr{A}}$ 和 $\mathscr{G}_B^{\mathscr{N}}$；规划取消集和新定义集为 $\mathscr{P}_B^{\mathscr{A}}$ 和 $\mathscr{P}_B^{\mathscr{N}}$，记 $\mathscr{K}_B=\mathscr{K}_B^{\mathscr{N}}+\mathscr{K}_A$，$\mathscr{B}'=\mathscr{B}_A^I-\mathscr{B}_B^{\mathscr{A}}+\mathscr{B}_B^{\mathscr{N}}$，$\mathscr{G}'=\mathscr{G}_A^I-\mathscr{G}_B^{\mathscr{A}}+\mathscr{G}_B^{\mathscr{N}}$，$\mathscr{P}'=\mathscr{P}_A^S-\mathscr{P}_B^{\mathscr{A}}+\mathscr{P}_B^{\mathscr{N}}$(+和−分别表示集合的并、差运算)，则 $\mathscr{B}_B^I$，$\mathscr{G}_B^I$ 和 $\mathscr{P}_B^S$ 可根据下列约束，从 $\mathscr{B}'$，$\mathscr{G}'$ 和 $\mathscr{P}'$ 得到：

(1) 若 $b(p)\in\mathscr{B}_B^{\mathscr{N}}$，$\forall\ b(q)\in\mathscr{B}_A^I(\vdash_{\mathscr{K}_B\cup\mathscr{B}}b(q)\rightarrow\neg\ b(p))\Rightarrow b(q)\notin\mathscr{B}_B^I$；

(2) 若 $g(p)\in\mathscr{G}_B^A$，$\forall\ g(q)\in\mathscr{G}_A^I(\vdash_{\mathscr{K}_B\cup\mathscr{B}}g(q)\rightarrow\neg\ g(p))\Rightarrow g(q)\notin\mathscr{G}_B^I$；

(3) 若 $p_i\approx p_j$，$p_i\in\mathscr{P}_A^S$，$p_j\in\mathscr{P}_B^S$，则 $p_i\prec p_j$($\prec$表示规划的优先级关系)．

例 2.1：在一个电子市场系统中，Bob 的计算机上有一个 Agent 软件 aide 可为其提供娱乐代理服务．aide 接受 Bob 的指令并进行目标分解，然后进入电子市场与其他 Agent(餐饮代理、影视代理等)会晤协商，电子市场则为各 Agent 提供调度等服务．设在 Bob_aide 的规范中分别定义了 $\mathscr{K}=\{k_1,k_2\}$，其中 k_1 表示"工作日不能娱乐"，k_2 表示"星期一～星期五是工作日"；$\mathscr{B}^I=\{b_1,b_2,b_3\}$，其中 b_1 表示"主人喜欢流行乐"，b_2 表示"主人不爱科幻片"，b_3 表示 $S(Master)$，即"主人喜欢看晚报"；$\mathscr{P}^S=\{p_1,p_2,p_3\}$，其中 p_1 进行电影票预订，p_2 进行餐馆预约，p_3 进行报纸预购；$\mathscr{G}^I=\{g_1\}$，g_1 表示"星期日看一场电影，然后在附近吃顿晚餐"．

Alice 可以通过继承 Bob_aide 生成自己的娱乐代理 Alice_aide．若 Alice 不喜欢流行乐，不喜欢看晚报，喜欢摇滚乐(b_1)，星期日去听音乐会(g_2)，并且在 Alice_aide 的规范中定义信念取消集为$\{b_1\}$和信念新定义集$\{\sim b_1,$

$\sim b_3, b_4\}$，目标取消集为$\varnothing$，目标新定义集为$\{g_2\}$，规划取消集为$\varnothing$，规划新定义集为$\{p_4\}$(p_4用来订音乐会门票)，则静态继承后Alice_aide的规范为$\mathscr{K}=\{k_1,k_2\}$，$\mathscr{B}'=\{b_2,b_3,\sim b_1,\sim b_3,b_4\}$，$\mathscr{P}^S=\mathscr{P}'=\{p_1,p_2,p_3,p_4\}$，$\mathscr{G}'=\{g_1,g_2\}$. 在Alice. aide的信念集和目标集中存在不一致，根据定义进行修正后，$\mathscr{B}^I=\{b_2,\sim b_1,\sim b_3,b_4\}$，$\mathscr{G}^I=\{g_2\}$.

在多继承时，为了对信念和目标进行一致化处理，我们采用在子类中记录信念和目标的继承来源的方法. 例如，类B的信念$b(v)$在其子类A中变为$b(A.b(B.v))$.

当父子信念(目标)集不一致时，以满足子类信念(目标)为前提，去除父类信念(目标)集中与其矛盾的部分. 主要问题在于两个父类的信念(目标)集存在矛盾时如何处理. 我们知道，Agent的多继承是为了实现知识、信念和规划的共享以及功能上的互补. 但当一个Agent类具有两个以上父类时，行为性质的增加必然降低其规划调度和执行的效率. 由于在实际应用中一个实体通常存在主要功能和辅助功能之分，为了提高Agent解决问题的效率，有必要为其父类划分出优先级，在其能够满足自身新定义的行为性质和优选父类的行为性质的前提下，再考虑满足其他父类的行为性质(在动态继承中，根据不同行为性质具有不同优先级这一特点，可以通过择优分割，产生多个Agent实例，从而有效地提高Agent的快速反应能力). 在静态规范中，用$pre(B)$表示B是优选父类，我们有如下定义.

定义 2.2(**静态多继承的语义**). 设$(\mathscr{K}_B,\mathscr{B}_B^I,\mathscr{G}_B^I,\mathscr{P}_B^S)$是$(\mathscr{K}_{A_i},\mathscr{B}_{A_i}^I,\mathscr{G}_{A_i}^I,\mathscr{P}_{A_i}^S)(1\leqslant i\leqslant k)$的子类，其他说明同定义2.1. 记$\mathscr{K}_B=\mathscr{K}_B'+\sum_{i=1}^{k}\mathscr{K}_{A_i}$，$\mathscr{B}'=\sum_{i=1}^{k}\mathscr{B}_{A_i}^I-\mathscr{B}_B^c+\mathscr{B}_B^{\mathscr{K}}$，$\mathscr{G}'=\sum_{i=1}^{k}\mathscr{G}_{A_i}^I-\mathscr{G}_B^c+\mathscr{G}_B^{\mathscr{K}}$，$\mathscr{P}'=\sum_{i=1}^{k}\mathscr{P}_{A_i}^S-\mathscr{P}_B^c+\mathscr{P}_B^{\mathscr{K}}$，则$\mathscr{B}_B^I$，$\mathscr{G}_B^I$和$\mathscr{P}_B^S$可根据下列约束，从$\mathscr{B}'$，$\mathscr{G}'$和$\mathscr{P}'$得到：

(1) 若$pre(A_i)$，$b(p)\in\mathscr{B}_{A_i}^I$，$\forall\ b(q)\in\mathscr{B}_{A_j}^I(1\leqslant j\neq i\leqslant k)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}'}b(q)\rightarrow\neg\ b(p)$，则有$b(q)\notin\mathscr{B}_B^I$(同级父信念相悖时，取消非优选父类的信念)；

(2) 若$\neg\ pre(A_i)$，$b(p)\in\mathscr{B}_{A_i}^I$，$\forall\ b(q)\in\mathscr{B}_{A_j}^I(1\leqslant j\neq i\leqslant k)\wedge\neg\ pre(A_j)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}'}b(q)\rightarrow\neg\ b(p)$，则有$b(q)\notin\mathscr{B}_B^I$，$b(p)\notin\mathscr{B}_B^I$(同级父信念相悖时，若均非优选父类，则同时取消)；

(3) 若$b(q)\in\mathscr{B}_B^{\mathscr{K}}$，$\forall\ b(q)\in\mathscr{B}_{A_j}^I(1\leqslant j\leqslant k)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}'}b(q)\rightarrow\neg\ b(p)$，则有$b(q)\notin\mathscr{B}_B^I$(父子信念相悖时，取消父信念)；

(4) 若$pre(A_i)$，$g(p)\in\mathscr{G}_{A_i}^I$，$\forall\ g(q)\in\mathscr{G}_{A_j}^I(1\leqslant j\neq i\leqslant k)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}_B^I}g(q)\rightarrow\neg\ g(p)$，则有$g(q)\notin\mathscr{G}_B^I$(同级父目标相悖时，取消非优选父类的目标)；

(5) 若$\neg\ pre(A_i)$，$g(p)\in\mathscr{G}_{A_i}^I$，$\forall\ g(q)\in\mathscr{G}_{A_j}^I(1\leqslant j\neq i\leqslant k)\wedge\neg\ pre(A_j)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}_B^I}g(q)\rightarrow\neg\ g(p)$，则有$g(q)\notin\mathscr{G}_B^I$，$g(p)\notin\mathscr{G}_B^I$(同级父目标相悖时，若均非优选父类，则同时取消)；

(6) 若$g(q)\in\mathscr{G}_B^{\mathscr{K}}$，$\forall\ g(q)\in\mathscr{G}_{A_j}^I(1\leqslant j\leqslant k)$，如果有$\vdash_{\mathscr{K}_B,\mathscr{B}_B^I}g(q)\rightarrow\neg\ g(p)$，则有$g(q)\notin\mathscr{G}_B^I$(子类目标优先)；

(7) 若$p_x\approx p_y$，$p_x\in\mathscr{P}_{A_i}^S$，$p_y\in\mathscr{P}_B^S$，则$p_x\prec p_y$；若$p_x\approx p_y$，$p_x\in\mathscr{P}_{A_i}^S$，$p_y\in\mathscr{P}_{A_j}^S$，并且有$pre(A_j)$，则$p_x\prec p_y$.

例 2.2：设Bob的计算机上有另一个Agent软件Bob_meeting可为其提供会议日程安排及代理服务. 在Bob_meeting的规范中定义了$\mathscr{K}=\{k_3,k_4\}$，其中k_3,k_4表示有关会议的常识；$\mathscr{B}^I=\{b_4\}$，b_4表示$\forall\ xM(x)\rightarrow\neg\ S(x)$，即"开会时不能看报"；$\mathscr{P}^S=\{p_4,p_5\}$，其中$p_4$表示进行会议室预订，$p_5$与Bob_aide的$p_2$一样，也表示进行餐馆预约，但因为是集体用餐，需要从特殊的帐号上结帐；$\mathscr{G}'=\varnothing$. 现在Bob要继承Bob_aide和Bob_meeting，生成一个娱乐性会议代理Bob_shedule，设Bob_shedule的初始规范$\mathscr{K}=\mathscr{B}^I=\mathscr{P}^S=\varnothing$；$\mathscr{G}^I=\{g_3\}$，$g_3$表示"星期日开会"，则通过静态多继承后，Bob_shedule的规范为$\mathscr{K}=\{k_1,k_2,k_3,k_4\}$；$\mathscr{B}'=\{b_1,b_2,b_3,b_4\}$；$\mathscr{P}'=\{p_1,p_2,p_3,p_4,p_5\}$；$\mathscr{G}=\{g_1,g_3\}$. 由于$b_4$与$b_3$存在矛盾，因此应删除$b_3$，$\mathscr{B}^I=\{b_1,b_2,b_4\}$. 由于$g_1$和$g_3$存在时间冲突，根据子类目标优先的约束得到$\mathscr{G}^I=\{g_3\}$. 由于$p_5$与$p_2$属于同一规划等价类，根据子类规划优先的约束，$p_5$的优先级比$p_4$高，只有$p_5$执行失败后才能尝试执行$p_2$.

3 动态复制行为

在多Agent系统中，个体Agent之间存在着功能互补关系，当个体Agent不具有某种能力时，就需要与其他

Agent 进行协商和协作. 如果不进行负载均衡，必然会导致某些 Agent 负载过重，而有些 Agent 长时间处于空闲状态. 对此通常有两种解决途径，一是在多 Agent 中设置具有路由、协调功能的 Facilitator，在 Facilitator 中记录各局部 Agent 的能力信息，由 Facilitator 动态控制各 Agent 的任务分配. 但 Facilititor 本身可能成为系统的瓶颈，如果 Facilitator 出现运行故障，则整个多 Agent 系统可能无法正常工作. 另一种方法就是，当一个 Agent 负载过重时，允许该 Agent 进行自我复制，从而生成多个并发运行的 Agent 实例. 我们将 Agent 的自我复制行为看做 Agent 的动态继承. 这种利用 Agent 动态继承来进行负载均衡的方法不依赖于外部因素，充分体现了 Agent 的自治性，是一种较好的解决方案.

Agent 的自我复制涉及到其信念集、目标集及规划集的处理. 我们根据多 Agent 系统的特点及实际需要将 Agent 实例的复制机制划归为 4 种方式. 设 a 为 Agent 类 A 的实例，我们有如下定义.

定义 3.1(功能分割). 设 $\mathscr{P}_a^S=p_1\cup\ldots\cup p_k(p_j(1\leqslant j\leqslant k)$是 Agent 实例 a 的服务簇$)$，$a_i(1\leqslant i\leqslant n)$是 a 的功能分割，如果满足 $\mathscr{P}_{a_i}^S=p_l\cup\ldots\cup p_m(1\leqslant l\leqslant m\leqslant k)$. 当 $l=m$ 时，每个 $a_i(1\leqslant i\leqslant n)$只负责执行一个服务簇中的功能.

以服务簇为单位进行分割可以保证规划的完整性，以避免实现 1 个功能需要多个 a_i 协作的情况. 例如，若 $p_1=\{p_1,p_2\}$，在规划 p_1 中调用了子规划 p_2，如果进行功能分割时将 p_1 和 p_2 分别赋予 a_1 和 a_2，则每次执行 p_1 时都需要与 p_2 协作，反而降低了 Agent 的反应速度.

对 a 进行功能分割时，为了保证 a_i 之间协作的平滑性，a_i 应全部继承 a 的信念集和目标集. 当然，视具体情况也可以部分继承，这需要在具体实现时在 Agent 的个体反应能力和解决问题能力上进行折衷. 分解后的操作集越小，该 Agent 的个体反应能力就越强，但自身解决复杂问题的能力也就越低，对协作的要求则越高.

定义 3.2(逻辑分割). 设 a 的信念集 $\mathscr{B}_a^L=B_1\cup\ldots\cup B_k(B_j(1\leqslant j\leqslant k)$是具有逻辑封闭性的信念子集$)$，$a_i(1\leqslant i\leqslant n)$是 a 的逻辑分割，如果满足 $\mathscr{B}_{a_i}^L=B_l\cup\ldots\cup B_m(1\leqslant l\leqslant m\leqslant k)$. 当 $l=m$ 时，每个 $a_i(1\leqslant i\leqslant n)$具有最小的协调信念集.

以具有逻辑封闭性的信念子集为单位进行逻辑分割，也是为了避免因自身信息不足而需要频繁地向其他 Agent询问的情况. 例如，若 $B_1=\{b_1,b_1\rightarrow b_2\}$，如果进行逻辑分割时将 b_1 和 $b_1\rightarrow b_2$ 分别赋予 a_1 和 a_2，则当 a_2 需要知道 b_2 是否成立时，必须向 a_1 或 a 询问 b_1 是否成立，因而增加了不必要的协作.

逻辑分割的优点是每个实体的推理集减少，降低了局部推理的复杂性，有利于提高局部的执行效率. 由于知识集中包括全域信息(网络拓扑知识等)，是个体 Agent 之间协作的前提，因此，在对 a 进行逻辑分割时，a_i 应当全部继承 a 的知识集. 规划集可以全部继承，也可与功能分割相结合，同时减少 Agent 的推理集和操作集.

定义 3.3(择优分割). $a_i(1\leqslant i\leqslant n)$是 a 的择优分割，如果满足 $a_1\prec\ldots\prec a_i\prec\ldots\prec a_n$，其中 $a_i\prec a_j$表示Agent a_i 的调度优先级比 a_j 低.

由于一个实体的功能通常有主次之分，Agent 在调度规划对外界提供服务时也应针对不同类型的请求，区分出任务的轻重缓急. 尤其是当某项服务体现该 Agent 的社会信誉度时，更应在反应速度和服务质量上对其他 Agent 作出承诺，这时，择优分割就显得异常重要. 如果 a 是某个 Agent 的单继承实例，设 $\mathscr{P}_a^S=p_1\cup\ldots\cup p_k(p_j(1\leqslant j\leqslant k)$是 Agent 实例 a 的服务簇$)$，并且根据不同服务对自己的重要程度赋予每个 p_j 相应的权重(可以相同)，则依权重的大小对 p_j 排序后，形成 n 层服务簇. 这样，进行择优分割时可以产生 n 个子 Agent，每个 $a_i(1\leqslant i\leqslant n)$负责提供具有同一权重值的服务. 在特殊情况下，上述择优分割可能产生与功能分割相同的结果，它们的区别在于，择优分割后的子 Agent 具有不同的操作优先级，而功能分割中各子 Agent 只是进行简单的并发操作.

对于多继承的情况，也可根据 a 中信息的来源进行择优分割：新定义信息；源自优选父类的信息；源自非优选父类的信息. 如果在这 3 个等级中再根据规划的权重划分，还可以形成二维的择优分割.

定义 3.4(返祖分割). 若 A 是 $A_i(1\leqslant i\leqslant n)$的子类(多继承)，$a$ 是 A 的实例，则 a 的返祖分割将产生 n 个分别属于类 $A_i(1\leqslant i\leqslant n)$的 Agent 实例. 即，若 $a_i(1\leqslant i\leqslant n)$是 a 的返祖分割，则 a_i 是 A_i 的实例.

返祖分割不是简单地生成各父类的实例，分割后的 $a_i(1\leqslant i\leqslant n)$需要满足一定的约束关系. 由于 a 在静态多继承时已经对其父类的信念集、目标集和规划集进行了一致化处理和调整，这些修正必然反映在 $a_i(1\leqslant i\leqslant n)$中，只有这样，$a_i$ 之间进行协作求解时才可能体现出 a 的行为性质，而不会违背 a_i 为 a 提供负载分流的初衷. 为

了在 a_i 中反映这种约束关系，a_i 应当全部继承 a 的知识集、信念集和目标集，a_i 的规划集只需继承 $\mathscr{P}_a^S$ 中的 $\mathscr{P}_{A_i}^S$ 部分.

例 3.1：在电子市场系统中，餐饮代理 nestle 负责在电子市场中与客户的代理 Agent 进行业务协商. 设 nestle 的规范为 $\mathscr{K}=K_1\cup K_2\cup K_3$，$K_1$ 表示有关客户私人帐号的信息，K_2 表示单位集体帐号的信息，K_3 表示本餐馆提供的服务类型及价格信息；$\mathscr{B}^I=\{No\_fund(John), No\_fund(Mike), Vacancy(20), Dipso(Smith), \forall x(No\_fund(x)\rightarrow No\_book(x)), \forall x(Dipso(x)\rightarrow No\_book(x))\}$，$No\_fund(x)$ 表示“x 欠帐不付”，$Vacancy(20)$ 表示本餐馆目前有 20 个空位数，$Dipso(x)$ 表示“x 爱喝酒闹事”，$No\_book(x)$ 表示“拒绝为 x 预订”；$\mathscr{P}^S=\{p_1, p_2, p_3, p_4\}$，其中 p_1 表示进行单个客户预订，p_2 表示进行集体客户预订，p_3 表示子规划“将目前的空位数减 1”，p_4 表示“与银行代理协商进行转帐操作”，其中 p_1 和 p_2 都要调用规划 p_3 和 p_4；$\mathscr{P}^I=\emptyset$.

若采用功能分割的方式对 nestle 进行动态继承，由于 $\mathscr{P}^S$ 有两个服务簇 $\{p_1, p_2, p_4\}$ 和 $\{p_2, p_3, p_4\}$，可将 nestle 分为两个 Agent，分别负责单个客户和集体客户的预订操作，$\mathscr{K}$，$\mathscr{B}^I$ 和 $\mathscr{P}^I$ 可保持不变. 若采用逻辑分割的方式对 nestle 进行动态继承，为保障 $\mathscr{B}^I$ 的逻辑封闭性，可将 $\mathscr{B}^I$ 分为 $\{No\_fund(John), No\_fund(Mike), Vacancy(20), \forall x(No\_fund(x)\rightarrow No\_book(x))\}$ 和 $\{Vacancy(20), Dipso(Smith), \forall x(Dipso(x)\rightarrow No\_book(x))\}$ 两个子集，其他心智状态集合保持不变，从而生成两个子 Agent；也可结合功能分割，并对 $\mathscr{K}$ 进一步划分（$K_1\cup K_3$ 和 $K_2\cup K_3$），从而生成 8 个子 Agent，分别提供不同的服务. 若某餐馆规定集体预订的优先级高于个人预订的优先级（不同的应用有不同的择优标准），则采用择优分割的方式对 nestle 进行动态继承时，可生成两个子 Agent 分别进行集体预订和个人预订服务. 在本例中择优分割的结果与功能分割相同，但在择优分割中若存在资源（空位数）竞争，则依优先级高低进行抉择，而在功能分割中可随机抉择. 设 Agent 类 *Music* 定义了卡拉 OK 代理的静态规范，*Rest* 定义了餐饮代理的静态规范，nestle 是通过继承 *Rest* 和 *Music* 产生的 Agent 实例，同时提供餐饮和卡拉 OK 代理服务，则在采用返祖分割的方式对 nestle 进行动态继承时，将生成两个代理 nestle1 和 nestle2，分别是类 *Music* 和 *Rest* 的实例.

综上所述，Agent 动态继承的语义体现在 Agent 实例的动态复制中，据此可以写出 Agent 实例的动态复制算法.

4 有关问题的讨论

为 Agent 引入继承后，一方面，具有相似或相关抽象行为的 Agent 可以共享大部分实现代码，从而降低编程的复杂性；另一方面，便于多 Agent 系统的模块验证，使得子类性质的验证可以建立在父类已有性质的基础之上. 即，如果 A 的某一行为特性已经验证，则其子类 B 的相关性质可直接从 A 出发，减少测试、验证的工作量.

文献[5]基于行为保持和继承异常对 Agent 的继承性进行了研究. Agent 的行为保持是指子类不能丧失父类的行为性质，否则，子类的性质验证必须从头做起. 当继承机制不足以为具有相关抽象行为的 Agent 提供实现代码共享的手段时，就产生继承异常问题. 行为保持有利于子类性质的验证；继承异常有悖于代码复用的初衷.

但是，行为的部分丧失的确客观存在于自然界和符号世界之中（编程人员有时需要冲破行为保持的限制）. 从物种进化的观点看，生物受环境的影响在新特征形成的过程中总是伴随着旧特征的遗忘或退化. 此外，继承究竟能为 Agent 的性质验证带来多大好处尚存疑虑. 由于 Agent 的性质验证与其测试密不可分，因此，我们有必要看一下继承对测试的影响. 文献[9]的结果表明，若一个类得到了充分的测试，当其被子类继承后，继承的方法在子类的环境中的行为特征需要重新测试. 换句话说，继承并未简化测试，反而使测试工作复杂化.

当然，由于部分行为丧失是一种跳跃式继承，虽然子类的性质不能依赖于跳变之前父类的性质，但在两次跳变之间的类层次中，父类可为子类的性质验证提供依据. 因此，如果继承确实能简化 Agent 的性质验证，则具有部分行为丧失的继承同样可以避免 Agent 性质验证中的寻根究底，从而减少大量的验证工作.

因此，我们不能仅仅因为行为保持为 Agent 性质验证所带来的部分便利，而排除 Agent 在继承中部分行为丧失的可能. 事实上，在实际 Agent 系统设计时，Agent 的部分行为丧失反而能为实现提供很大便利.

与行为保持相比，继承异常则是 AOP 语言设计者应关注的主要问题. Crnogorac 等人认为[5]，若 b 继承 a，则

b 中与 a 有关的所有行为必须能够从 a 通过增量式转换导出.显然，只有单调增量式继承才能严格杜绝继承异常问题.然而，新行为(规划)的增加与原有行为的更新是必不可少的.对于这种大量存在于实际系统的非单调继承，或多或少地会存在继承异常问题.因此，既然继承异常不能避免，不如在进行 AOP 语言设计时就放宽对继承异常的语法限制，而将是否避免继承异常留给编程人员根据实际需要来进行抉择.

5　结束语

当通过继承从 Agent 父类产生子类时，可以增加新的或取消父类中已有的信念、目标和规划，也可以修改或重定义父类的规划.因此，Agent 继承涉及到信念、目标和规划的多重考虑，比对象继承复杂得多.此外，Agent 的多继承除了要处理不同父类之间、父类与子类之间的心智状态冲突，还需要处理同一规划等价类中规划的优先级等问题.本文基于 BDI Agent 模型研究了软件 Agent 的静态继承和动态继承，形式定义了单继承和多继承的语义.采用优先级的方法保证了规划语义的无歧义性和信念、目标的协调性.对于 Agent 的动态继承，我们将 Agent 实例的复制行为划归为 4 种类型：功能分割、逻辑分割、择优分割和返祖分割，分析了不同分割方式的作用，并基于电子市场系统应用背景给出了相应的实例.

参考文献

1 Rao A S. AgentSpeak(L): BDI agents speak out in a logical computable language. In: Vande Vlde W, Perram J W eds. Proceedings of the 7th European Workshop on Modeling Autonomous Agents in a Multi-agent Word (MAAMAW'96)——Agents Breaking Away. LNAI 1038, Berlin: Springer-Verlag, 1996. 42～55

2 Shoham Y. Agent-oriented programming. Artificial Intelligence, 1993,60(1):51～92

3 Burkhard H D. Agent-oriented programming for open systems. In: Wooldridge M, Jennings N R eds. Proceedings of the 1994 Workshop on Agent Theories, Architectures, and Languages (ATAL'94)—— Intelligent Agents. LNAI 890, Berlin: Springer-Verlag, 1995. 291～306

4 Wooldridge M. Agent-based software engineering. IEEE Transactions on Software Engineering, 1997,144(1):26～37

5 Crnogorac L *et al*. Analysis of inheritance mechanism in agent-oriented programming. In: Martha E P ed. Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI'97). San Fransisco: Morgan Kaufmann Publishers, Inc., 1997. 647～652

6 Wooldridge M, Jennings N R. Intelligent agents: theory and practice. The Knowledge Engineering Review, 1995,10(2):115～152

7 Rao A S. Decision procedures for propositional linear-time belief-desire-intention logics. In: Wooldridge M, Muller J P, Tambe eds. Proceedings of the IJCAI'95 Workshop on Agent Theories, Architecture, and Languages—— Intelligent Agent. LNAI 1037, Berlin: Springer-Verlag, 1996. 33～48

8 李未. 一个开放的逻辑系统. 中国科学(A 辑),1992,35(10):1104～1113
(Li Wei. An open logic system. Science in China (series A), 1992,35(10):1104～1113)

9 Perry D E, Kaiser G E. Adequate testing and object-oriented programming. Journal of Object-oriented Programming, 1990,2(5):13～19

Research on Inheritance of Software Agent

FAN Xiao-cong　XU Dian-xiang　HOU Jian-min　ZHENG Guo-liang

(*State Key Laboratory for Novel Software Technology　Nanjing University　Nanjing*　210093)
(*Department of Computer Science and Technology　Nanjing University　Nanjing*　210093)

Abstract　Being a kind of restricted intelligent objects, agent is a natural way to research the inheritance feature of software agent and integrate inheritance mechanisms irto AOP (agent-oriented programming). Based on BDI model of agents, the semantics of inheritance and cloning behavior of agents are addressed in this paper. The semantics of inheritance are discussed from two aspects: single inheritance and multiple inheritance. For cloning behavior, the authors identify and formally classify the dynamic cloning mechanisms of agent instances into four types: function split, logic split, preference split and retrogress split. The principle of each cloning mechanism is presented and the examples are provided based on the electrical commerce systems.

Key words　Software agent, inheritance, cloning behavior, BDI model, plan.