一种服务质量驱动的企业应用软件构件组装方法

孟凡超[1] 初佃辉[1] 战德臣[1,2] 徐晓飞[1,2]
(哈尔滨工业大学(威海)计算机科学与技术学院 威海 264209)[1]
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)[2]

摘 要 针对现有面向全局构件组装方案选择技术的不足,提出了一种基于服务质量优化的构件组装方案选择方法。该方法主要是面向大型复杂企业应用软件系统的配置管理,其主要思想是将构件组装方案的选择问题转化为带约束的多目标优化问题。针对该问题,给出了一种基于向量编码的构件组装方案选择遗传算法,该编码方式可以非常方便地表示构件组装模型中构件接口之间的连接关系,从而克服了现有编码在描述构件组装模型中的局限性。最后通过实验分析了算法的可行性。

关键词 构件组装,服务质量,遗传算法,向量编码
中图法分类号 TP311.5 **文献标识码** A

Approach to QoS Driven Component Composition of Enterprise Software and Application

MENG Fan-chao[1] CHU Dian-hui[1] ZHAN De-chen[1,2] XU Xiao-fei[1,2]
(School of Computer Science and Technology, Harbin Institute of Technology at Weihai, Weihai 264209, China)[1]
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)[2]

Abstract Aiming at the problem of current component assembly projects selection technology, an approach to component composition selection based on QoS optimization which is mainly oriented large and complex enterprise software and application system configuration management was proposed. The essence of this approach is that the problem of component composition projects selection is transformed into a problem of multi-objective optimization with constraints. A genetic algorithm based on vector encoding scheme was presented for component composition projects selection. The vector encoding scheme can easily express the connection relationships between interfaces in the component composition model, which can't be expressed in other encoding schemes. Finally, experiments results show the feasibility and efficiency.

Keywords Component composition, Quality of service(Qos), Genetic algorithm, Vector encoding

构件的组装是基于构件的软件开发的核心技术,构件必须经过组装才能形成应用系统,实现其核心价值。基于构件的软件框架和实现构件模型为构件的组装提供了技术基础。随着构件库中可复用构件数量的不断增多,我们经常面对更高层次的问题,即选择哪些构件,按照什么样的结构连接这些构件才能完成用户对系统的功能需求,还有在一组功能等价而服务质量不同的构件中如何选择一个最符合用户要求的构件进行组装系统来提高系统的服务质量,从而满足用户对所使用系统的个性化需求。构件的服务质量(QoS)是指构件的一组非功能属性集合,例如可靠性、可用性和执行代价等。随着构件技术的推广,服务质量驱动的构件组装将会成为基于构件的软件开发的关键需求[1]。本文主要研究服务质量驱动的企业应用软件的构件组装方案选择问题。

在服务质量驱动的企业应用软件系统的构件组装中,构件组装方案的选择可以分为局部策略和全局策略。局部策略是以系统中的单个功能为粒度来选择构件,即分别考察各个功能的候选构件集,从一组功能等价的候选构件集中选择适当的或最好的构件作为该功能的实现构件[2]。全局策略则是以子系统或者整个系统为粒度来选择构件,其目标是使系统的整体服务质量在满足给定的约束条件下达到最优,因此构件组装方案的选择需要综合考虑各个构件的组装效果,即选择适当或最好的组装方案[3]。由于全局策略对于提高软件系统的质量更具有价值,因此本文主要关注于全局选择问题。

在基于构件组装的企业应用软件系统中,构件之间是通过请求接口和提供接口的连接进行组装的,因此在构件组装时,必须考虑构件接口之间的连接所产生依赖关系。虽然目

到稿日期:2009-01-16 返修日期:2009-03-05 本文受国家自然科学基金项目(60673025),国家863计划资助项目(2006AA04Z150,2008AA04Z101),山东省自然科学基金项目(2007ZRA10003)资助。

孟凡超(1974—),男,博士,讲师,主要研究领域为模型驱动的体系结构、软件重构与复用;初佃辉(1970—),男,副教授,主要研究领域为现代企业资源计划、电子商务、供应链管理、物流管理等;战德臣(1965—),男,博士,教授,博士生导师,CCF高级会员,主要研究领域为数据库与知识工程、软件重构与复用、虚拟企业集成与电子商务、企业资源计划系统;徐晓飞(1962—),男,博士,教授,博士生导师,CCF高级会员,主要研究领域为企业智能计算、管理与决策信息系统、数据库、企业资源计划与供应链管理技术、电子商务与商务智能。

前许多方法也考虑了构件之间的依赖关系[4-6]，但是这些方法也仅仅是将构件看作一个只具有单个提供和请求接口的实体，并没有考虑具有多个提供和请求接口以及构件与系统中功能之间的绑定关系，从而限定了其适用范围。针对上述问题，本文提出了一种服务质量驱动的企业应用软件构件组装方法。首先，提出了一个面向配置构件组装模型，基于单个构件 QoS 的分析，给出了构件组装方案的 QoS 的定义，并将服务质量驱动的构件组装问题转化为一个多目标优化问题。针对该优化问题，提出了一个基于向量编码的构件组装方案选择遗传算法，并通过实验验证了该算法的可行性和有效性。

1 服务质量驱动的构件组装过程

目前，大型复杂企业软件系统（例如 ERP/SCM）一般都采用基于构件的软件框架来实现。本文以适合中国国情的可重构 ERP 系统软件框架（简称 CERP 框架）为载体来研究服务质量驱动的构件组装方案选择问题[7]。CERP 框架是由构件库、配置管理器、可执行应用系统和框架公共服务等主要部分构成。构件库中存储了应用领域中的所有构件规约和代码，这些构件来自不同的项目组，一般是由不同的开发者所开发，它们可以在同一领域的多个应用系统中复用。构件分为两类：过程构件和实体构件。过程构件是对企业中业务活动的实现，实体构件则是对企业中业务对象的实现，过程构件的执行一般需要相应实体构件所提供的服务。配置管理器是对每个系统所包含的功能（每个功能对应企业中的一项业务活动）、执行每个功能所需要的过程构件以及构件之间（过程构件与实体构件、实体构件与实体构件）连接关系的定义。通过配置管理器的配置，可以生成同一领域的多个可执行应用系统。框架公共服务提供一些公共服务，例如，事务和安全性等。

如图 1 所示，服务质量驱动的构件组装可以分为两个主要过程选择候选构件和选择最佳组装方案。选择候选构件是指根据需要组装的应用系统的每个底层功能的需求，从构件库中选择满足其需求的一个或多个候选构件以及这些构件所依赖的构件。选择最佳组装方案是指根据用户对应用系统的服务质量需求，对应用系统每个底层功能所需的构件以及构件之间的连接关系进行选择，从而确定一个满足用户需求的最优或较优的构件组装方案（构件及其之间连接关系）。

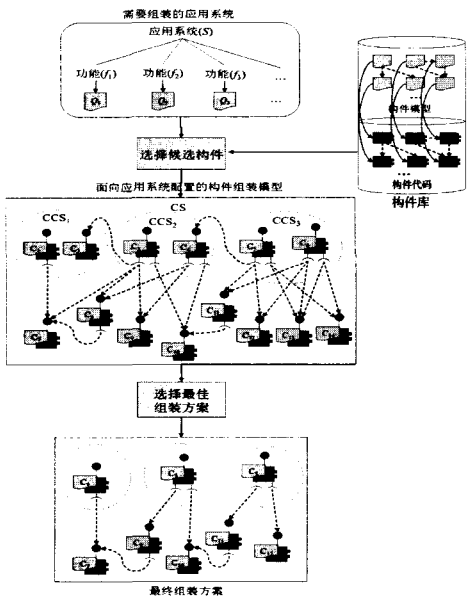

图 1 服务质量驱动的构件组装过程

2 面向应用系统配置的构件组装模型

本节给出面向应用系统配置的构件组装模型中一些基本概念的形式化定义。

定义 1 一个面向组装的构件可以定义为五元组 $C=(n, PI, RI, P, B)$，其中，n 为构件的名称，PI 为构件的提供接口集，RI 为构件的请求接口集，每个接口是由一组服务构成的，这里的服务既可以是被调用的方法，也可以是异步消息，P 为构件的行为协议，B 为构件的实现体。每个构件可以不包含请求接口，但是至少要包含一个提供接口。

定义 2 设 CL 为构件库，$C_1=(n_1, PI_1, RI_1, P_1, B_1)$ 和 $C_2=(n_2, PI_2, RI_2, P_2, B_2)$ 为 CL 中的两个构件，$RI_{1i}\in RI_1$ 为 C_1 的一个请求接口，$PI_{2j}\in PI_2$ 为 C_2 的一个提供接口，如果 PI_{2j} 所包含的服务在语义上能够完全满足 RI_{1i} 的服务请求，并且满足行为协议的兼容性，则 RI_{1i} 到 PI_{2j} 是可连接的，记为 $RI_{1i}\rightarrow PI_{2j}$。

定义 3 设 $C_1=(n_1, PI_1, RI_1, P_1, B_1)$ 为构件库 CL 中任意一个具有请求接口的构件，如果对于任意 $RI_{1i}\in RI_1$，在 CL 中都至少存在一个构件 $C_2=(n_2, PI_2, RI_2, P_2, B_2)$，$PI_{2j}\in PI_2$ 为 C_2 的一个提供接口，满足条件：$RI_{1i}\rightarrow PI_{2j}$，则称 CL 为一个完备的构件库。本文假定构件库是完备的。

定义 4 设 S 为一个应用系统，则面向 S 配置的构件组装模型可以定义为 $CAM(S)=(F, CS, LS, CCS)$，其中，$F=\{f_1, f_2, \cdots, f_n\}$ 为 S 所包含的底层功能的集合，CS 为所有候选构件的集合，$LS=\{(C_1.RI_{1i}, C_2.PI_{2j}) \mid C_1, C_2\in CS, RI_{1i}\in RI_1, PI_{2j}\in PI_2, RI_{1i}\rightarrow PI_{2j}\}$ 为 CS 中构件之间的所有可能的连接关系集，$CCS=\{CCS_1, CCS_2, \cdots, CCS_n\}$ 为 F 中每个功能的候选绑定构件集的集合，$CCS_i\subseteq CS$ 为 f_i 的候选绑定构件集，在系统执行时刻，每个 f_i 都要绑定到 CCS_i 中的一个构件上。可以用 $F(S)$、$CS(S)$ 和 $LS(S)$ 分别表示 S 的功能集，候选过程构件和连接集。

用于组装应用系统 S 的构件可以分为两类：绑定构件和间接依赖构件。我们将 S 中的每个功能的候选绑定构件集中的构件称为绑定构件，绑定构件一般为过程构件。设 $CS_A(S)=\bigcup_{i=1}^{n} CCS_i$ 为 S 的绑定构件集。$CS_A(S)$ 中的构件所直接或者间接依赖的，但是并不包含在 $CS_A(S)$ 中的构件称为间接依赖构件，间接依赖构件一般为实体构件。设 $CS_B(S)$ 为 S 的间接依赖构件集，则 $CS(S)=CS_A(S)\cup CS_B(S)$。构件之间通过请求接口与提供接口之间的连接进行组装。对于一个构件的请求接口，可能存在多个可连接的构件的提供接口满足其服务需求。我们用 $Dep(C_1.RI_{1i})=\{C_2 \mid (C_1.RI_{1i}, C_2.PI_{2j})\in LS(S)\}$ 表示构件 C_1 的请求接口 RI_{1i} 所依赖的构件集。

定义 5 二元组 $\langle f_i, C_{ij}\rangle (C_{ij}\in CCS_i)$ 称为功能 f_i 的一个绑定，$D(f_i)=\{\langle f, C_{ij}\rangle \mid C_{ij}\in CCS_i\}$ 表示功能 f_i 的候选绑定集。n 元组 $d=(\langle f_1, C_{1j_1}\rangle, \langle f_2, C_{2j_2}\rangle, \cdots, \langle f_n, C_{nj_n}\rangle)$ 称为 S 的一个绑定，其中，$\langle f_i, C_{ij_i}\rangle\in D(f_i)(i=1,2,\cdots,n)$。$D(S)=\{(\langle f_1, C_{1j_1}\rangle, \langle f_2, C_{2j_2}\rangle, \cdots, \langle f_n, C_{nj_n}\rangle) \mid f_i\in F, (f_i, S_{ij_i})\in D(f_i)\}$ 表示应用系统 S 的候选绑定集。

定义 6 设 $C\in CS(S)$ 为一个用于组装应用系统 S 的候选构件，$RI=\{RI_1, RI_2, \cdots, RI_m\}$ 为 C 所包含的请求接口集，则 m 元组 $\langle C_1, C_2, \cdots, C_m\rangle$ 称为 C 的一个接口配置，在这里，C_i

$(Dep(C.I_i)(1\leqslant i\leqslant m)$为 C 的请求接口 RI_i 的一个依赖构件，我们用 $Config(C)=\{\langle C_1,C_2,\cdots,C_m\rangle|C_i\in Dep(C.I_i)\}$ 表示 C 的请求接口配置集。

例如，在图 1 中，$D(f_1)=\{\langle f_1,C_1\rangle,\langle f_1,C_2\rangle\}$，$D(f_{21})=\{\langle f_{21},C_3\rangle,\langle f_{21},C_4\rangle\}$，$D(f_{22})=\{\langle f_{22},C_5\rangle,\langle f_{22},C_6\rangle\}$。根据接口配置的定义，对于构件 C_3，则有 $Config(C_3)=\{\langle C_2,C_{10}\rangle,\langle C_7,C_{10}\rangle,\langle C_8,C_{10}\rangle,\langle C_9,C_{10}\rangle\}$。

如果我们为应用系统 S 选择一个绑定 $d=(\langle f_1,C_{1j_1}\rangle,\langle f_2,C_{2j_2}\rangle,\cdots,\langle f_n,C_{nj_n}\rangle)\in D(S)$，并且对 d 中的每个构件以及这些构件所直接或间接依赖的构件指定一个相应的接口配置，则会产生一个组装方案。

定义 7 应用系统 S 的一个组装方案可以定义为 $x=(d_x,CS_x,LS_x)$，其中，d_x 为对 S 的一个绑定，$CS_x\subseteq CS(S)$ 为组装方案所包含的构件集，$LS_x\subseteq LS(S)$ 为组装方案所包含的连接集。我们用 $PS(S)$ 表示 S 的所有组装方案的集合。

例如，在图 1 中，如果我们为 S 选择绑定 $d_1=\{\langle f_1,C_1\rangle,\langle f_{21},C_3\rangle,\langle f_{22},C_5\rangle\}$，则 d_1 所包含的构件集为 $\{C_1,C_3,C_5\}$。如果对于 C_1，我们为其指定配置 $\langle C_7\rangle$，对于 C_3，我们为其指定配置 $\langle C_8,C_{10}\rangle$，并且对 C_8 指定配置 $\langle C_7\rangle$，对于 C_5，我们为其指定配置 $\langle C_{11},C_{15}\rangle$，并且对 C_{11} 指定配置 $\langle C_{10}\rangle$，对于 C_{15} 指定配置 $\langle C_{13}\rangle$，则可以得到如图 1 所示的一个组装方案。

3 构件组装方案的服务质量

影响构件服务质量的因素有很多，本文主要考虑 5 种 QoS 属性：复杂度、复用度、可靠性、可用性和执行代价。

- 复杂度：表示构件实现其服务的复杂程度，可以根据构件提供接口中所包含的操作的复杂度来度量。
- 复用度：是指构件被复用的状况，可以通过构件被具体应用系统选择使用的频率来计算它的复用度。
- 可靠性：可靠性表示维持构件服务质量的程度。每天、每月或每年的失效次数是衡量构件所提供服务可靠性的尺度[8]。
- 可用性：是指构件是否已经就绪可供立即使用。可以采用平均修复时间和平均故障间隔时间来计算可用性[8]。
- 执行代价：是指构件运行所需资源的数量，CPU 和内存是构件运行所需的两类重要资源。由于不同的执行环境，构件的执行所需的资源数量是不同的，因此，在度量执行代价时应该以某一个特定的测试环境为基准。

定义 8 设 $C\in CS(S)$ 为一个用于组装应用系统 S 的候选构件，则 C 的服务质量可以定义为：$Q(C)=(Q_{Comp}(C),Q_{Reus}(C),Q_{Relia}(C),Q_{Avai}(C),Q_{Cost}(C))$，其中，$Q_{Comp}(C)$，$Q_{Reus}(C)$，$Q_{Relia}(C)$，$Q_{Avai}(C)$ 和 $Q_{Cost}(C)$ 分别为构件 C 的复杂度、复用度、可靠性、可用性和执行代价。

定义 9 设 $x=(d_x,CS_x,LS_x)\in AS(S)$ 为应用系统 S 的一个组装方案，则 x 的服务质量可以定义为：$Q(x)=(Q_{Comp}(x),Q_{Reus}(x),Q_{Relia}(x),Q_{Avai}(x),Q_{Cost}(x))$，其中，$Q_{Comp}(x)$，$Q_{Reus}(x)$，$Q_{Relia}(x)$，$Q_{Avai}(x)$ 和 $Q_{Cost}(x)$ 分别为组装方案 x 的复杂度、复用度、可靠性、可用性和执行代价。x 的复杂度、复用度和执行代价可以分别由 CS_x 中的每个构件的复杂度、复用度和执行代价的总和来度量，可靠性和可用性可以分别由 CS_x 中的每个构件的可靠性的乘积和可用性的乘积来度量[9]，即：

$$Q_{Comp}(x)=\sum_{C\in CS_x}Q_{Comp}(C),\ Q_{Reus}(x)=\sum_{C\in CS_x}Q_{Reus}(C)$$
$$Q_{Relia}(x)=\prod_{C\in CS_x}Q_{Relia}(C),\ Q_{Avai}(x)=\prod_{C\in CS_x}Q_{Avai}(C)$$
$$Q_{Cost}(x)=\sum_{C\in CS_x}Q_{Cost}(C)$$

4 构件组装方案选择问题描述

对于一个可配置应用系统，可能存在多种构件组装方案，并且这些组装方案在功能上都能满足系统的功能需求。服务质量驱动的构件组装是指从这些组装方案中选择一个组装方案，使得该组装方案是所有组装方案中最优的一个，即组装方案的复杂度和执行代价最低，复用度、可靠性和可用性最高。由于各种质量因素之间可能存在相互制约关系，因此很难使得所有质量因素都达到最优，所以基于服务质量的构件组装是一个多目标优化问题，该问题可以描述为：

$$\text{Min } F(x)=(Q_{Comp}(x),Q_{Reus}(x),-Q_{Relia}(x),-Q_{Avai}(x),Q_{Cost}(x)) \quad (1)$$

$$\text{s.t. } Q_{Comp}(x)\leqslant p_0 \quad (2)$$
$$-Q_{Reus}(x)\leqslant -s_0 \quad (3)$$
$$-Q_{Relia}(x)\leqslant -r_0 \quad (4)$$
$$-Q_{Avai}(x)\leqslant -a_0 \quad (5)$$
$$Q_{Cost}(x)\leqslant c_0 \quad (6)$$

其中，$x\in PS(S)$ 为应用系统 S 的一个组装方案，p_0 代表对复杂度的最高要求，s_0 代表对复用度的最低要求，r_0 代表对可靠性的最低要求，a_0 代表对可靠性的最低要求，c_0 代表对执行代价的最高要求。

多目标优化的特点在于各个目标之间的相互冲突，所以基于服务质量优化的业务构件组装的结果不是单一解，而是在多个质量因素之间取折衷，得到一个满足约束条件的 Pareto 优化解集。设 $AS(S)=\{x|Q_{Comp}(x)\leqslant p_0,-Q_{Reus}(x)\leqslant -s_0,-Q_{Relia}(x)\leqslant -r_0,-Q_{Avai}(x)\leqslant -a_0,Q_{Cost}(x)\leqslant c_0\}\subseteq PS(S)$ 为满足约束条件(2)－(6)的可行组装方案集，$x^*\in AS(S)$ 为一个可行组装方案，如果在 $AS(S)$ 中找不到一个组装方案 $x(x\neq x^*)$ 使得 $F(x)\leqslant F(x^*)$，则称 x^* 为非劣方案或 Pareto 最优方案。一般来说，非劣方案不止一个，非劣方案的集合称为非劣方案集，用 $AS^*(S)$ 来表示。用户可以根据特定的需要从 $AS^*(S)$ 中选择满意的解。

5 基于向量编码的构件组装方案搜索算法

基于服务质量优化的构件组装方案选择可以分为两个过程，首先求出所有的非劣组装方案，然后从这些非劣组装方案中选择一个满足用户需求的组装方案。本文主要讨论第一个过程。搜索出一个构件组装模型中的所有非劣组装方案是一个 NP-Complete 问题，如果采用枚举算法，当问题规模较大时其求解效率是非常低的。为了有效地提高非劣组装方案搜索的效率，本文提出了一个基于向量编码的构件组装方案搜索遗传算法(VGA-CS)，VGA-CS 从多目标优化的角度出发，搜索出满足所有约束条件一组近似非劣组装方案。

5.1 向量编码

采用遗传算法进行构件组装的搜索的关键是如何将构件组装方案编码为一个染色体，并通过染色体之间的交叉和变异等操作，产生具有更高目标准则值的新的染色体。目前遗传算法中常见的染色体编码方式有 0-1 编码、顺序编码、实数编码、整数编码和矩阵编码[10]，但是这些编码方式很难表达

构件组装模型中构件接口之间的连接关系。为此，本文提出一种新的编码方式：向量编码。该编码方式可以非常方便地表示构件组装模型中构件接口之间的连接关系。基于向量编码方式，可以有效地进行构件组装方案的搜索以及不同组装方案之间的交叉和变异操作。

向量编码类似于顺序编码，其染色体的前 n 位中的每个基因代表一个功能，后 m 位中的每个基因代表一个候选构件，其中，n 为系统中功能数量，m 为系统中候选构件的数量。为了表达功能的绑定构件集以及构件接口之间的连接关系，每个基因被编码为一个向量，表示该基因所代表的功能的绑定构件集或者构件的请求接口集。如果一个基因代表一个功能，则该基因表示为一个 1×1 向量，其中，基因位的取值范围是该功能的候选绑定构件集；如果一个基因代表一个构件，则该基因表示为一个 $(k+1)\times1$ 向量，其中，k 为该基因所代表的构件的请求接口数目，向量的 1 到 k 位的每个元素表示该构件的一个请求接口，元素的取值范围是该接口所依赖的构件集。由于在一个构件组装方案中，并不是所有的构件都会被选择，因此，对于染色体中的每个构件，我们在其所对应的向量的最后一位设置为一个标志位表示其是否被使用，如果标志位的取值为 0，表示该构件没有被使用；否则，如果标志位的取值为 1，则表示该构件被使用。

图 2 描述了染色体的向量编码的表示方式的示意图。下面我们介绍相应符号的含义。

- R^i：表示对功能 f_i 的一个绑定构件，R^i 的取值范围为 f_i 的绑定构件集 CCS_i 中的每个构件。
- R_k^i：表示对构件 C_j 的第 k 个请求接口 RI_{jk} 所依赖的构件的一个接口配置，R_k^i 的取值范围为 C_j 的第 k 个请求接口 RI_{jk} 所依赖的构件集 $Dep(C_j.RI_{jk})$。
- F_j：构件 C_j 选择标志位，如果 $F_j=0$，表示 C_j 没有被选择；如果 $F_j=1$，则表示 C_j 被选择。当一个构件没有被选择时，可以不对其请求接口进行配置，此时令 $R_k^i=-1$。

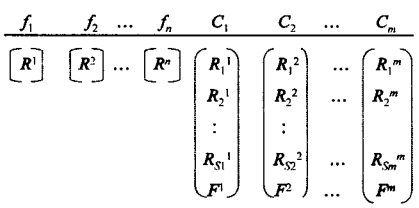

图 2　染色体的向量编码

5.2　初始化操作

初始化操作是指对初始种群中的每条染色体的基因位进行赋值，由于构件之间依赖关系以及约束条件的存在，因此在对染色体进行赋值时，必须保证组装方案的合法性。下面，我们给出染色体初始化操作的步骤。

Step 1　生成一个初始的空染色体 P（空染色体是指 $R^i=-1$，$R_k^i=-1$，$F_j=0$）。

Step 2　对染色体 P 的 1 到 n 的每个基因位依次进行随机赋初值，并将其所对应的构件增加到有序集合 L 中。

Step 3　依次检查 L 中的每个构件 j，如果染色体的第 $n+j$ 个基因的标志位 $F^j=0$，则初始化第 $n+j$ 个基因位的取值，设定构件 j 的每个输出接口所依赖的构件，然后将第 $n+j$ 个基因值所对应的构件增加到集合 L 中；如果染色体的第 $n+j$ 个基因的标志位 $F^j=1$，则不改变第 $n+j$ 个基因位的取值。

Step 4　转至 Step3，直到集合 L 中的每个构件都被检查到。

在上述步骤 2 和 3 中，如果所取得基因值不满足约束条件，则需要重新赋值，直到满足约束条件为止。

5.3　进化操作

交叉和变异是遗传算法中的两个重要的进化操作，针对本文提出的向量编码方式，下面我们给出相应的交叉和变异操作。

(1)交叉操作

交叉操作采用双切点交叉，其操作过程如下。

Step 1　在种群中选择两个染色体 P_1 和 P_2 作为两个父代染色体，并且生成两个空的子染色体 C_1 和 C_2。

Step 2　在 1 到 $n-1$ 之间随机选择一个整数 x 作为切点 1，在 n 到 $n+m-1$ 之间随机选择一个整数 y 作为切点 2，切点 1 和切点 2 将 P_1 和 P_2 划分为 4 个区间：$[1,x]$，$[x+1,n]$，$[n+1,y]$和$[y+1,m]$。

Step 3　将父代染色体 P_1 和 P_2 的区间$[1,x]$部分的基因值分别赋给子代染色体 C_1 和 C_2 的相应部分，将父代染色体 P_1 的区间$[x+1,n]$部分的基因值赋给子代染色体 C_2 的相应部分，将父代染色体 P_2 的区间$[x+1,n]$部分的基因值赋给子代染色体 C_1 的相应部分。

Step 4　依次检查 C_1 的区间$[1,n]$部分的基因值所对应的每个构件，并将其增加到有序集合 L_1 中。

Step 4.1　从 L_1 中取出构件 j，(1)如果 $n+j\in[n+1,y]$，检查父代染色体 P_2 的第 $n+j$ 个基因的标志位 F^j 的取值，如果 $F^j=1$，将 P_2 的第 $n+j$ 个基因值所包含的构件增加到 L_1 中，并将其基因值赋给 C_1 的第 $n+j$ 位，否则，将 P_1 的第$n+j$个基因值所包含的构件增加到 L_1 中，并将其基因值赋给 C_1 的第 $n+j$ 位；(2)如果 $n+j\in[y+1,m]$，检查 P_1 的第 $n+j$ 个基因的标志位 F^j 的取值，如果 $F^j=1$，将 P_1 的第 $n+j$ 个基因值所包含的构件增加到 L_1 中，并将其基因值赋给 C_1；否则，将 P_2 的第 $n+j$ 个基因值所包含的构件增加到 L_1 中，并将其基因值赋给 C_1。

Step 4.2　重复步骤 4.1，直到 L_1 为空。

Step 5　依次检查 C_2 的区间$[1,n]$部分的基因值所包含的每个构件，并将其增加到有序集合 L_2 中。

Step 5.1　从 L_2 中取出构件 j，(1)如果 $n+j\in[n+1,y]$，检查父代染色体 P_1 的第 $n+j$ 个基因的标志位 F^j 的取值，如果 $F^j=1$，将 P_1 的第 $n+j$ 个基因值所包含的构件增加到 L_2 中，并将其基因值赋给 C_2 的第 $n+j$ 位；否则，将 P_2 的第 $n+j$ 个基因值所包含的构件增加到 L_2 中，并将其基因值赋给 C_2；(2)如果 $n+j\in[y+1,m]$，检查父代染色体 P_2 的第 $n+j$ 个基因的标志位 F^j 的取值，如果 $F^j=1$，将 P_2 的第 $n+j$ 个基因值所包含的构件增加到 L_2 中，并将其基因值赋给 C_2，否则，将 P_1 的第 $n+j$ 个基因值所包含的构件增加合 L_2 中，并将其基因值赋给 C_2。

Step 5.2　重复步骤 5.1，直到集合 L_2 为空。

(2) 变异操作

变异是对染色体按照变异概率任选若干个基因位改变其值，其操作过程如下。

Step 1　在 1 到 n 之间随机选择一个数 x 作为变异点，然后改变染色体 P 的第 x 个基因位的取值。

Step 2　依次检查染色体 P 的区间$[1,n]$部分的基因值所对应的每个构件，并将其增加到有序集合 L 中。

Step 2.1　依次检查 L 中的每个构件 j，如果染色体的第 $n+j$ 个基因的标志位 $F^j=0$，则初始化第 $n+j$ 个基因位的取值，设定构件 j 的每个请求接口所依赖的构件。然后将第 $n+j$ 个基因值所对应的构件增加到集合 L 中，如果染色体的第 $n+j$ 个基因的标志位 $F^j=1$，则不改变第 $n+j$ 个基因位的取值。

Step 2.2　重复步骤 2.1，直到集合 L 中的每个构件都被检查到。

Step 3　依次检查每个基因 $n+j(1\leqslant j\leqslant m)$，如果构件 j 不属于集合 L 并且 $F^j=1$，则令 $F^j=0, R_k=-1(1\leqslant k\leqslant S_j)$。

5.4　选择操作

多目标遗传算法与单目标遗传算法的一个主要区别在于适应值的分配和个体选择机制。适应值分配就是给每个染色体赋予一个标量适应值，从而对当代群体内个体之间的优劣关系进行排序。本文采用两个步骤进行个体适应值分配：个体分级和同级排序。

对种群 POP_t（t 为进化代数）内的所有个体进行分级的过程如下：

Step 1　设置初始序号 $r=1$。

Step 2　求出种群中的 Pareto 最优个体，定义这些个体的排序号为 r。

Step 3　从种群中去掉 Pareto 最优个体，并更改排序号 $r=r+1$。

Step 4　转到步骤 2，直到处理完种群中的所有个体。

对于出现多个个体具有相同排序号的情况，可以通过个体多样性的保持策略来对同级个体进行排序。本文采用如下方法对同级个体进行排序：

Step1　计算种群 POP_t 中任意两个染色体 P_i 和 P_j 之间的欧式距离：$D(P_i,P_j)=\sqrt{\sum_{k=1}^{n}\left(\frac{f_k(P_i)-f_k(P_j)}{f_k^{\max}-f_k^{\min}}\right)^2}$，其中，$n$ 为目标函数的个数，$f_k(P_i)$ 和 $f_k(P_j)$ 分别为染色体 P_i 和 P_j 的第 k 个目标函数值，$f_k^{\min}$ 和 $f_k^{\max}$ 分别为第 k 个目标函数的最小值和最大值。

Step2　基于欧式距离，计算种群 POP_t 中每个染色体的 $P_i(i=1,2,\cdots,s)$ 的小生境数：$N_t(P_i)=\sum_{j=1}^{s}\max\{1-\frac{D(P_i,P_j)}{\sigma_{share}},0\}$，其中，$\sigma_{share}$ 为预定义的小生境调节参数。

基于个体分级和同级排序，我们可以比较种群内的任意两个染色体适应值的大小。设 P_i 和 P_j 是种群 POP_t 内任意的两个染色体，如果 $r_t(P_i)>r_t(P_j)$，则 P_i 的适应值大于 P_j 的适应值，如果 $r_t(P_i)=r_t(P_j)$，并且 $N_t(P_i)<N_t(P_j)$，则 P_i 的适应值大于 P_j 的适应值。

选择种群中需要复制的个体采用联赛法，其执行过程如下：

Step 1　从种群 POP_t 中随机选择两个染色体 P_i 和 P_j。

Step 2　如果 P_i 和 P_j 中的一个满足约束条件，而另一个不满足约束条件，那么我们选择满足约束条件的染色体作为被选择的染色体；如果两个都满足约束条件，则取适应值大的染色体作为被选择的染色体；否则，转到步骤 3。

Step 3　转到步骤 1，直到选择到一个满足约束条件的染色体或者选择次数达到了一个预定义的上限，对于前者，选择过程结束，对于后者，我们比较其不满足约束条件的程度（可以根据它们满足约束条件的个数来判断），其中，满足约束条件大的染色体作为被选择的染色体。

5.5　精英策略

由于算法的随机性往往导致在优化过程中优良解的丢失，因此，为了保证算法的优化性和收敛性，本文采用如下两种精英策略来保持优良解。

(1)父-子代种群合并策略

为了能够将父代的优良个体遗传到下一代，可以将父代种群和子代种群合并成为一个种群，然后对合并后的种群中的个体按照非劣解等级和同级个体比较的方法排序，并取前 s（s 为种群中的数量）个染色体作为下一轮进化操作的父代种群。

(2)辅助种群策略

为了避免进化过程中优良个体的丢失，可以采用一个辅助种群来存储每一代所产生的非劣解。在进化过程的每一代，可以将新产生的满足约束条件的非劣解加入到辅助种群中，然后重新计算辅助种群中解的支配关系，求出新的非劣解集，并且删除被新的非劣解支配的个体。

5.6　VGA-CS 算法

基于上述分析，下面我们给出 VCA-CS 算法的描述：

算法 1　VCA-CS 算法

输入：目标函数 F、约束条件集 C、种群规模 s、进化代数 T、交叉概率 p_c、变异概率 p_m 和小生参数范围参数 σ_{share}。

输出：Pareto 解集 POP^*。

算法描述：

Step 1　令 $t=0$。

Step 2　生成初始种群 POP_t 和辅助种群 POP_t'（$POP_t'=\phi$）。

Step 3　计算 POP_t 中每个个体的适应函数值，并对 POP_t 中的所有个体进行分级和同级排序。

Step 4　将 POP_t 中等级为 1 的个体加入到 POP_t' 中。

Step 5　计算 POP_t' 中的非劣个体集 POP_t^*'，保留 POP_t^*' 中的个体，删除 $POP_t'-POP_t^*'$ 中的个体。

Step 6　如果 $t>T$，令 $POP^*=POP_t'$，输出 POP^*，否则，转到步骤 7。

Step 7　通过联赛法从 POP_t 中选出两个父代个体，然后按照概率 p_c 进行交叉操作，产生两个子代个体，并对这两个子代个体按照概率 p_m 进行变异操作，产生新的子代个体。

Step 8　重复步骤 7，直到子代种群 POP_{t+1} 的个数为 s。

Step 9　将父代种群 POP_t 和子代种群 POP_{t+1} 合并为一个种群，令 $POP_t=POP_t\cup POP_{t+1}$，$t=t+1$。

Step 10　对 POP_t 中的个体进行分级和同级排序，然后：

a)将 POP_t 中等级为 1 的个体加入到 POP_t' 中，计算 POP_t' 中的非劣个体集 POP_t^*'，保留 POP_t^*' 中的个体，删除 $POP_t'-POP_t^*'$ 中的个体。

b)保留 POP_t 中的前 s 个，并删除其它个体。

Step 11　转到步骤 6。

在上述算法中，步骤 2 的生成初始种群中的每个染色体的方法见第 5.2 节。步骤 4 和步骤 8 的个体分级和同级排序方法见第 5.4 节。步骤 5 和步骤 10.a)的保持优良解的说明

见第 5.5 节。算法中所用到的交叉和变异操作见第 5.3 节。算法 1 的终止条件为进化代数。

VCA-GS 算法的时间复杂度主要取决于种群的规模和迭代次数。算法的每次迭代的时间主要取决于两个部分：个体分级排序和保持优良解。个体分级排序的时间复杂度为 $O(2(|F|+|C|+m+n)(2s)^2+2s\cdot\log(2s))$，保持优良解的时间复杂度为 $O((|F|+|C|)(2s)s^*)$，其中，s^* 为辅助种群的上限。因此，整个算法的时间复杂度为：$O(((|F|+|C|+m+n)s^2+ss^*)T)$。

6 实验分析

实验程序采用 Java 语言开发，算法运行微机的配置为 PentiumIV 1.73GHZ 处理器，0.99G 内存，操作系统为 Windows XP。我们假定依赖构件的数量等于绑定构件的数量，每个构件的请求接口的最大数量为 2，每个请求接口所依赖的构件的最大数量为 5。实验中每个功能的候选构件集，构件的接口以及接口之间的连接关系，每个构件的 QoS 以及 QoS 的约束条件均是随机生成的。在进行实验中，由于 QoS 约束的存在，因此有可能产生空解，在空解的情况下只要重新调整各个参数的随机值，必然最终会得到相应的解。

实验以 10 个功能为例，分别考虑每个功能的候选构件数量为 10，15 和 20，进化代数为 100，200，300 和 400 的情况下，利用 VGA-CS 算法求解满足约束的非劣组装方案的 CPU 时间开销。对于每一种情况，算法分别运行 10 次取平均值。图 3 描述了随着候选构件数量的增加，在不同进化代数下，执行算法的 CPU 开销。从该图可以看出，随着候选构件数量的增加，CPU 的执行时间并没有明显增加，CPU 的执行时间与进化代数有关，并随着进化代数的增加而呈线性增加。在功能数量为 10，每个功能的候选构件数量为 20 的情况下，执行时间为 10s 左右。这一求解规模可以满足构件组装方案选择问题的需求。

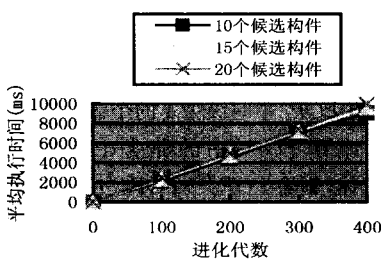

图 3 VGA-CS 的平均执行时间

结束语 为了提高应用系统的构件组装质量，提出了一个面向配置的构件组装模型，基于此模型给出了一个基于向量编码构件组装方案选择遗传算法（VGA-CS）以帮助用户选择最佳的构件组装方案。与目前所提出的面向全局构件选择算相比，VGA-CS 具有如下优点：

（1）能够有效地表示构件之间的依赖关系。目前大多数面向全局 QoS 优化的组装方案选择方法都是将构件看作是一个独立的单元，不考虑构件之间的依赖关系，从而限定了其适用范围，而本文所提出的构件组装模型既适用于独立构件的情况，也适用于构件之间具有复杂依赖关系的情况，因此能够适用于各种类型的体系结构。

（2）VGA-CS 是一个基于多目标优化的构件组装选择算法，其可以得到一组满足约束条件的非劣组装方案，与基于单目标优化得到一个优化解相比，可以更好地满足用户的需求。

参 考 文 献

[1] 杨芙清. 软件工程技术发展思索[J]. 软件学报，2005，16(01)：1-7

[2] Ruhe G. Intelligent Support for Selection of COTS Products[C]// Proceeding of the Net. ObjectDays 2002. Erfurt: Springer 2003

[3] Sheng Jinfang, Chen Songqiao, Wang Bin. COTS Evaluation and Selection Based on Requirements Decomposition[J]. Chinese Journal of Electronics, 2005(1): 62-67

[4] Cui Yi, Nahrstedt K. QoS-Aware Dependency Management for Component-Based Systems[C]// Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing. Washington, DC, USA, 2001

[5] 廖渊，唐磊，李明树. 一种基于 QoS 的服务构件组合方法[J]. 计算机学报，2005，29(4)：627-634

[6] 唐磊，廖渊，李明树，等. 面向普适计算的服务构件动态部署问题及算法[J]. 计算机研究与发展，2007，44(5)：815-822

[7] 战德臣，徐晓飞，李成严. 时间-成本双主线 ERP 管理体协研究[J]. 计算机集成制造系统，2002，8(8)：635-639

[8] 赵俊峰，王亚沙，谢冰，等. 一种支持构件服务质量的构件管理框架[J]. 电子学报，2004，32(a2A)：165-168

[9] Sedigh-Ali S, Ghafoor A. A graph-based method for component-based software development [C]// Proceeding 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. Feb. 2005: 254 - 259

[10] 汪定伟，王俊峰，王洪峰，等. 智能优化方法[M]. 北京：高等教育传版社，2007：4

（上接第 78 页）

架构，更好地保证了企业信息的安全使用。

参 考 文 献

[1] Aboba B, Blunk L, Vollbrecht J, et al. Extensible Authentication Protocol(EAP) [S]. IETF RFC3748, June 2004

[2] Trusted Computing Group. TCG Specification Architecture Overview[EB/OL]. [2007-11-01]. https://www.trustedcomputinggroup.org/groups/TCG_1_0_Architecture_Overview.pdf

[3] Trusted Computing Group. TPM Main Part 1 Design Principles [EB/OL]. [2007-11-01]. https://www.trustedcomputinggroup.org/specs/TPM/tpmwg-mainrev62_Part1_Design_Principles.pdf

[4] Shen J J, Lin C W, Hwang M S. A Modified Remote User Authentication Scheme Using Smart Cards[J]. IEEE Transactions on Consumer Electronics, 2003, 49(2): 414-416

[5] Yang W H, Shieh S P. Password authentication scheme with smartcards[J]. Computers & Security, 1999, 18(8): 727-733

[6] 杨照芳，程小平. 基于 Diff-Hellman 的改进 Yang-Shieh 智能卡认证协议[J]. 西南师范大学学报：自然科学版，2006，31(6)：110-113

[7] Diffie W, Hellman M. New Directions in Cryptography[J]. IEEE Transactions on Information Theory, 1976, IT-22(6): 644-654

[8] Stumpf F, Tafreschi O, Roder P, et al. A Robust Integrity Reporting Protocol for Remote Attestation[C]// Second Workshop on Advances in Trusted Computing(WATC'06 Fall). Tokyo, Japan, November 2006