生成式程序设计研究概述*)

范少锋 张乃孝
(北京大学信息科学系 北京 100871)

摘 要 随着软件规模越来越大、软件复杂度越来越高,如何高效地开发出高质量的软件以及如何有效地维护和更新软件都是当前软件方法学研究所关注的重点。为了实现这一目标,已经陆续出现一些有效的方法和技术。Krzysztof Czarnecki 和 U. Eisenecker 融合先进方法和技术的精髓,提出了一种新的软件工程范型——生成式程序设计:基于为软件系统族进行建模,对给定的需求规格说明,利用配置知识,把基本的可重用构件根据需求进行自动化的配置以产生高定制、优化的软件产品。生成式程序设计的基础是面向系统族的生成式领域模型,该模型包括三个基本成分:问题域、解域和连接这两个域的配置知识。生成式程序设计包含两个开发周期:一个是设计和实现生成式领域模型,即支持重用的开发;另一个是利用生成式领域模型生产出具体的软件系统,也即利用重用的开发。本文将详细地介绍此软件工程范型的分析与设计方法和实现技术。

关键词 生成式程序设计,系统族,配置知识,支持重用,利用重用,领域工程,特征模型

Survey of Generative Programming

FAN Shao-Feng ZHANG Nai-Xiao
(Department of Information Science, Peking University, Beijing 100871)

Abstract Not only the size of software is becoming larger and larger, but also its complexity is becoming higher and higher. How to efficiently develop software products with high quality and how to effectively maintain software and evolve it become the focus of software methodology. To realize these goals, some effective methods and technologies have been proposed in sucess. Combining the soul of some advanced methods and technologies, Krzysztof Czarnecki and U. Eisenecker proposed a new software engineering paradigm, generative programming. GP is based on modeling software system families such that, given a particular requirement specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary reusable implementation components by means of configuration knowledge. GP encompasses two complete development cycles: one for designing and implementing the generative domain model, that is to develop for reuse; the other for using the generative domain model to produce concrete system, that is to develop with reuse. This paper will discuss the analysis and design method, as well as the related implementing technologies of the new proposed software engineering paradigm.

Keywords Generative programming, System family, Configuration knowledge, For reuse, With reuse, Domain engineering, Feature model

1 引言

传统的软件工程方法主要是针对个别用户的需求、面向单个特定的系统,其目标只是得到一个具体的软件系统,其中分析、设计等开发过程中的信息基本上都没有得到保留,这导致软件的重用度很低,以及软件的后期维护和进一步演化都十分困难。随着软件规模越来越大、软件复杂度越来越高,如何高效地开发出高质量的软件以及如何有效地维护和更新软件都成为软件方法学研究所关注的重点。为了实现这一目标,已经陆续出现一些先进的方法和技术,例如领域工程、泛型编程技术、构件技术、面向方面的编程技术(Aspect-Oriented Programming,下面简称 AOP)、元编程技术等。

Krzysztof Czarnecki 和 U. Eisenecker 融合了部分先进方法和技术的精髓,提出了一种新的软件工程范型——生成式程序设计 (Generative Programming,下面简称 GP)[1]。GP 类似于机械、电子产品的工业化生产方式,力图通过对一些基本构件进行自动化的装配来生产出相应的软件产品。

首先通过一个例子说明 GP 的思想:假设用户现在需要购买一辆允许自己定制的汽车:不是直接得到一辆已经组装好的成品,而是选择一些汽车构件,并把这些构件装配成一辆汽车。实际上这些汽车构件可能并不完全吻合,所以还需要一定的修整才可以进行组装。这种思想正是目前基于构件的软件工程(CBSE)所解决的问题。另外,即使现在所有的构件都是完全吻合的可以直接进行组装,而整个装配过程还是要考虑很多的细节问题。所以理想的办法就是改用抽象的方式来描述用户的需求,然后利用生成器来自动生成满足用户需求的特定系统,这正是 GP 追求的目标。而要实现这一目标需要以下几点:1、设计出符合通用生产线结构的构件以支持重用;2、把将抽象的需求配置成构件组合的模型;3、实现一个生成器来自动完成构件的组装和配置。这三点正好和汽车工业的

*)本文受国家自然科学基金项目(编号 60273001)资助。**范少锋** 博士研究生,主要研究方向为软件自动化、形式化方法。**张乃孝** 教授、博士生导师,主要研究方向为软件方法学。

可互换部件、生产装配线以及利用机器人进行自动化生产相对应。

同样的道理，为了开发出真正可重用的构件，GP 提出把对单个系统的工程转移到整个系统族的工程上来。同时，为了把抽象的需求对应成构件的组合，实现构件的自动化装配，GP 提出一方面对配置知识进行足够具体的建模，另一方面强调利用生成器来实现配置知识。GP 以生成式领域模型(Generative Domain Model，下面简称 GDM，第 2 节将详细介绍)为基础，来生成各个具体的软件系统。GP 包含了两个开发周期：一个是支持重用的开发，即设计和实现生成式领域模型；另一个是利用重用的开发，也即利用生成式领域模型生产出具体的软件系统。其中支持重用最大的特点就是面向系统族，而利用重用的开发周期则要求从整体上对系统进行认真的设计以求最大化地重用在支持重用开发过程中得到的各种可重用信息。

本文第 2 节介绍 GP 的基础——生成式领域模型；第 3 节介绍 GP 的分析与设计方法，包括支持面向系统族的领域工程(Domain Engineering)方法、特征建模(Feature Modeling)技术和领域语言的重要作用；在第 4 节，将对 GP 的实现技术进行介绍，包括构件技术、泛型编程技术、AOP、元编程技术、以及意图式的编程技术(Intentional Programming)；最后总结本文，并对 GP 的进一步发展进行展望。

2 生成式领域模型

由 Krzysztof Czarnecki 和 U. Eisenecker 提出的这种新的软件工程范型——生成式程序设计的定义为：基于为软件系统族进行建模，以特定的需求规格说明(Requirement Specification)为基础，结合配置知识(Configuration Knowledge)对基本的可重用的实现构件进行自动化的配置[2~5]，以产生高定制、优化的软件产品。它将视点集中在软件系统族，而不是单个的特定系统。

GP 在 GDM 的基础上生成不同的软件系统实例，从而避免了传统的每次都是从头开始设计、开发一个特定系统。所以说，GDM 是 GP 的基础(见图 1)，它包含以下三个基本成分：

1. 问题域(Problem Space)，即详细说明整个软件系统族的各种方式，包括用来表达用户需求的各种领域概念和特征(具体介绍见 3.2 节)。如何描述用户的需求主要依靠领域语言(见 3.4 节)。

2. 解域(Solution Space)，即用来装配软件系统族中某个特定软件系统的实现构件以及构件之间的可能配置。如何实现构件以及如何表示构件之间的结构配置等都需要相关实现技术的支持。

3. 配置知识(Configuration Knowledge)，它将抽象的规格说明与其相应的实现进行匹配，是连接问题域和解域的桥梁，具体内容包括各种非法的特征组合、缺省设置、缺省依赖性、构建规则和最优化操作等。可以利用生成器(具体在 4.4 节介绍)来实现配置知识。

这里 GDM 强调把问题域和解域进行区分，主要的好处是为了使它们能够各自相对独立地进行演化。由于用户通过问题域的领域语言来定制特定的系统，然后利用实现配置知识的生成器将抽象的规格说明匹配成相应实现构件的具体配置，所以我们可以在解域中添加新的构件或改进某构件，只要仍然满足问题域中对特定系统的抽象描述，就无需对用户代码进行任何的改动，而只需对生成器进行一定的修改即可。另外，随着领域自身的发展，会不断出现一些新的领域概念，那么就需要对问题域的领域语言进行演化甚至是替换为另一种领域语言。由于 GDM 把问题域和解域进行区分，这时只需对负责匹配这两个域的生成器进行一定的修改，而无需对解域进行任何的改动。可以看出，这里 GP 显式地提出配置知识的概念是十分有意义的。

3 GP 的分析与设计方法

GP 包含两个开发周期：一个是支持重用的开发，也就是设计和实现生成式领域模型；另一个就是利用重用的开发，也即利用生成式领域模型生产出具体的软件系统。其中支持重用最大的特点就是面向系统族，为了支持真正意义的重用，GP 提出了面向系统族的概念。3.1 节将首先介绍支持面向系统族的领域工程方法，通过介绍我们可以看出 GP 是如何把领域工程作为其分析、设计方法的。3.2 节讨论 GP 如何利用特征建模技术得到重用度很高的领域模型及其对 GP 的重要作用。接着 3.3 节介绍如何对某一特定领域的领域模型加以扩充得到 GP 的基础——GDM。最后 3.4 节将讨论领域语言对于 GP 的重要性。

3.1 面向系统族的领域工程

领域工程包括领域分析、领域设计和领域实现这三个阶段。其中，领域分析包括确定领域范围、分析领域中系统需求的共性和可变性并建立其领域模型。该阶段最主要的成果就是得到相应的领域模型。领域设计就是得到该领域整个系统族的体系结构，并设计出相应的生产方案。领域实现，就是利用相关的实现技术来实现整个系统族的体系结构和所需的实现构件。

领域工程与传统的软件工程方法有着显著的不同：传统的软件工程方法包括面向对象的分析与设计方法(OOA&D)，只是针对个别用户的需求开发一个特定的系统。我们说这些方法都是单系统(Single-system)的软件工程方法。而领域工程则强调面向系统族，它考虑该领域内不同用户的各种各样的需求，致力于开发出具有较高可重用性的软件。所以说，领域工程是一种针对多系统(Multi-system)的软件工程方法。值得注意的是，领域工程是一个反复的、逐渐精化的过程。在实施领域工程的每个阶段中，都可能返回到以前的步骤，对以前步骤得到的结果进行修改和完善，然后再回到当前步骤，在新的基础上继续进行本阶段的行为。

领域工程是面向系统族的方法，领域工程的开发过程是支持重用的开发。一般的软件开发过程就是从需求分析开始，然后进行设计、实现，最后进行整合与测试。而利用重用的开发则要求从整体上对系统进行认真地设计以求最大化地重用在支持重用开发中得到的各种可重用信息，这个过程属于应用工程(Application Engineering)(见图 2)。对于一个具体的应用，GP 中利用重用的开发包括：在需求分析阶段，在已有领域模型的基础上，利用领域模型中的特征来描述用户的需求。如果领域模型中包含新的用户需求，那么这个新的需求就要用户自己进行相应的设计和开发(这个过程类似于一般的软件开发过程)，同时这个新需求也会反馈给支持重用的领域工程，然后对原领域模型进行相应的精化和扩充，以便于今后的重用。最后就是组装现有的可重用构件和用户自开发的构件得到满足用户需求的具体软件系统。显然，这里的利用重用的应用工程过程和一般的软件开发过程有着显著的不同。

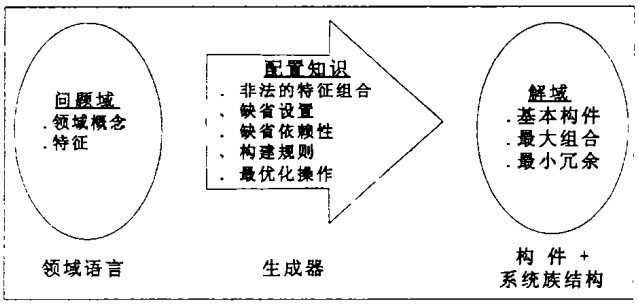

图1 生成式领域模型

3.2 特征和特征模型(Feature Model)

所谓特征,就是被关注的概念的显著性质[7]。这种定义把特征围绕着客观世界的概念(Concept)来定义,"概念"较面向对象技术里的"类"更为抽象:概念的实例无预定义的语义,而类的实例也就是对象却包含对象的表示、行为和状态等语义信息。所以概念和特征能更自然地抽象各领域中不同应用系统的异同点,以及它们之间的依赖关系。从实现的角度,可以把特征分为三类:

- 原子特征,可直接实现成构件技术中的基本构件;
- 方面(Aspect)特征,可直接实现对应为面向方面的编程技术中的方面;
- 抽象特征,可对应实现成一些基本构件和方面的组合。

特征模型是领域分析阶段得到的最主要成果,也是领域设计和领域实现的基础。该模型独立于实现,在较高的抽象层次上,为用户提供了一种对可变性(Variability)的显式、精确的表示。具体得到特征模型的过程就是特征建模[6]的过程。与一般应用软件相比,可重用软件通常包含更多的可变性,而特征建模则是识别和获取可变性的重要技术,所以可以说特征建模技术是面向重用的软件工程的一个必需的技术。

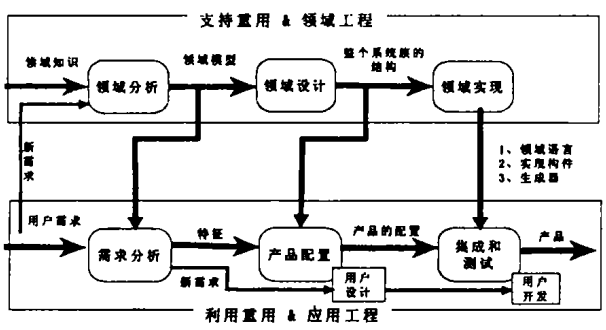

图2 领域工程与应用工程

特征模型在GP中起到了非常重要的作用:1. GP的重要特点是面向整个系统族,而特征模型十分适用于定义一个系统族的范围和配置方面的性质。2. 特征模型是设计整个系统族的体系结构以及实现各个构件的基础。3. 特征模型中还提供了被用来自动生成软件系统族中某特定软件系统的配置知识。4. 特征模型允许派生出被用户用来定制特定族成员的可配置的用户需求的模型。

一个概念的特征模型表达了该概念的意图(Intention),它所描述的实例的集合叫做此概念的外延(Extension)。特征模型包括特征图和一些关于特征的具体描述或约束,其主要部分是特征图。

下面将结合一个简单例子说明特征图(见图3)。特征图中结点和边构成一个树状结构,其根部为一个概念结点,表示一个概念;其他结点表示特征,叫做特征结点。这些特征结点按照包含于概念实例的标准可分为以下几类:

1. 必需特征结点:由以实心点结束的简单边指向的特征结点表示,表示该特征必然包含于该概念的实例当且仅当该特征结点的父结点也包含于此实例,如图中的"发动装置"、"马力"两个特征结点。

2. 随意特征结点:由以空心点结束的简单边指向的特征结点表示,表示该特征可能包含于某概念的实例当且仅当该特征结点的父结点也包含于此实例,如图中的"空调"结点。

3. 选择特征结点:由通过弧线连接的一组边来表示,表示该概念的实例中至少包含其中一个特征结点,如图中所示的"手工"、"自动"两个选择特征结点。

通过图3这个简单的特征图,清晰地描述了汽车这个概念的各个实例之间的共同点和不同点,其共同点由必需特征来体现,不同点(也即可变点)就体现在随意特征和选择特征上。由图3可作如下分析:所有的汽车实例都具有发动装置、马力等特征;但有的汽车发动装置是手动,有的是自动,有的是两者兼有;有的汽车可能有空调,有的不具备空调。通过这个简单的特征图,就可以简单且直观地描述2*3=6种不同的汽车实例。

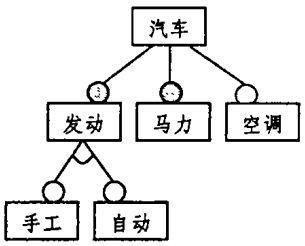

图3 汽车概念对应的特征图

3.3 对领域工程的扩充

领域工程为GP提供了必要的基础:它支持面向系统族,包括为特定领域进行建模、设计和实现具有较高可重用性的构件,还包括为配置知识进行建模等。但是,领域工程方法还是仅局限于为某一特定领域开发和实现领域模型,而GP关注的不单是某一个特定领域,而是可用来在用户需求规格说明的基础上自动生成具体软件系统的生成式领域模型(第2节已经介绍)。

为了适用于GP,需要对领域工程方法进行以下两方面的扩充:1. 开发出用来定制具体族成员的合适的方式。具体的系统可以用文本式的或图形式的领域语言来进行定制(例如详细的规格说明),而定制过程则可由交互式工具(比如文本式或图形等)支持。2. 为配置知识进行足够具体的建模以至于可以进行自动化的操作。要支持对非法特征组合、缺省设置、缺省依赖性、构建规则以及优化等配置知识的建模。而且这些配置知识必须足够具体且精确,可以利用生成器(见4.4节元编程技术)来进行实现。

3.4 领域语言对于GP的重要性

GP根据给定的需求规格说明,结合配置知识对可重用的构件实现自动化的配置以生成满足需求的具体软件系统。那么支持用户定制特定的软件系统、描述具体用户需求的任务就落在领域语言的身上了。同时,由于GP强调对配置知识进行足够具体的建模,以支持把需求规格说明自动匹配成相应实现构件的组合,那么描述配置知识的任务也同样由相应

的领域语言来完成。

领域语言[8]是面向特定领域或特定问题的程序语言，通常为文本(如SQL、HTML)或图形(如GUI builder中GUI的图形描述)等形式，由于规模一般比较小，所以又叫做小语言。领域语言提供了合适的领域概念，同时又隐藏相关的实现细节，提供了较高层次的抽象，因此为领域问题提供了较强的表达能力，通常用领域语言书写的程序段较通用语言更精确且易读。所以说，利用领域语言来表达用户的需求是十分自然的。在GP中，领域语言被用来定制具体族成员和描述相应的配置知识，可见，领域语言在GP中起到了十分重要的作用(见图5)。

由于领域语言对GP有着重要的作用，所以如何快速开发出合适的领域语言也是GP所关注的重要问题。针对此问题，GP提出把领域语言视为模块化、可组装的单元。也即把每个领域语言视为一个可组装的单元，这样就有或大或小的各种领域语言单元，它们可以以不同的方式进行组装配置，从而大大提高领域语言的重用性和扩展性。模块化的领域语言可划分为以下两种：一种是封装式的(Encapsulated)领域语言，它们只是能够进行简单的组装，并且在组装的时候不会互相影响各自的语义；另一种是方面式的(Aspectual)领域语言(如表示同步限制或错误处理的领域语言)，它们会在组装的时候影响其他语言的语义。从另一个角度来看这两种划分就对应了构件技术和AOP的思想。

4 GP的实现技术

在前面几节，我们主要介绍了GP的主要思想和设计与分析方法。通过介绍，我们可以看出GP决不是凭空产生的，它是充分借鉴了已有的优秀成果，在一些先进方法(包括软件重用的思想、领域工程方法等)的基础上提出的。作为一种软件工程范型，GP也需要一些实现技术的支持。实际上，GP的特点之一也在于它很好地结合了当前流行的一些方法和技术，比如构件技术、泛型编程技术、AOP、元编程技术等。融合这些先进的方法和技术，提出演化式(Evolving)的GP方法体现了当今软件方法学研究的趋势。

4.1 构件技术

关于构件，最早源于Douglas McIlroy在1968年发表的著名论文"Mass Produced Software Components"。其中提出了"可重用构件"的概念[11]。经过这么多年的发展，大家已经公认构件技术是实现最小化代码复制、最大化重用的有效技术[12,13]。对于构件的认识，不同的环境有着不同的定义。在GP里，我们把它认为是构建不同软件系统的组成部分，它内部封装了一些服务，并有定义明确的接口，其功能完全通过接口来体现。构件的实现对外完全隐藏。面向构件的开发原则把软件开发的重点转移到构件的组装配置上。如今，面向构件的开发原则主要通过一些构件技术，包括OMG CORBA[14]、Microsoft COM/DCOM(或.NET)[15]以及Sun JavaBean/EJB[16]等来实现。

GP利用配置知识对基本的可重用构件进行自动化的配置来生成满足用户需求规格说明的软件系统。这里，基本的实现构件是实现GP的基础，由此可见构件技术对GP的重要性(见图5)。领域工程和特征建模是GP的分析和设计方法，在经过分析设计得到特征模型的基础上，可将其原子特征利用构件技术对应实现成一个基本构件，而抽象特征可由一些基本构件的组合来实现。

但是和前面提到的一般的构件技术相比，GP在对构件的处理上又有着显著的区别：1、用户通过抽象的规格说明详细描述所需求的一些性质，由构件库自动生成满足需求的构件，而不是在构件库里通过各种方式查找一个具体的构件；2、GP在自动生成特定构件后会利用生成器对构件进行自动地装配，而不同于目前在查找到满足需求的构件后仍然进行手工组装的方式。3、由于是通过声明所需构件的性质，而不是从库中查找挑选具体的构件，通过这种方式提高了代码的抽象层次。这样即使当构件库进行了更新升级后，这个新版本的构件库依然会按照用户抽象的声明所需求的性质来自动生成满足用户需求的、甚至是功能更好的构件，而且无需对客户端代码做任何的修改。显然，通过这种方式提高了抽象层次，更好地支持重用。

4.2 泛型编程技术(Generic Programming)

泛型参数最初是在函数式语言ML[9]中提出的。后来，随着Ada语言的标准化过程，泛型编程技术也开始赢得人们的关注。泛型编程技术就是用抽象的参数化的方式来表述算法和数据结构。它的最重要也是应用最广泛的一个实例就是标准模板库STL (Standard Template Library)[10]。随着STL成为ANSI/ISO标准C++的一部分，泛型编程技术已经成为大家十分关注的技术，其目标是减少代码的复制以及最大化地实现重用。

泛型编程技术把某个领域表示为一些高度泛化的、抽象的构件的集合，这些构件可以通过不同的方式进行组合来生成有效的具体程序。在GP中，泛型编程技术是一种组织GDM的解域的重要技术，也即由基本的实现构件得到具体软件系统的技术。利用泛型编程技术中的类型参数化、各种多态和参数化构件等机制将有助于得到具有高度可重用性、正交性和非冗余的实现构件。

"泛型"表示对一族或一类事物的描述，而"生成式的"则指不仅能够描述一族或一类事物，而且还具有引发、产生和生成的能力，因此GP在泛型编程技术之上，而泛型编程技术只是用来处理GDM中解域的一个重要技术(见图5)。

4.3 面向方面的编程技术

目前的程序设计方法大多都将视点集中在功能单元上，这些功能单元通常被表述成对象、模块、过程或构件。但是对于诸如错误处理、同步、一致性、保密性等非功能成分，它们通常是分散到多个功能单元之内，不能进行清晰定位式的表述。

针对上述问题，Gregor Kiczales于1997年在他的文章"Aspect-Oriented Programming"[17]里首次提出了AOP的概念。所谓方面，就是贯穿(crosscut)多个模块化单元的性质。AOP的目标是使得能够尽量独立处理软件系统的行为的性质。在AOP系统中，把构件和方面进行交织组合得到一个系统的具体实现。

AOP较泛型技术为生成式领域模型的解域提供了更强大的参数化技术。通过抽象出方面这一概念，就可以直接把特征模型中的方面特征对应实现成一个具体的方面，从而避免了将生成式领域模型问题域中的某个特征对应实现成解域中分散的多个小构件的集合。所以说，AOP是关于GDM解域的重要实现技术(见图5)。

4.4 元编程技术

GP的目标是实现软件开发的自动化，也即是在给定的需求规格说明的基础上，结合配置知识自动地装配构件，以生成满足用户需求的优化的软件产品。而这里，规格说明的检

查、优化，以及如何装配基本构件等都需要元编程技术[18]。

所谓的元程序(Metaprogram)，就是操作其他程序或它自身的程序。元程序的重要的例子就是生成器(Generator)，它把一个系统抽象的规格说明作为输入，产生该系统的实现。正如工业上的汽车生产自动装配线，在生产线中就包含了大量的有关汽车装配的信息，这样就可以通过该生产线来装配各汽车构件，实现各种汽车生产的自动化过程，这里的生产线就是在不断地被重用。

生成器的具体过程如图 4 所示：首先是检查规格说明的系统是否可以构建，由于规格说明是由领域语言来完成的，通常领域语言不能充分地给出各种非法的特征组合，所以这项工作非常必要；然后结合配置知识，通过计算一些缺省值，来完成规格说明并实现优化；最后是装配相关的实现构件，以生成具体的系统或者构件。

在 GP 中，配置知识以程序的形式来表示，也即利用生成器来实现相应的配置知识，这样生成器就能够根据抽象的需求规格说明来组合装配各个构件以生成满足需求的软件系统。

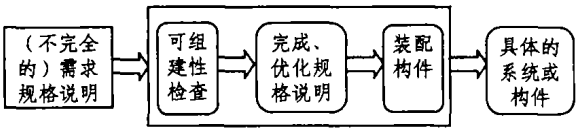

图 4　生成器

4.5　意图式的编程

IP[19,20]是微软剑桥研究院从 1995 年开始由 Simonyi 领导的一个项目，目的是研究一种新的软件开发方法，可用于开发领域语言，该方法支持用户通过表达意图的方式来实现软件。所谓意图，是指语言的抽象或者说是语言的特征。区别于传统的、确定的(Fixed)语言，IP 把语言看成是意图的组装配置。同时，IP 也可以看成 GP 的一个完美的实现平台。

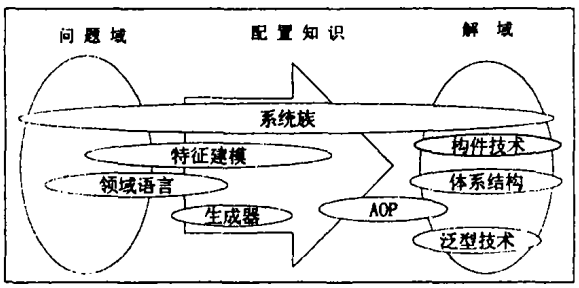

图 5　GP 与其它方法和技术的关系

具体来讲，IP 既是一种编程环境，又是一种元编程环境。作为编程环境，IP 支持应用程序员进行程序的设计开发；作为元编程环境，它又支持语言的实现者开发具有元编程能力的扩展库(Extension Libraries)。另外，IP 不使用文本的方式来表示程序，而是用抽象语法树式的源图(Source Graph)来表示程序。这样就避免了传统语言的词法解析问题，有益于语言的扩展性。在 IP 中，程序的行为通过操作在源图上的方法(Method)来实现，方法定义了程序源的很多方面，包括：可见性、登录(Entry)、浏览(Browsing)、编译、调试等。IP 通过声明意图和实现意图的方法来实现一个新的意图。其中方法被编译到扩展库中，当使用某个意图的时候，就把相应的扩展库载入(Load)到开发环境中。区别于传统的、确定的语言，IP 把语言看成是意图的组装配置，并且在需要某个意图的时候就可以把其载入系统。因此，IP 不仅可以实现特定的领域概念，而且还可以实现领域优化、领域的报错处理和调试支持等。

本节对 GP 的实现技术进行了概要的介绍，最后通过图 5 大致总结一下 GP 与其他方法和技术领域的关系，使得对 GP 有更深入的认识。

总结　GP 基于对整个系统族进行建模，对于给定的需求规格说明，结合配置知识把基本的可重用构件进行自动化的配置来生成满足需求的软件系统。GP 包含支持重用和利用重用两个开发周期。而 GP 最主要的贡献在于提出了面向系统族的概念和强调配置知识这两个方面：只有面向系统族才能实现真正的重用；把配置知识独立地抽象出来对其进行建模，并利用它自动生成满足用户的需求的软件系统。GP 方法不是凭空产生的，它是充分借鉴了已有的先进思想和方法，包括软件重用的思想、领域工程方法等，巧妙地融合了当前流行的各种软件技术，包括构件技术、泛型编程技术、AOP、元编程技术等，在这些优秀成果的基础上提出的。融合这些先进的方法和技术，提出一种演化式的软件开发方法正体现了当今软件方法学研究的趋势。

目前国际上已经出现了一些关于 GP 的论坛[21]，包括一些以 GP 为主题的国际会议[22]，这说明 GP 已经引起了广泛的重视。

当然，我们也认识到它还存在一些不足或有待进一步完善的地方：1. 对于特征模型的支持工具不足。通过前文的介绍可以看出特征模型对 GP 的重要作用，所以说开发出适合于特征建模的支持工具是十分有意义的。2. 目前在 GP 中，提出用各种领域语言来定制特定的软件系统，然而对于领域语言的处理却比较简单，特别是对于领域语言的语义处理太少。对于方面式的领域语言，其在组装时会影响相互的语义，这在很大程度上不能保证组装得到的领域语言的正确性，而这对于使用领域语言的各个应用领域会带来极大的隐患。

由于 GP 提出不久，且目前我国在该方面系统的介绍和研究还很少，但愿这里的介绍能起到一个抛砖引玉的效果。

参 考 文 献

1　Czarnecki K, Eisenecker U. Generative Programming: Methods, Tools, And Applications. Addison-Wesley, 2000
2　Czarneckil K, Ulrich W. Eisenecker Components and Generative Programming, Invited talk. In: Proc. of the 7th European Software Engineeing Conf. France, Sep. 1999, LNCS 1687, Springer-Verlag, Berlin and Heidelberg, Germany, 1999
3　Czarnecki K, Eisenecker U, Glück R, Vandevoorde D, Veldhuizen T. Generative Programming and Active libraries. LNCS, Springer-Verlag, Berlin and Heidelberg, Germany, 1999
4　Eisenecker U. Generative Programming (GP) with C++. In: Proc. of Modular Programming Languages, Linz, Austria, March 1997
5　Czarnecki K. Generative Programming: Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models: [Ph. D. thesis]. Technische Universität Ilmenau, Germany, 1998
6　Schupp S. Feature Modeling. CSCI-4966, Software Construction, Spring 2003
7　Simos M, Creps D, Klinger C, Levine L, Allemang D. Organization Domain Modeling (ODM) Guidebook, Version 2. 0: [Informal Technical Report for STARS]. STARS-VC-A025/001/00, 1996. www. domainmodeling. com
8　van Deursen A, Klint P, Visser J. Domain-Specific Languages - An Annotated Bibliograph. 2000. ACM SIGPLAN Notices
9　http://www. dcs. napier. ac. uk/course-notes/sml/manual. html
10　Musser D R, Saini A. STL Tutorial and Reference Guide. Addison-Wesley, Reading, Massachusetts, 1996
11　Douglas McIlroy M. Mass Produced Software Components, in Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germay, Oct. 1968

（下转第 25 页）

表2 移动 Agent 系统安全机制比较

系统	认证	传输通道	数字签名	对主机的保护	对 Agent 的保护
Telescript	基于 RSA	RC4加密	不支持	通过授权进行访问控制	不支持
Tacoma	不支持	无	不支持	简单的访问控制	不支持
Agent Tcl	基于 PGP	PGP 加密	支持	Agent 执行在 Safe Tcl 环境下;根据安全策略实施粗粒度的访问控制	不支持
Aglets	基于域	对称算法	不支持	通过代理对象实施访问控制	不支持
JumpingBeans	智能认证技术	SSL 协议	支持	四层安全保障体系	Agent 安全事件日志
Mogent	X.509	类似 SSL 的协议	支持	基于信任传递的授权并通过授权进行访问控制	不支持

结论 随着移动 Agent 应用领域的日益广阔,移动 Agent 系统中的安全问题也日益突出。传统的面向主机安全的保护机制主要关注于保证 Agent 平台的安全稳定,而在移动 Agent 所能执行任务复杂化的同时,面向 Agent 的保护技术也逐渐提出并迅速发展。

本文对移动 Agent 系统中存在的安全问题进行了详尽的分析和讨论;并在此基础上总结了现有的特别针对主机和 Agent 的安全保护技术;同时介绍了几种较有代表性的移动 Agent 系统实例,对它们的安全实现机制进行了比较。

不同的 Agent 应用具有不同的安全实现要求,从而需要有选择地对各种安全机制进行高效配置。如何构造一个适用于多种应用的灵活的移动 Agent 系统安全框架,对于移动 Agent 系统中的安全性问题建立一种通用的解决方案,以及如何增强系统中针对 Agent 的保护机制的研究,将是下一阶段工作的重点。

参 考 文 献

1 Lange D B,Oshima M. Progamming and Deploying Java Mobile Agents with Aglets. Addison-Wesley,USA,1998
2 Paul M. Three Tier Client/Server Systems: Building Distributed Systems. Goteborg University Press,Sweden,2001
3 Lange D B. Mobile Objects and Mobile Agents: The Future of Distributed Computing. In:Proc. of European Conference on Object-Oriented Programming,Brussels,Belgium,1998
4 Cao J,Zhou J,Chen D,Chan T S. Mobile Agent Technology and Its Application in Internet Computing. an invited book chapter of "Computational Web Intelligence: Intelligent Technology for Web Applications",World Scientific Press
5 Guilfoyle C,Warner E. Intelligent Agents: The New Revolution in Software. [Technical Report]. Ovum Ltd,London,UK,1994
6 Wayne J,Tom K,Mobile Agent Security. NIST Special Publication 800-19,National Institute of Standards and Technology,Aug. 1999
7 McGraw G,Felten E. Understanding the keys to Java security:the Sandbox and authentication. JavaWorld,May 1997,2(5)
8 Chess D,Grosof B,et al. Itinerant Agents for Mobile Computing. In:Proc. of IEEE Personal Communications,Oct. 1995,2(5):34～49
9 Necula G C. Proof-Carrying Code. In:Proc. of Conf. Record of POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages,1997. 106～119
10 Farmer W,Guttman J,Swarup V. Security for Mobile Agents: Authentication and State Appraisal. In:Proc. of the European Symposium on Research in Computer Security (ESORICS),1996. 118～130
11 Pointcheval D,Stern J. Security Arguments Proofs for Digital Signatures and Blind Signatures. Journal of Cryptology,2000. 361～396
12 Hohl F. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. Mobile Agents and Security,LNCS,1998,1419:92～113
13 Vigna G. Cryptographic Traces for Mobile Agents. Mobile Agents and Security,1998. 137～153
14 Silva F A,Popescu-Zeletin R. An Approach for Providing Mobile Agent Fault Tolerance. In:Proc. of Second Int. Workshop on Mobile Agents,1998. 14～25
15 Yee B. Monotonicity and Partial Results Protection for Mobile Agents,In:Proc. of the Intl. Conf. on Distributed Computing Systems,2003. 582～591
16 Tardo J,Valente L. Mobile agent security and Telescript. In:Proc. of the 41st Intl. Conf. of the IEEE Computer Society (CompCon '96),1996. 58～63
17 Johannsen D,Renesse R V,Schneider F B. An Introduction to the TACOMA DistributedSystem:[Technical Report 95-23]. Department of Computer Science,University of Tromsϕ,Norway,1995
18 Kotz D,Gray R,Nog S,et al. AGENT TCL: Targeting the Needs of Mobile Computers. IEEE Internet Computing,Jul. 1997. 58～67
19 Gray R,Cybenko G,Kotz D,et al. D Agents: Applications and Performance of a Mobile-Agent System. Software-- Practice and Experience,2002,32(6):543～573
20 http://www.jumpingbeans.com
21 陶先平. 基于 Internet 的移动 agent 技术和应用研究:[南京大学博士学位论文]. 南京大学,2001
22 Borselius N. Mobile agent security. Electronics & Communication Engineering Journal,IEE,London,UK,Oct,2002,14(5):211～218
23 Corradi A,Montanari R,Stefanelli C. Security Issues in Mobile Agent Technology. In:Proc. of 7th IEEE Workshop on Future Trends of Distributed Computing Systems FTDCS'99,Dec. 1999. 3～8
24 Wayne J. Countermeasures for Mobile Agent Security. Computer Communications,Special Issue on Advanced Security Techniques for Network Protection,Elsevier Science BV,2000. 1667～1676

(上接第16页)

12 Pour G,Component-Based Software Development Approach: New Opportunities and Challenges. In:Proc. of the 26th Intl. Conf. on Technology of Object-Oriented Languages and Systems (TOOLS),1998
13 Larsson M,Crnkovic I. Development Experiences of a Component-based System. In:Proc. of Engineering of Computer Based Systems (ECBS 2000),IEEE,2000
14 OMG CORBA. http://www.corba.org,2001
15 Box D. Essential COM. Addison-Wesley, ISBN0-201-63446-5, 1999
16 Sun. The Java Language,Java Beans. at http://java.sun.com, Sun Microsystems
17 Kiczales G,et al. Aspect-Oriented Programming. In: Proc. of the European Conf. on Object-Oriented Programming (ECOOP). Springer-Verlag,Finland,1997
18 Czarnecki,Eisenecker U. Template- Metaprogramming. http://home.tonline.de/home/Ulrich.Eisenecker/meta.htm
19 Charles S. The death of computer languages- The birth of intentional programming:[Microsoft Research technical report MST-TR-95-52]. Sep. 1995
20 Charles S. Intentional programming-innovation in the legacy age. Presentation at IFIR WG 2.1 on Algorithmic Language and Calculi. June 1996
21 http://www.generative-programming.org
22 http://www.program-transformation.org/twiki/bin/view/Gpce