基于 Agent 技术的智能软件协同开发研究

卢正鼎　霍晓丽

（华中科技大学计算机科学与技术学院　武汉 430074）

摘　要　结合协同开发和 Agent 技术，构造了一个基于 Agent 组件的智能软件协同开发模型 CMISA，对模型的组成元素进行了语义定义，并阐述了模型的设计思想。实现从传统软件结构到智能软件结构的转变，从实体单元的被动性到主动自主性的转变，使开发出来的软件具有典型的智能特征。

关键词　Agent 组件，智能软件，协同开发，CMISA 模型

Study on the Cooperation Intelligent Software Development Based on Agent Components

LU Zheng-Ding　HUO Xiao-Li

(Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　Agent components provide a significant tool for the development of intelligent software because of it's intelligence and self-organizing. This essay analyzed the advantages of Agent components comparing with ordinary components and established a Cooperation Model of Intelligent Software Development based on Agent Component(DMISA). Then the semantic definition for component elements of CMISA is proposed and the designing philosophy of model is stated. The preliminary design of cooperation mechanism in CMISA is made according to different situations. Finally, the problems required to notice in the process of cooperation development of intelligent software are proposed from four aspects.

Keywords　Agent components, Intelligent software, Cooperation development, CMISA model

1　引言

随着大型项目开发人员人数的增加，通信的复杂性也会随之增大，其“增大的程度与所增加人数的平方成正比”。Bandinelli 等人指出，由于软件开发具有协作性，开发成功与否，依赖于“建立在开发团队中的通信渠道的质量和效果”。通信固然是重要的，同时，为了提高团队工作的效率，将重心从成员间的协调转移到成员的协同工作也非常重要。利用计算机支持的协同工作 CSCW(Computer Supported Cooperative Work)进行软件开发(即在计算机技术支持的网络环境中，一个地域分散的群体协同工作完成一个软件的开发任务)可以提高工作效率，缩短开发周期。

Internet 环境异构、开放、动态和多变的特征导致软件系统开始呈现出一种智能性、多目标、连续反应式的新系统形态。在软件开发方法上，最初的模块结构化开发已经被后来的面向对象(OO)的开发方法所替代，OO 方法后来进一步演进为面向过程的开发，如今又被基于组件的软件开发思想所取代。在软件功能上，计算机软件已经从单机应用发展到 Internet 集成系统，经过了涵盖多功能、大而全的阶段后，如今又向着智能化的方向发展。基于组件的开发方法是软件开发的一次历史性革命，也是计算机技术日新月异的必然结果。

2　Agent 组件相对于普通组件的特点

基于组件的软件开发是一种用来提高复用水平的最有前途的软件工程技术。组件是个很复杂的概念，文献中的各种定义都不能全面概括它。简单地说，组件是一个能提供特定功能，具有开放接口的可以实现“即插即用”的软件。这里谈到的组件既可以包括程序代码、设计过程等核心元素，也可以包括测试用例、设计文档甚至领域知识等边界元素。它们或者是构成整个目标软件系统的部件，或者在软件开发过程中发挥某种作用。基于组件的开发思想就是在软件开发过程中，应尽可能重用已有的软件元素，这样可以减少软件开发活动中大量的重复性工作，能够提高软件生产率，降低开发成本，缩短开发周期。组件和应用程序的连接是通过接口来实现的，一个组件具有若干个接口，每个接口代表组件的某个属性或方法，其他组件或应用程序调用这些属性和方法来进行特定的逻辑处理。负责集成的开发人员无需了解组件功能是如何实现的，只需简单地创建组件对象并与其接口建立连接。

而 Agent 组件是一种特殊的组件形态。Agent 通常指在一定的环境下持续自主运行的实体，是一种能准确行动以代表用户完成任务的一些软件和硬件的组合。Agent 技术起源于人工智能，其概念在 20 世纪 60 年代提出，真正的发展在 20 世纪 90 年代中期。Agent 思想最初来源于 John McCarthy 提出的“The Advice Taker”系统，是被设想为具有目标性，系统内实体间用人类的术语进行交流，它们从用户利益来考虑从事各种任务。广义上它是指具有智能的任何实体。

Agent 组件具有一般组件的典型特征，即内部封装、功能独立，同时具有标准化的接口。Agent 组件无论从概念上还是实现上来讲都是一种封装模型，其内部结构和算法可以由不同的人在不同的时间和地点采取不同的方法加以实现，并像“黑匣子”一样可以被装配到任何合适的软件系统中。但与普通组件相比，Agent 组件具有如下特点：

卢正鼎　博士生导师，主要研究方向：软件工程；霍晓丽　博士生，主要研究方向：软件工程。

(1)智能性。智能性与普通组件具有的自主性不同,自主性只是被动地进行自我调整、做出反应,无法主动进行某些行为,而智能性却使 Agent 组件具有优良的判断力或合理的"思维"能力。Agent 组件的智能性集中体现在两个方面:学习机制——从以前的行为结果或别的 Agent 组件那里进行学习,改变自身的行为策略,使自己的功能不断得到完善。协作能力——使多个 Agent 组件在一个复杂的软件系统中按照一定的行为策略和协作机制,既完成每个 Agent 组件的功能,又发挥系统的整体功能。

(2)自适应性。Agent 组件除了能够按照预定程序完成相应功能之外,还可以根据不同的软件环境做出相应的调整,适应环境并执行相应的功能。这是 Agent 组件自适应性的集中体现,也是其他普通组件所不具有的特殊功能。对于环境的变化和不确定因素,Agent 组件可以通过在协调机制下交互和自学习适应新的条件,保持系统的健壮性。

正是由于 Agent 组件具有以上两个特点,因此在开发具有特殊需要的软件系统尤其是智能软件方面,Agent 组件有着先天的优势。

3 基于 Agent 组件的智能软件协同开发——CMISA

3.1 CMISA 的相关定义

智能软件由于其智能化的特点,使得它的开发过程与一般软件的开发过程颇有不同。由于智能软件的基本组成元素——Agent 组件能够根据外界的变化作出主动性的反应,因此,开发智能软件时需要充分发挥和注意 Agent 组件的这种特点,使开发出的软件具有最大的智能性。本文在多 Agent 协商、协作的基础上构造了一个基于 Agent 组件的智能软件协同开发模型——Cooperation Model of Intelligent Software Development Based on Agent Component,CMISA。在 CMISA 模型中,我们把所有参与执行软件功能的 Agent 称为 Agent 集合。集合中各个 Agent 都是自治的,能相互通讯,进行协商,达成协议。一项需要实现的软件功能可以用一个二元组来表示:

$Func_i=(Ag_i,Resource_i)$

有关 CMISA 中的功能及 Agent 组件执行功能所需要的资源及相关性质和定义描述如下:

定义 1(功能 Func) 功能 Func 是 CMISA 中 Agent 组件要完成的目标,它是一个如下表示的集合:

$\langle F_{a_1},\cdots,F_{a_n}\rangle$

功能 F 可以是某一个 Agent 组件要完成的目标,也可以是 CMISA 中多个 Agent 组件组件或整个软件系统要完成的目标。整个智能软件系统要完成的目标我们用 FUNC 表示,显然有 $FUNC=\sum_i func_i$。

定义 2(资源 Resource) $f(Func)\rightarrow R^+$。资源 Resource 是执行功能 Func 的花费,它是一个从 Func 到正实数 R^+ 的一个映射。资源 Resource 可以是时间或计算机系统硬件资源等的耗费,Resource 满足如下性质:

性质 1 资源 Resource 是单调的,可以定义成如下形式:

如果 F_{a_1},F_{a_2} 属于 $Func$ 是软件功能的集合,使得 $F_{a_1}\subseteq F_{a_2}$ 则 $Resource(F_{a_1})\leqslant Resource(F_{a_2})$。直观上,资源 Resource 的单调性可以理解为:增加功能 Func 不可能降低资源。

性质 2 在不结成联盟的条件下,整个 Agent 组件系统完成目标的所有资源是各 Agent 组件独立完成的资源之和,即 $RESOURCE=\sum_i resource_i$。

性质 3 不执行任何功能的资源是零,即 $Resource(\Phi)=0$。

CMISA 系统中,整个多 Agent 组件集合形成了一个开放的系统,该系统所要实现的功能是 Func,它的整体资源是 Resource。与基于普通组件的软件系统不同,CMISA 系统由于其智能性的特点,要求系统中必须有一个核心 Agent 组件担负"领导"功能,该 Agent 组件除了要对其他 Agent 组件分配任务外,还要担负组织协调的功能,以保证整个智能软件系统能够协调、有序地运行。我们把担任这种角色的 Agent 组件称为核心 Agent 组件。于是,每个 Agent 组件(Ag_1,Ag_2,…)都有了自己的 Func 和 Resource。事实上,这些独立的 Agent 组件之间并非没有联系,它们之间有协商、有协作,所以它们之间有可能形成系统内的子系统——联盟。

定义 3(联盟 Union) 联盟是一组合作的、共同实现某一功能的 Agent 组件集合,即 $Union=\sum_i Ag_i$。

CMISA 中联盟可能有多个,每个 Agent 组件联盟有多个 Agent 组件,每个 Agent 组件也可能同时属于多个联盟。联盟成员总是试图增大其联盟效益(最大限度实现软件功能),即联盟成员总是选择能使联盟取得最大值的功能。联盟形成时不需中央决策,形成联盟时的通讯开销和计算工作量仅限于联盟参与者。

假设某 CMISA 中有 n 个 Agent 组件,这些 Agent 组件未结成联盟时,各 Agent 组件所承担的功能分别为:$func_1$,$func_2$,…,$func_n$;它们享有的资源分别为:$resource(Ag_1)$,$resource(Ag_2)$,…,$resource(Ag_n)$。如果其中的两个或者几个 Agent 组件结成了联盟(比如 Agent 组件 i 和 Agent 组件 j 结成了联盟),则根据常识有:

$resource(Ag_i+Ag_j)<resource(Ag_i)+resource(Ag_j)$

而且由于 Agent 组件 i 和 Agent 组件 j 结成了联盟,它们在实现 $func_i$ 与 $func_j$ 这两个功能时可以进行有效的沟通、协商和协作,因此二者联合行动所实现的功能效果要比这两项功能单独实现要好,也就是产生了所谓的"1+1>2"的效果。如果整个 Agent 组件系统能够形成一个大的联盟,各个 Agent 组件之间互相协商、协作,取长补短,那么整个软件系统所耗费的 Resourcf 就是最小的,而 Func 的实现效果却是最大的。这也就是所谓的"团队"概念。所以我们应该鼓励多 Agent 组件之间的协商、协作,充分发挥其智能特点,使各 Agent 组件形成实现软件功能的合力,达到功能效果的最大化。

3.2 CMISA 的协同工作机制

3.2.1 核心 Agent 组件与其它 Agent 组件之间的协同机制设计

核心 Agent 组件与其它 Agent 组件之间的协同机制如图 1 所示。图中,核心 Agent 组件将总体功能进行分配,各 Agent 组件或通过独立方式或通过联盟方式实现自己所承担的功能后,将结果反馈给核心 Agent 组件,核心 Agent 组件收到各 Agent 组件的功能实现结果后,对于需要综合考虑的可以进行进一步的协调处理。核心 Agent 组件在 Agent 组件集合中扮演着管理者和协调者的双重角色,也是团队的核心。它在实现软件整体功能 Func 的过程中主要解决两大问题:一是任务分配问题,即确定什么 Agent 组件应该完成什么任务;二是 Agent 组件之间的结果共享问题,包括共享任务执行结果信息以及 Agent 组件对环境的观察信息。

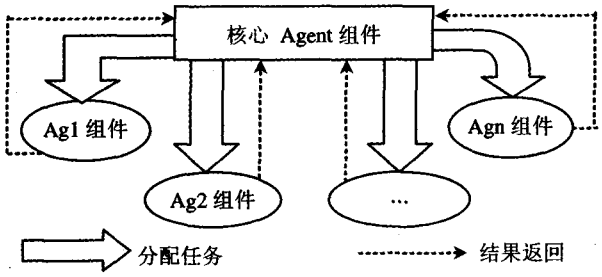

图 1　核心 Agent 组件与其它 Agent 组件协同机制示意图

(1)Agent 组件任务分配问题。任务分配方式是指各 Agent 组件通过分担执行整个任务的子任务而相互协作，系统中的控制以目标为指导，各 Agent 组件的处理目标是为了实现整个软件功能的一部分。这种方式假定其核心子任务可由单个 Agent 组件以独立工作方式解决，采用这种方式的系统要求对任务进行适当的分解，并能够快速地对 Agent 组件和任务进行有效的匹配，最终使功能得到实现。它适宜于实现具有层次结构的功能，要求总体功能能分解成多个独立的子功能，且彼此之间的关联必须尽可能少，即尽量减少功能之间的耦合度。

(2)各 Agent 组件的结果共享。当各 Agent 组件的子功能实现后，有一个功能的综合过程，即各 Agent 组件的结果共享。结果共享方式是指 Agent 组件之间通过接口共享部分结果的互相协作，它基于从某些不同的角度对整个问题的分析和看法。在这种方式的系统中，核心 Agent 组件作为全局规划者(或协调者)，其它的 Agent 组件向此 Agent 组件发送信息，通过相互协作构造完成功能的整体计划。这种方式的代价是 Agent 组件间的通信量比较大。它适宜于实现各子功能的结果相互影响，并且部分结果需要综合才能实现功能的情形。

事实上，这两个问题只是实现功能的不同阶段，在构造软件系统时，往往将两者结合起来使用。

3.2.2　其它 Agent 组件之间的协同机制设计

其它 Agent 组件之间的协同机制有两种方式，一种是被动的，如图 2 所示。在被动协同机制中是以某 Agent 组件的需求为导向的，即某个 Agent 组件在接受要完成某个功能 Func 后首先判断：自己能否独立完成该任务？如果能，就按部就班地去完成；如果不能，就必须寻求其它 Agent 组件的帮助，即搜寻可以求助的 Agent 组件。找到目标 Agent 组件后，它可以与之先进行协商，通过沟通交流把问题表达清楚，并询问对方是否有帮助的意愿。之后二者可展开协作，共同攻克问题的难关，直至最后任务完成。而在图 3 所示的主动协同机制中，各 Agent 组件充分发扬“主动帮助”的精神，在完成自身任务的同时，还主动通过系统询问其它 Agent 组件是否需要帮助？如果没有，只需完成自己的任务即可；如果搜寻到有需要帮助的 Agent 组件，就会主动上门与之协商交流，搞清需要帮助的地方并以双方协作的方式最终完成任务的执行。需要注意的是，主动协同机制中，主动提供帮助的 Agent 组件必须能够保证按时完成自身的任务，然后才能用“余力”去帮助其它 Agent 组件。

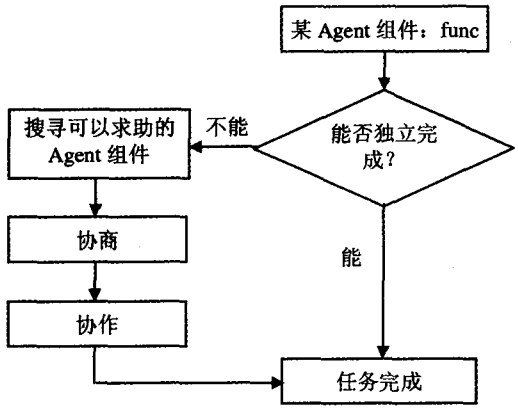

图 2　被动协同机制

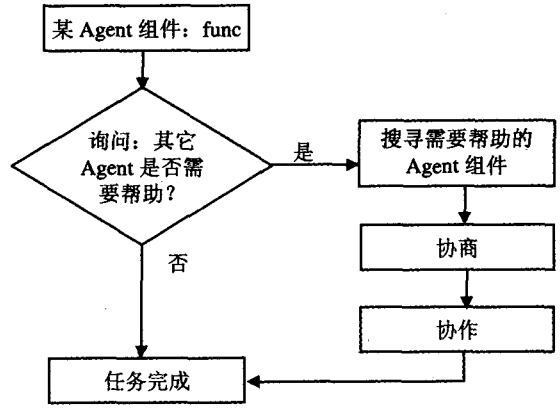

图 3　主动协同机制

4　基于 CMISA 的智能软件开发过程需要注意的问题

(1)合理确定核心 Agent 组件

从前文所述的 CMISA 工作机制可以看出，核心 Agent 是一个非常重要的角色，因此要慎重选择和设计核心 Agent 组件的功能，既要保留足够的接口与其它 Agent 组件通讯，又要具备较强的组织、协调能力，以充分调动其它 Agent 组件的自身能力。具体操作中，可以组织软件开发人员根据实际的需求分解任务，找出不同功能的共性部分和特有的部分，将共性部分设计为核心 Agent 组件，特有部分设计为其它 Agent 组件，对组件模块逐层细化，得出每个组件的概要设计方案，建立系统的软件体系结构，并创建一份正确、完整的组件功能规划设计文档。

(2)合理划分各 Agent 组件功能

确定各 Agent 组件的功能时，组件粒度不宜过大。除核心 Agent 组件之外，尽量让每一个 Agent 组件实现某一个或一类相似的应用请求，而不要追求其功能的过分繁多，保证每个 Agent 组件完成的逻辑功能相对单一。其它的统筹协调工作一律交给核心 Agent 组件去完成。这样，当需求发生变更时，只需要更换中间层的个别 Agent 组件而不会影响其他 Agent 组件的继续使用，有助于充分发挥重用机制。规划组件功能时要统一技术标准、规范通信协议，满足可重用、可裁减和“即插即用”的要求。

(3)合理设计 Agent 组件接口

与普通组件一样，Agent 组件必须通过接口才能实现其价值。尤其是对于智能软件，确定组件的接口要经过深思熟虑，根据需求确定组件的对外接口，即组件支持的事件、方法和属性，使它能支撑各种不同的智能性应用。接口设计时应具有较高的通用性，以提高整个软件系统的复用能力，同时还要尽量估计到将来可能出现的各种情况，力争设计出具有高复用性、适应性和灵活性的接口。

(下转第 249 页)

节点进行搜索，直至搜索到汇点。

设开始搜索的库所标号为1，用 S_1 表示所求 S-封闭基本有向贯通路簇中库所的集合，用 T_1 表示所求 S-封闭基本有向贯通路簇的变迁集合，以 k 作为序号，v 为正在检查的库所，w 为待检查的库所，$N(i)$ 为给第 i 个库所的标号。

算法如下：

```
Step1:if 网 N 存在源点集和汇点集{
Step2:then while 网 N 中源点集中存在未被搜索的源点时 do{
Step3:任取一个未被搜索的源点 s,记为 v,令 v:=1;k:=1;N(1):=
      1;S1={v};T1=∅;
Step4:if v 存在未被检验的输出弧/*寻找没有被检验的输出弧*/
       then 取与 v 输出弧关联的第一个未被检验的变迁,设为 t;
Step5:  if t·≠∅ then{从 t· 中选择一个未被检验的库所记为 w;
        S1=S1∪{w};T1=T1∪{t};
        goto step8;  }
Step6:        else 输出:"从源点 s 开始没有 S-封闭基本有向贯通路"
Step7:else/*若没有这样的输出弧,即库所 v 的每一个输出变迁都检
          验过了*/
          goto step10;
Step8:if w 是未被访问过的库所,即 N(w)尚未确定
       then{令 v:=w; k:=k+1; N(w):=k;
       goto step4;}
Step9:  else /*w 是被访问过的库所,即 N(w)≠0*/
         goto step4;
Step10:if 最后被搜索的库所 v 是汇点 then {
Step11:  if (·v)≠∅∩·(·v)=∅ then
           输出:"从源点 s 开始没有 S-封闭基本有向贯通路";
Step12:else v=·(·v); goto step4;}
Step13:else/*最后搜索的库所不是汇点*/
          输出:"从源点 s 开始没有 S-封闭基本有向贯通
```

3.2 网 N 的有向回路求解

对于网 N 的有向回路求解时，我们基于保留库所的输入与输出弧不变的思想将网 N 进行改造后再应用文[10]、[11]或是文[12]中提出的方法可以得到网 N 的所有的有向回路。

对网 N 的改变方法是我们将任意变迁 t 的 $\cdot t$ 中所有库所与 $t\cdot$ 中的所有的库所均用有向弧来连接，省略了变迁 t，则得到的最终的图形是由所有库所形成一个有向图。

3.3 S-封闭基本有向回路簇求解步骤

(1) 任选一个有向回路 c，对于 $\forall s\in c(S,T)$ 检查其封闭性即 $\cdot s\cup s\cdot\in c(S,T)$ 即是否成立，并用 SC 记下所有不封闭的库所集合；

(2)对于 $\forall s\in SC$ 依次从剩下的有向回路集 $\{c_1,c_2,\cdots,c_k\}$ 中寻找包含它的有向回路，当有多条有向回路 $\{c_{i1},c_{i2},\cdots,c_{il}\}$，(其中 $i1,i2,\cdots,il\in\{1,\cdots,k\}$)满足条件时，我们从中选择 $\max(|c(S,T)\cap c_{ij}(S,T)|)$(其中 $j=1,2,\cdots l$，并且 $|c(S,T)\cap c_{ij}(S,T)|$ 表示两个有向回路相交节点的个数)这样的有向回路和有向回路 c 合并；

(3)设合并后的有向回路簇记为 CN，我们再对有向回路簇 CN 进行封闭性检查把不封闭的库所加进 SC 中，返回到步骤 2；

(4)直至合并后的有向回路簇中所有的库所都是封闭的。

总结 本文所提出的通过 S-封闭基本有向贯通路簇和 S-封闭基本有向回路簇来求解 Petri 网的 S-不变量有很大的方便，不同于以前的求解方法那么复杂，此方法直观可见。尤其是对于那些网结构不是很复杂的网来说，求解时就更加方便。由于 T-不变量和 S-不变量是对偶的关系，对原网的 T-不变量的求解可以转化为对其对偶逆网的 S-不变量的求解，所以此方法也能用于求解 T-不变量。

参 考 文 献

1 Peterson J L 著. 吴哲辉译. Petri 网理论与系统模拟[M]. 徐州：中国矿业大学出版社，1989
2 袁崇义著. Petri 网原理[M]. 北京：电子工业出版社，1998
3 吴哲辉. Petri 网导论[M]. 北京：机械工业出版社，2005
4 殷剑宏，吴开亚 图论及其算法[M]. 合肥：中国科学技术大学出版社，2003
5 Murata T. Petri Nets: Properties, Analysis and Applications [A]. In: Proceedings of IEEE, 1989. 354～355
6 Martinez J, Silva M. A Simple and Fast Algorithm to Obtain All Invariants Of a Generalized Petri Nets[J]. In: Proceedings of Second European Workshop on Application and Theory of Petri Nets, Informatik Fachberichte 52, Springer Publishing Company, Berlin, 1982
7 Takata M, Matsumoto T, Moro S. A Direct Method to Derive All Generators of Solutions of a Matrix Equation in a Petri Net [J]. In: The 2002 International Technical Conference on Circuits/systems, Computers and Communitions, 2002
8 Yamauchi M, Wakuda M, Taoka S, Watanabe T. A Fast and Space-Saving Algorithm for Computing Invariants of Petri Nets. IEEE
9 曾小伟，陈吉红，向华. 计算 Petri 网的 S-不变量和 T-不变量算法[J]. 华中科技大学学报，2001，29(11)
10 毕双艳，郭金梅，齐明. 有向图中初级有向回路的求解算法及实例[J]. 长春邮电学院学报，1999，17(2)
11 熊德琰. 生成有向图的有向回路基集及全部有向回路的一个搜索算法[J]. 电子学报，1986，14(6)
12 杜树贵. 生成有向图的有向通路和有向回路的一个新算法[J]. 电子与系统学报，1999，4(4)

(上接第 210 页)

(4)合理设计应用数据库

智能软件的应用离不开以数据库为平台，数据库设计不合理，数据得不到合理有效的存储，数据就会存在潜在的不一致性、不完整性或有大量冗余，造成系统性能降低，甚至使系统崩溃。数据库的设计要符合现行的数据标准和管理规范，建立一个统一的、权威的、规范的数据标准及数据规范，充分利用现有的硬件和软件平台，包括现有的网络、服务器、操作系统、高级数据库语言和编程语言，同时要有一定的容错和纠错能力。

结束语 本文从开发智能软件的角度分析了 Agent 组件的特点，并提出了一个基于 Agent 组件的智能软件协同开发模型——CMISA，从理论上对模型的思想和原理进行了定义和设计，并进一步阐述了开发过程中需要注意的问题。需要指出的是，本文只是提出了一种软件设计的初步设想，论述了开发智能软件的基础理论框架，为智能软件开发提供了新理论、新方法的尝试。实际上，智能软件的开发不仅仅需要用到计算机知识，要想软件真正具有人性化的思维能力，就不可避免地要涉及到生理认知、生物科技等领域，还需要智能心理学做理论基础。因此，开发具有高级智商的智能软件将会成为人类科技的重大挑战。

参 考 文 献

1 Ma X X, Lu J, Tao X P, et al. A mobile-agent-based approach to software coordination in the HOOPE system. Science in China, Series F, 2002, 45(3): 203～219
2 Chen T L, Han T T, Lu J. A model logic for pi-calculus and model checking algorithm. Electronic Notes in Theoretical Computer Science, 2005, 123: 19～33
3 McConnell S. Rapid development [M]. Microsoft Press, 1996
4 Bandinelli S, Nitto E D, Fuggetta A. Supporting cooperation in the SPADE -1 environment [J]. IEEE Transactions on Software Engineering, 1996, 22(12) : 841～865
5 吕建，张鸣，廖宇. 基于移动 Agent 技术的构件软件框架研究. 软件学报，2000，11(8)：1018～1024
6 翁南钐. 基于组件的软件工程及其测试、维护与实践. 计算机工程与应用，2000(2)：33～36
6 张健，曾广周，杨鹏. 面向 Agent 软件工程研究现状与展望. 计算机工程与应用，2006(15)：30～33