doi:10.11887/j.cn.201403019 http://journal.nudt.edu.cn

支持社会技术系统构造的程序设计模型和语言*

毛新军,陈 寅,张婷婷
(国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:互联网上越来越多软件系统呈现出社会与技术交融、环境多样和开放、系统动态和演化等特点,如何支持这类复杂软件系统的构造是软件工程面临的一项重要挑战。通过将社会技术系统视为多Agent组织,把社会组织学的概念和思想引入到面向Agent程序设计范畴,提出了以Agent、组织、角色和职位为核心的基于组织程序模型来支持社会技术系统的开发;通过对多Agent组织不同层次动态性的识别和分析,设计了一组程序设计机制来实现社会技术系统的动态演化;包括角色绑定和组合机制、基于角色的交互机制以及组织的自我管理机制。基于上述模型和机制,提出了社会技术系统程序设计语言OragentL,给出了OragentL程序在组织、角色和组合三个层次的语法形式定义,介绍了OragentL的编译器及其运行支撑环境OragentBurg,并通过案例分析和演示展示了研究成果的有效性。

关键词:社会技术系统;多Agent组织;基于组织的程序设计
中图分类号:TP391 **文献标志码**:A **文章编号**:1001-2486(2014)03-0103-08

The programming model and language for constructing socio technical systems

MAO Xinjun, *CHEN Yin*, *ZHANG Tingting*
(College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract: Software systems situated in open environment like the Internet are increasingly characterized as socio technical systems that evolve dynamically various variations and changing requirements. To construct such complex software system has become an open issue in the literature of software engineering. This research takes socio technical systems as multiple agent organizations and presents an organization-based program model ORAP that introduces organization metaphors like organization, agent, role and position to construct complex socio technical systems. A series of programming mechanisms like role binding and composition, role-based interaction and self-management of organization were designed to tackle the dynamics issues of socio technical systems by categorizing the organization dynamics at different organization levels. Furthermore, an organization-based programming language called OragentL for implementing social technical systems were proposed, which involves its syntax formal definition at organization, role and composition levels respectively. The OragentL's compiler and running environment called OragentBurg were introduced and a sample was studied to illustrate our approach and show its effectiveness.

Key words: socio technical system; multi-agent organization; organization-base programming

随着互联网、社交网、网络计算等技术的快速发展及其之上各种应用的日益普及和深入,软件系统无论在构成、规模、形态、特征等方面都发生了深刻变化。驻留环境已从传统的封闭、静态和单一的计算环境(如单机或者局域网络环境)转变为以互联网、物理信息交融、社会技术交融等为代表的开放、动态和多样化环境,并具有不确定、难控、不可预知等方面的特点。软件系统不仅在规模方面(如代码行、节点数目、参与人数、数据量等)出现了急剧增长,出现了规模超大的系统(Ultra-Large Scale system, ULS)[1],而且呈现出开放、自主、自适应、自组织、层次化、发散、持续演化、宏观涌现等方面的特点。软件系统的外在形态发生了很大的变化,越来越多的软件系统是现实社会系统(如社交、企业和商业运作等)在计算世界中的映射,它们不再单纯地服务于独立的个体或者表现为单一的技术系统,而是由诸多系统联盟组成[2],将不同的人、组织甚至是社会有机地联系在一起的一类社会技术系统(Socio Technical System,STS),如城市交通管理系统、健

* **收稿日期**:2013-09-11
基金项目:国家自然科学基金资助项目(61070034);教育部新世纪优秀人才计划资助项目(NCET-10-0898);北航软件开发环境国家重点实验室开放课题(SKLSDE-2012KF-0X)
作者简介:毛新军(1970—),男,博士,教授,博士生导师,E-mail: mao.xinjun@gmail.com

康护理系统、社会保障系统等。

近年来人们提出了诸多概念、思想和方法以促进这类软件系统的开发、管理和演化,如自治计算、网构软件、主动对象和构件、面向 Agent 软件工程(Agent-Oriented Software Engineering, AOSE)[3]、自适应和自组织系统的软件工程技术等。它们从不同的角度和层次来认识软件系统的复杂特征(如上下文敏感、自主和主动、自适应、自组织等),并通过与相关学科交叉(如生命科学、人工智能、控制学等),寻求有效的机制(如基于认知的自主决策机制等)、模型(如基于 BDI 的自主软件模型)和语言(如建模语言和程序设计语言等)等,从而来促进其构造和运行。

面向 Agent 的软件工程将软件系统视为由一组自主软件 Agent 以及它们之间的交互合作所构成的多 Agent 系统,有助于在一个更高的抽象层次来认识社会技术系统中的个体以及它们之间的交互,并采用一种更为自然的方式对社会技术系统进行建模和设计。本文旨在根据社会技术系统的社会性、动态性等特点及其对程序设计提出的要求,将社会组织的思想、概念和机制引入到面向 Agent 的程序设计范畴,建立起支持社会技术系统构造的程序设计模型和语言。

1 社会技术系统及其构造问题

社会技术系统是指由人、过程和技术要素等构成的系统[2]。其中,人既是系统的使用者和参与者也是构成系统的要素,它们在社会技术系统中的作用和行为取决于它们在系统中所处的地位和所扮演的角色;过程定义了技术系统的设计者要求人如何使用和参与技术系统。本文所讲的技术系统主要是指建立在计算和网络平台之上的各种软件密集型系统。社会技术系统通过技术手段(如软件系统)将社会系统中的人紧密地联系在一起,并支持它们之间的交互和合作;人使用和参与技术系统通常会受到社会系统中各种社会法规和组织制度等的限制,需要按照社会系统的要求和约束来行事。

部署在互联网之上的网上商场就是一个典型的社会技术系统,它为供应商、店主和店员、银行、顾客、送货商等提供了一个商品交易的虚拟平台,使得它们能够利用所部署的各种软件系统实现他们之间的交互与合作,如买进和卖出商品、拍卖商品、送货、交易等。它们是现实世界的商场在计算机和网络世界的映射,体现了虚拟的社会组织形态,任何处于其中的个体必须遵循社会系统的约束,如诚实交易等。技术系统表现为各种软件系统,它们为社会技术系统中的各类个体提供各种功能和服务,个体所能操作的功能和获得的服务取决于它们在虚拟社会中扮演的角色。例如,店主可以通过拍卖的方式出售其商品,自身不具备交易的能力,可以通过银行所提供的交易功能来实现交易等。不同于单一的技术系统(如部署在单机上的个人软件)以及单纯以交互为目的的社交系统(如 Facebook),社会技术系统通常具有以下特点:1) 社会要素和技术要素共同存在并且相互作用。系统中的技术要素不能独立于社会要素而存在,社会要素的变化反过来会引起技术要素发生变化。人不仅是系统的使用者,也是系统的组成成分;技术系统作为媒介来实现人与人之间的关联、交互和协同,从而使得他们构成结构化的组织。2) 社会技术系统通常驻留在多样和开放的环境中,并受到环境变化的影响。构成环境的成分不仅包括各种技术要素(如网络环境及其连通性、服务的可获得性等),而且还包括各种社会因素(如社会结构的组成、组成成员及其数量等等)。环境的变化具有事先不确定、难控或者不可控等方面的特点。3) 由于受所驻留环境影响以及系统建设需求的变化,社会技术系统通常动态变化,从而呈现出长期持续演化特点。

社会技术系统的上述特点决定了其中软件系统的构造和实现不能单纯地借助于传统意义上的技术观点,需要综合地从技术的角度来理解社会系统,从社会的角度来认识软件系统。这就要求我们提供一个统一的抽象和模型以及有效的机制和语言设施来支持其软件系统的设计、构造和运行。

2 基于组织的软件抽象及模型设计

2.1 基于 Agent 和组织的软件模型

软件工程的发展趋势之一就是寻求新颖、高层的抽象来管理和控制软件系统的复杂度。社会技术系统的特点需要我们以一种不同以往的方式看待这类系统,认识其组成、内在机制、运行规律和实现方式。近年来,人们尝试采用交叉社会组织学的思想来开展面向 Agent 软件工程的研究[6],借助于社会组织的抽象、模型和理论来认识复杂软件系统的组成并支持其开发,将多 Agent 系统视为是社会化组织[7],利用社会组织学的概念(如角色、组织、法规、职位等)来规约、描述、分析、设计和实现复杂软件系统[5]。基于组织的抽象和思想有助于独立于实现细节和具体平台在一个较高的抽象层次来描述和分析技术系统,并采

用统一的抽象来研究、设计和构造社会技术系统,它还可以作为实现概念和机制来支持社会技术系统的构造和运行,设计出符合社会技术系统特征的各种动态机制,进而开发出基于组织的程序设计语言及其支撑环境[4]。

本文将社会技术系统中的个体视为 Agent,利用社会组织的思想来认识社会技术系统的结构和行为,将整个社会技术系统视为多 Agent 组织。我们提出了如图 1 所示的基于组织的社会技术系统程序模型 ORAP。

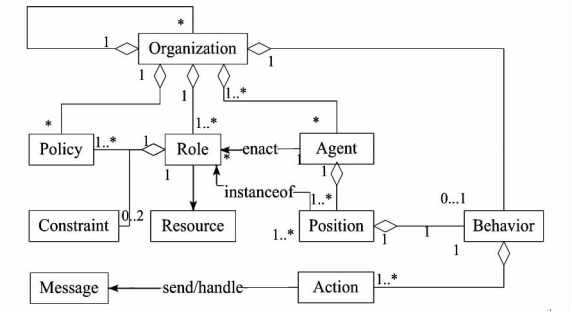

图 1　社会技术系统的软件模型

Fig. 1　Software model of socio technical systems

Agent: Agent 是多 Agent 组织中的基本运行软件实体,它封装了状态、动作和行为(包括交互),具有环境驻留性和行为自主性的基本能力。Agent 所封装的状态、动作和行为取决于它在组织中所扮演的角色。Agent 通过扮演组织中的角色而成为组织中的成员。一个 Agent 可以在不同组织中扮演多个角色,在其生命周期中所扮演角色可动态变化。

角色(Role): 角色对 Agent 在特定组织上下文中结构和行为的抽象,它定义了独立于 Agent 的交互模式和访问控制。角色封装了一组相关的资源、动作、行为、角色扮演约束条件(Constraint)以及交互策略(Policy)。其中,资源被抽象为角色所具有的属性,行为定义了一组动作序列,约束条件描述了扮演该角色需满足的基本条件,交互策略定义了系统如何选择角色扮演者来响应消息实现交互。

组织(Organization): 组织定义了在特定上下文中的 Agent 集合,封装了一组相关的角色、组织行为和策略。组织中的角色分为 2 类:内部角色和外部角色。内部角色封装了组织的功能和行为;外部角色定义了组织对外提供的服务接口。组织中的 Agent 也可以分为 2 类:本地 Agent 和外来 Agent。本地 Agent 由组织自身所创建,它们扮演了组织内部角色以实现组织功能;外来 Agent 由其他组织创建,但扮演该组织的角色(如外部角色等)来获得该组织的服务。

职位(Position): 职位是对组织中角色的实例化,它是角色在运行时的一种展现。在我们所提出的方法中一个组织可以将组织中的角色实例化为一个或者若干个职位。Agent 可以通过占有(Occupy)该职位从而动态地绑定相应的角色。它实际上是 Agent 扮演角色的一种实现机制。

2.2　组织动态性机制

多 Agent 组织的动态性表现为不同的形式,作用于不同的组织对象之上,论文设计了相应的机制来支持组织动态性的构造和实现。

1)角色绑定机制

社会技术系统中的 Agent 往往随环境的变化而动态地调整它所在的组织以及在组织中所扮演的角色。例如,网上商场中的 Agent a 在工作时扮演了商家的角色来提供服务,在业余时间扮演了客户的角色来购买网上商品。为此,我们提出了角色绑定机制。组织中的 Agent 可以通过 enact、deact、activate、deactivate 等原子动作动态地绑定一个角色、退出一个角色、激活一个角色或者挂起一个角色。绑定一个角色意味着 Agent 拥有了该角色定义的资源、动作和行为;退出一个角色意味着 Agent 失去了该角色定义的属性、动作和行为;挂起一个角色意味着角色所定义的行为不再发挥作用,但 Agent 仍然可以访问角色的结构信息;激活一个角色是指将挂起的角色恢复工作。角色扮演机制定义了 Agent 和角色之间动态和多重的关系,即一个 Agent 可以在运行时动态地扮演多个角色,同时一个角色可以被多个 Agent 扮演。它是实现 Agent 和组织动态性的基础。

2)角色组合机制

组合角色(composite role)定义了角色扮演者(即 Agent)在其生命周期中可以变迁的角色集以及角色变迁行为。构成组合角色的基本角色称为组件角色(component role)。角色变迁行为定义了组合角色的扮演者在运行时如何根据接收到的消息和自身的状态,在组件角色之间进行变迁的。组合角色中的变迁行为由一系列角色变迁规则组成,每条规则包括两个部分:基于消息和状态定义的角色变迁条件和四个基本角色变迁元操作:扮演(*enact*)、退出(*deact*)、激活(*activate*)和挂起(*deactivate*)。

3)基于角色的交互机制

在开放的多主体系统(Multi-Agent System, MAS)组织中,组织中的个体(即 Agent)可动态地发生变化,因而在设计时要明确参与交互的对象变得非常困难。另一方面,参与交互的 Agent 可

能只关心所需的服务而不关心服务的具体提供者。为此,我们设计了基于角色的交互机制,允许程序员在设计时基于角色来定义交互模式,而不考虑参与交互的具体 Agent,在运行时 Agent 通过扮演不同的角色来动态地发现所需服务的提供者和交互模式。在这种机制中,角色名可以被看作是 Agent 可提供的服务,由于角色的扮演者是不断变化的,通过角色来定义交互可有效地支持 Agent 之间交互的动态性,它包括以下 2 种交互方式:1) 一对一,Agent 只需知道与之交互的角色,而无须知道具体的 Agent 地址,运行平台在运行时动态地选择该角色的扮演者来响应其请求;2) 一对多,Agent 可以向某个角色的所有扮演者广播消息,而无须知道该角色当前具体的扮演者。

4)组织的自我管理机制

MAS 组织往往是动态变化的,但是在此过程中必须满足一定的约束、遵循一定的条件、符合相关的规范等,这就需要 MAS 组织具备自我管理的能力。例如,网上商场中一个网店的店主至多只有二个人,一个供应商至多只能给若干个商店供应商品等。组织的自管理是指组织在运行时可以根据其内部状态和外部环境对组织中的成员和构成进行动态调整。组织结构调整包含了三个基本的动作:创建一个新的 Agent、删除某个角色的所有扮演者和删除某个角色的一个扮演者,此外还允许组织动态地创建和解散其成员组织。

3 基于组织的程序设计语言

3.1 OragentL 语言的语法

为了支持社会技术系统的构造,我们设计了一个基于组织的程序设计语言 OragentL,其部分语法定义如图 2、图 3、图 4 所示。它将组织抽象作为一阶的编程对象,借鉴了 Actor 语言 Scala 中对消息的抽象方式和匹配机制,即采用样本(case)类对消息进行封装和匹配。一个完整的 OragentL 程序由若干组织和角色组成,其中任何角色需属于某个特定组织,组织为角色提供了命名空间,角色不能独立于组织而存在。角色当且仅当属于一个组织类,不允许同一角色在不同组织类间的重复出现。

```
1  <org_prog>       ::= {"employ" <org_cname> ";"}<org_cdef>
2  <org_cdef>       ::= "organization" <org_cname> "("[<form_param>] ")" "{"
3                         {<field_dec>} <disband_rule> {<role_ dec>}+ {<role_ def>}+
4                         [<init_act>] {<policy_def>} [<beh>] "}"
5  <field_dec>      ::= <type> <exp>";"
6  <disband_rule>   ::= "disbandOn" <bexp> ";"
7  <role _dec>      ::= <modifier> "role" <role _name> "("[<form_param> ] ")" ";"
8  <init_act>       ::= "initialize{" <statements> "}"
9  <modifier>       ::= ("internal" | "external") ["singular"]
10 <policy_def>     ::= "policy" <policy_name> "(" <role_names> ")" "{" <policy_body> "}"
11 <role_names>     ::= <role_name> {"," <role_name>}
12 <policy_body>    ::= [<rename_stm>] (<bexp> | <predication>)
13 <rename_stm>     ::= <role_name> "." <exp> {"," <role_name> "." <exp>} "->" <exp>
14 <predication>    ::= ("max" | "min") "(" <exp> ")"
15 <beh>            ::= ["loop"] "behavior" "{" <statements> "}"
16 <statements>     ::= {<statement>}+
17 <statement>      ::= ";" | <assign_stm> | <if_stm> | <while_stm> | <creation_stm> |
18                       <remove_stm> | <disband_stm> | <statements>
19 <creation_stm>   ::= "new" <org_name> "(" <exp> ")" ";" |
20                       "spawn" <role_ name> "(" <exp> ")" ";"
21 <remove_stm>     ::= "remove" <role_ name> "with" <policy_name> ";" |
22                       "remove" <role_ name> "*" ";"
23 <disband_stm>    ::= "disband" <org_id> ";"
24 <form_param>     ::= <type> <exp> {"," <type> <exp>}
25 <type>           ::= "int" | "boolean" | "ID"
26 <exp>            ::= <identifier> | <val>
27 <org_cname>      ::= <literal>
28 <role_name>      ::= <literal>
```

图 2 OragentL 组织程序的部分语法定义

Fig. 2 Part syntax definition of OragentL organization program

图 2 描述了 OragentL 语言组织编程的主要语法结构。组织程序由成员组织声明和组织定义两部分组成。成员组织是指一个组织在初始化或运行过程中,创建其成员组织依赖的组织。OragentL 必须显式地声明其成员组织。成员组织的声明由关键字“employ”和成员组织名组成。组织的定义由四部分组成:关键字“organization”、组织名、构造参数和组织体。组织名的命名采用 Java 中类的命名规则;构造参数定义了创建组织时需要提供的组织状态的初始值。组织类体的定义包括:组织域的声明、组织解散规则的定义、角色声明、初始化动作、角色和行为的定义。其中,组织域与组织的构造参数共同描述了组织的状态,OragentL 允许组织状态可以被其成员共享,即所有的组织成员都可以访问组织的状态。为了防止多个 Agent 同时对组织状态进行修改,从而发生冲突等问题,OragentL 要求组织状态只能被组织中的一个 Agent 修改。因此,组织应包含一个组织状态管理者角色 OrgManager,该角色最多只能有一个扮演者,负责对组织状态进行修改。

图 3 描述了 OragentL 语言角色编程的语法结构,主要包括角色所属组织的声明和角色定义两部分组成。在 OragentL 中,角色程序必须显式地声明其所属的组织,其声明语句由关键字“within”和组织类名组成。角色的定义由五个部分组成:角色修饰符、关键字“role”、角色名、构造参数和角色体。其中,角色修饰符与构造参数的含义与组织中角色声明的修饰符与构造参数一样,而且角色程序中角色的修饰符与构造参数的声明和定义,必须与其所属的组织中对应的角色声明的修饰符与构造参数完全一致。角色体的定义包括:角色域的声明、角色扮演约束条件的定义、角色动作、行为及交互策略的定义。角色域描述了其扮演者在其组织中的状态属性,即 Agent 成功扮演一个角色,则拥有了该角色的域定义的状态。

图 4 描述了 OragentL 角色组合的语法结构。OragentL 使用关键字“uses”显式地声明角色的组合关系,其组件角色的声明由关键字“uses”、组件角色名和对组件角色构造参数初始化的实参组成。在 OragentL 中,一个角色可以拥有多个组件角色,不同组件角色声明之间以逗号“,”分开。组合角色的扮演者 Agent 在其组合角色与组件角色的变迁过程中可能需要进行数据传递以实现协同和通信。

```
1  <role_prog>     ::= "within" <org_cname> ";" <role _def>
2  <role _def>     ::= <modifier> "role" <role _name> "("[<form_param>] ")" "{"
3                      {<field_dec>} [<constraint>] {<policy_def>} <beh>{<act>}+ "}"
4  <modifier>      ::= ("internal" | "external") ["singular"]
5  <constraint>    ::= "constraint" <bexp> ";"
6  <beh>           ::= ["loop"] "behavior" "{" <statements> "}"
7  <act>           ::= <act_name> "(" [<form_param>] ")" "{" <statements> "}"
8  <statements>    ::= {<statement>}+
9  <statement>     ::= ";" | <assign_stm> | <if_stm> | <while_stm> | <act_ivk_stm> |
10                     <creation_stm> | <send_stm> | <receive_stm>
11 <act_ivk_stm>   ::= <act_name> "(" <exp> ")" ";"
12 <send_stm>      ::= <agent_id> "!" <msg> ";" |
13                     <role_ name>"*" "!" <msg> ";" |
14                     <role_ name> "!" <msg> "with" <policy_name> ";"
15 <msg>           ::= <msg_name> "(" <exp> {"," <exp>} ")"
16 <receive_stm>   ::= "receive {" {<msg_match>}+ "}" ["after(" <exp> "){" <statements> "}"]
17 <msg_match>     ::= "case" <msg_temp> ":" <statements>
18 <msg_temp>      ::= [<role_name>] "?" <msg_name> "(" <form_param> ")"
19 <form_param>    ::= <type> <exp> {"," <type> <exp>}
20 <type>          ::= "int" | "boolean" | "ID"
21 <exp>           ::= <identifier> | <val> | "_"
22 <msg _name>     ::= <literal>
```

图 3　OragentL 角色程序的部分语法定义

Fig. 3　Part syntax definition of OragentL role program

```
1  <role_def>              ::= <modifier> "role" <role_temp_name> "("[<form_param>] ")"
2                              ["extends" <role_temp_dec>] "promotes" <lower_role_dec>
3                              <promote_cond> {"," <lower_role_dec><promote_cond>}
4                              ["uses" <component_role_decs>] "{" <rename_stm>
5                              {<field_dec>} [<constraint>] {<policy_def>} <beh> {<act_dec>}+
6                              {<act>}+ "}"
7  <component_role_decs>   ::= <component_role_dec> {"," <component_role_dec>}
8  <component_role_dec>    ::= <role_name> "(" <exp> {"," <exp>} ")"
9  <rename_stm>            ::= ... | <role_name> "." <exp> {"," <role_name> "." <exp>} "->"
10                             <type> <exp> ";"
11            ……
12 <statement>             ::=  ... | <role_trans_stm> | <exception_stm>
13 <role_trans_stm>         ::= ("enact" | "deact" | "activate") <exp>
14 <exception_stm>          ::= "throw" <msg> |
15                             "try" {" <statements> "}catch(" <msg_temp> "){" <statements> "}"
16 <msg>                   ::= <msg_name> "(" <exp> {"," <exp>} ")"
17 <msg_temp>              ::= ... | <msg_name> "(" <form_param> ")"
18 <form_param>            ::= <type> <exp> {"," <type> <exp>}
19 <exp>                   ::= ... | <org_id> "." <role_name>
```

图 4 OragentL 角色组合程序的部分语法定义

Fig. 4 Part syntax definition of OragentL role composition program

3.2 OragentL 语言的编译和执行

支持 OragentL 程序开发、部署和运行的支撑环境 OragentBurg 的总体架构如图 5 所示，共分为三个层次：开发支持、编译支持和运行支持。

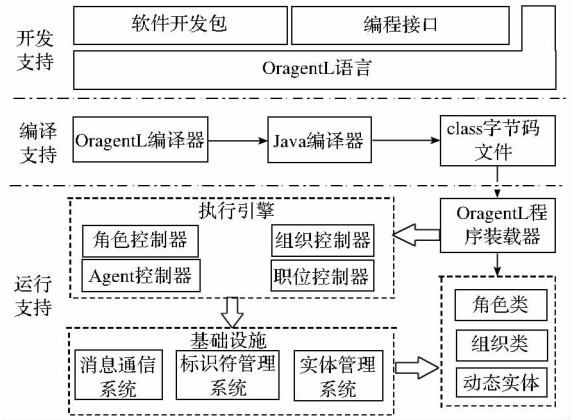

图 5 OragentBurg 支撑环境总体架构

Fig. 5 Framework of OragentBurg environment

开发支持：主要为 OragentL 程序的开发提供支持。OragentL 语言提供了基本的语言设施，OragentL 编程接口帮助开发者操纵或处理 OragentL 语言的特殊机制，方便对系统功能的调用，简化 OragentL 程序的编写。编程接口的设计基于 OragentL 语言的规范，以系统类库的形式提供给开发者使用。

编译支持：主要将 OragentL 程序转换为运行的 Java 代码，然后再利用 Java 编译器将 Java 代码进一步转换为字节码，在运行时由 OragentL 程序加载器将相应程序代码加载到内存中并运行。

运行支持：负责在运行时为 OragentL 程序运行实体提供生命周期管理、标识符管理和消息通信等基本服务，同时为 OragentL 程序的核心运行实体提供多样化的执行引擎，分别负责不同运行实体的运行。同时，它还提供了 OragentL 程序加载器以实现 OragentL 程序的动态加载。

4 社会技术系统的案例分析和实现

本节针对第二节所介绍的网上商场案例开展分析和实现。由于篇幅考虑，本文仅仅分析网上商场中的“支付”和“拍卖”组织以及这些组织的自我管理。

支付组织“PaymentOrg”程序声明了三个角色：代理 Broker、付款人 Payer 和收款人 Payee（见图 6），其中 Broker 是内部角色，Payer 和 Payee 是外部角色（见图 6 中第 2 ~ 4 行）。角色 Broker 拥有一个布尔类型的构造参数 free，表示其扮演者当前是否处于空闲状态。角色 Payer 拥有一个整数型的构造参数 money，指明扮演者需要支付的钱数。支付组织的初始化部分将创建一个代理 Agent，并将其状态设置为空闲。角色 Broker 定义了策略 RegulateBrokerPolicy，用来查找当前空闲的扮演者。图 6 中第 11 ~ 16 行定义了支付组织的自管理行为，即根据当前付款人数和代理数之间的比例来动态地调整代理 Agent 的数量。在 OragentL 中，“# RoleName”用来获取角色 RoleName 当前的扮演者（即 Agent）数目。因此支

```
1  organization PaymentOrg(){
2      internal role Broker(boolean free);
3      external role Payer(int money);
4      external role Payee();
5      initialize{
6          spawn Broker(true);
7      }
8      policy RegulateBrokerPolicy(Broker){
9          free==false;
10     }
11     loop behavior{
12         if(#Payer/#Broker > 5)
13             spawn Broker(true);
14         if(#Broker>#Payer && #Broker>1)
15             remove Broker with RegulateBrokerPolicy;
16     }
17 }
18 organization AuctionOrg() {
19     abstract external role Bidder(int top);
20     abstract internal singular role Auctioneer();
21     external singular role Provider();
22     disband On    #Provider==0;
23     initialize {spawn Auctioneer();}
24 }
```

图 6　在线支付组织和拍卖组织的 OragentL 代码片断

Fig. 6　Part of OragentL program codes of on-line payment and auction organization

付组织“PaymentOrg”的自管理行为可以描述为:如果付款人数与代理 Agent 数比例大于 5,即每个代理 Agent 都有 5 个付款人在等待或正在转账,组织则增加一个新的代理 Agent,并将其属性 free 初始化为 true;如果当前代理 Agent 数大于付款人数,且组织中多于一个代理 Agent,则移除空闲的 Agent,基于策略 RegulateBrokerPolicy 实现。

拍卖组织“AuctionOrg”声明 3 个角色:拍卖商 Auctioneer、竞价者 Bidder 和提供商 Provider,其中 Bidder 和 Provider 是外部角色,而 Auctioneer 是内部角色。拍卖组织是一个暂时性组织,它由提供商因拍卖商品而动态地创建,并随着商品拍卖的完成而自动解散。因此图 6 中第 22 行定义了拍卖组织的解散条件:“#Provider = =0”表示如果提供商退出了拍卖组织,则该组织自动解散。包含上述代码片段的完整程序经过 OragentL 编译器的编译后将生成相应的 Java 代码,进而在 OragentBurg 环境下运行。

5　相关工作和结论

将组织抽象和思想用于抽象、解释、分析和设计多 Agent 系统是近年来面向 Agent 软件工程的一项重要研究内容,如规约组织的适应性变化[12]等。Ferber 提出的多 Agent 系统的 ARG (Agent - Role - Group)模型[7]对基于组织建模语言和开发方法学的研究产生了重要影响。随着组织抽象思想在面向 Agent 分析和设计中的成功应用,将组织抽象思想引入面向 Agent 的程序设计(Agent Oriented Programming, AOP)成为 AOSE 领域新的焦点。近年来 AOP 的研究更加关注多 Agent 系统层面的内容及其实现技术,应用范畴和研究背景转向更为广泛的、与主流计算技术相结合的应用领域,如网络环境下的软件密集型系统、自适应和自组织系统、面向服务计算的系统、并发系统等等[13,15]。目前支持基于组织软件构造的方法主要有两种:中间件方法和程序设计语言方法。通过组织中间件使得基于传统 Agent 语言开发的 Agent 具有组织感知能力(organization - awareness),使得 Agent 能够根据组织状态、规则等信息来调整自身的行为并修改组织的状态(如 JaCaMo[9])。另外,组织中间件还可以作为一种外部机制来对 Agent 的行为进行管理和约束,如 MACODO[11]。Janus 和 powerJade 分别是基于 Java 和 Jade 开发的支持组织、角色等组织概念的运行平台,它为开发人员提供一系列开发包和运行时支持[10]。由于基于中间件的构造方法借助于已有的程序设计语言,在组织层和 Agent 层缺乏统一的概念抽象,并通过软件开发包的形式来支持组织实现,这会带来二方面的问题:一是导致代码混杂、程序逻辑不清晰,如在 JaCaMo 中,在 Agent 内部定义组织的动态性逻辑,使得组织动态性与 Agent 的内部规划混合在一起;二是受现有程序设计语言的限制,无法充分发挥组织抽象的优势,如 powerJade 在构造复杂 MAS 时可能引起 Agent(在 powerJade 中,角色也是一类特殊的 Agent)数量迅速膨胀和 Agent 交互的急剧增加。

基于程序设计语言的组织构造方法将组织作为显式的程序设计设施,通过设计专门的基于组织程序设计语言,从而来支持多 Agent 系统的构造。一些工作通过对 AOP 语言的 Agent 结构进行扩展来描述 Group,并通过增加基本操作元语来实现 Group 层次的动态变化但它们往往缺乏对组织概念的显式定义和支持。在面向 Agent 程序设计语言 3APL 中[8], Dastani 引入了“enact”,

“deact”，“activate”和“deactivate”来刻画 Agent 扮演角色的动态性，但3APL 中的 Agent“enact”一个角色是指内部实例化一个角色规约，而“activate”一个角色是指 Agent 基于角色的规约进行推理。最近该方面的工作试图将组织层面的规范(norm)或者组织法规(organization rule)作为程序设计设施[14]，代表性工作包括 2OPL (Organization Oriented Programming)。2OPL 支持基于规范的组织编程，侧重于研究组织规范对 Agent 行为的管理和约束，对 Agent 组织的动态性支持不足。

本文针对社会技术系统的特点及其构造对程序设计提出的一系列要求，开展了基于组织的社会技术系统程序设计技术的研究，取得了以下三方面的创新成果。首先，将社会技术系统视为多 Agent 组织，把社会组织学的概念和思想引入到面向 Agent 程序设计范畴，提出了基于组织抽象的社会技术系统软件模型，从而基于统一的抽象来分析社会技术系统中的技术因素和社会因素，并进一步支持技术系统的构造和实现。其次，通过对社会系统的分析和抽象，提出了一组基于组织的程序设计机制，以支持对软件系统不同层次上的动态性进行描述和实现，包括：角色绑定、角色晋级、角色组合、基于角色的交互、组织自我管理等等。第三，基于程序模型和核心机制，设计了基于组织的程序设计语言 OragentL，该语言具有基于组织抽象的构造、支持多层次动态性实现等方面的特点。我们已经开发了 OragentL 语言的编译器和运行支撑环境，并成功开展了若干应用的开发。

参考文献(References)

[1] Feiler P H, Sullivan K, Wallnau K C, et al. Ultra-large-scale systems: the software challenge of the future[R]. Software Engineering Institute, Carnegie Mellon University, 2006.

[2] Sommerville I, Cliff D, Calinescu R, et al. Large-scale complex IT system[J]. Communication of ACM, 2012, 55(7): 71 - 77.

[3] Lucena C, Nunes I. Contributions to the emergence and consolidation of agent-oriented software engineering[J]. Journal of Systems and Software, 2012, 86(4): 890 - 904.

[4] 毛新军，胡翠云，孙跃坤，等. 面向 Agent 程序设计的研究[J]. 软件学报，2012，23(11)：2885 - 2904.
MAO Xinjun, HU Cuiyun, SUN Yuekun, et al. Research on agent-oriented programming[J]. Journal of Software, 2012, 23(11): 2885 - 2904. (in Chinese)

[5] Wester-Ebbinghaus M, Moldt D, Reese C, et al. Towards organization oriented software engineering[C]//Proceeding of Software Engineering Konferenz ,2007: 205 - 217.

[6] Mao X J, Yu E. Organizational and social concepts in agent oriented software engineering [C]//Proceeding of the 4th International Workshop on Agent Oriented Software Engineering, 2004:1 - 15.

[7] Ferber J, Gutknecht O, Michel F. From agents to organizations: an organizational view of multi-agent systems[C]//Proceeding of the 4th International Conference on Agent-Oriented Software Engineering , 2004:214 - 230.

[8] Dastani M D, van Riemsdijk M B, Hulstijn J, et al. Enacting and deacting roles in agent programming[C]//Proceeding of the 5th International Workshop on Agent-Oriented Software Engineering, 2005:189 - 204.

[9] Boissier O, Bordini R H, Hübner J F, et al. Multi-agent oriented programming with JaCaMo[J]. Science of Computer Programming, 2013, 78(6): 747 - 761.

[10] Baldoni M, Boella G, Genovese V, et al. How to program organizations and roles in the JADE framework[C]// Proceeding of the 6th German conference on Multiagent System Technologies, 2008:25 - 36.

[11] Weyns D, Heasevoets R, Helleboogh A, et al. The MACODO middleware architecture for context-driven dynamic agent organizations[J]. ACM Transactions on Autonomous and Adaptive Systems, 2010,5(1): 1 - 25.

[12] Carr H, Artikis A, Pitt J. Software support for organised adaptation [C]//Proceedings of the 8th International Workshop on Programming Multi-Agent Systems, 2012: 96 - 115.

[13] Bordini R H, Dastani M, Dix J, et al. Programming multi-agent systems [C]//Proceedings of the 12th International Workshop on Agent-Oriented Software Engineering, 2012:23 - 52.

[14] Tinnemeier N, Dastani M, Meyer J J. Roles and norms for programming agent organizations[C]//Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, 2009:121 - 128.

[15] Muscar A. Agents for the 21st century: the blueprint agent programming language [C]//Proceedings of The 1st International Workshop on Engineering Multi-Agent Systems, 2013: 255 - 269.