国 防 科 技 大 学 学 报
第 23 卷 第 3 期 JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY Vol.23 No.3 2001

文章编号：1001-2486（2001）03-0083-05

CORBA 分布计算软件平台的一种基于定制机制的优化技术*

项君，吴泉源，王怀民

（国防科技大学计算机学院，湖南 长沙 410073）

摘 要：CORBA IDL 语言的描述能力非常弱，缺乏描述分布对象动态行为语义的机制，软件开发的自动化也只限于生成含有基本功能的后端形式，不能满足不同应用的时空需求。通过扩展 IDL 文件中的注释来描述分布对象方法适配的行为，提供了一种定制分布对象方法适配行为的机制。这种机制不仅有利于提高分布系统的时空效率和软件自动化的程度，而且不影响应用的可移植性和互操作性。另外，本文进一步给出了通过扩展 CORBA IDL 解决类似问题的设计模板。

关键词：CORBA；接口定义语言；定制；优化

中图分类号：TP301.6 **文献标识码**：B

Research of Optimization Technology Based the Customization Mechanism in CORBA

XIANG Jun，WU Quan-yuan，WANG Huai-min

（College of Computer，National Univ. of Defense Technology，Changsha 410073，China）

Abstract：The expressiveness of current CORBA IDL is poor. Semantic properties of distributed objects can't be documented formally. The automation of the software development in CORBA is limited to the generation of the stub and skeleton that support basic functions. A kind of mechanism for customizing the dispatching and layering strategies of distributed objects is provided through extending the annotations used for describing objects behavior in IDL files. The mechanism not only improves time and space performance of applications, but also has no effect on portability and interoperability of applications. Moreover, a design pattern to resolve similar problems is presented by extending CORBA IDL.

Key words：CORBA；interface description language；customization；optimization

CORBA（Common Object Request Broker Architecture）支持异构平台的对象互操作，已成为企业级分布计算的主流。CORBA 平台通过 CORBA IDL 中间描述语言定义分布组件的接口，使用 IDL 编译器将接口定义映射为实现语言形式的 Stub 和 Skeleton，构成了应用与 ORB（Object Request Broker）核心之间的粘合层[1]。目前，CORBA IDL 的描述能力非常有限，只能描述分布组件最一般的性质，缺乏描述分布组件语义的机制。另外，在分布应用中，分布对象的很多动态行为都能根据其时空效率和应用需求在设计时确定。其中，方法适配是影响 CORBA 平台时空效率的重要因素之一。然而，目前大多数的 CORBA 平台如 MICO[2]、Orbix[3]、Visibroker[4] 等等，都只能提供单一（或低效）的适配策略，不能满足应用时空效率的要求。

1 方法适配模型

ORB 为完成 Agent 之间的交互，将对象请求映射为实现该请求的方法实现。我们将该映射中通过 GIOP 请求消息[1] 中的操作名获得方法实现的过程称为方法适配。基于上述概念的定义，本文给出方法适配过程的形式化描述，并进行相应的分析。本文约定，文本所指对象若无特殊说明，都是指 CORBA 对象；IDL 都是指 CORBA IDL。

定义 1 对象的操作集。对于类型为 T 的对象（对应于接口类型为 T），定义它的操作集为 $\Omega(T) = \{op_1^T, op_2^T, \cdots, op_n^T\}$，操作集包含对象自定义和所继承的操作；由于对象都继承了根对象 Object，

* 收稿日期：2000-10-10
基金项目：国家 863 项目（863-306-ZD-02）和国家自然科学基金项目（69833030）资助
作者简介：项君（1973-），男，博士生。

所以 $\Omega(T) \neq \Phi$；并且 $Opname(op_i^T) \neq Opname(op_j^T)$，$i \neq j$，$op_i^T$，$op_j^T \in \Omega(T)$，函数 Opname 返回操作 op_i^T 的操作名。

定义 2 对象的自定义操作集。对于类型为 T 的对象，由该类型对象的自定义操作所组成的集合为该对象的自定义操作集，记为 self（T）。对象的自定义操作集可能为空，self（T）$\subset \Omega$（T）。

类型为 T 的对象的所有父对象类型表示为 $super_i$（T），其中 $1 \leqslant i \leqslant m$（m 是 T 的父对象类型个数），为简化表示，我们用 $super_1$（T）表示根对象 Object，则有公式（1）成立。

$$\Omega(T) = (\bigcup_{i=1}^{m} self(super_i(T))) \cup self(T) \tag{1}$$

根据 OMG IDL 语法规范对于有继承关系的接口之间操作名不能相同的约束，则有公式（2）成立：

$$self(super_i(T)) \cap self(super_j(T)) = \Phi \wedge self(super_i(T)) \cap self(T) = \Phi \quad (1 \leqslant i,j \leqslant m, i \neq j) \tag{2}$$

定义 3 对象的操作名集。对应于类型为 T 的对象的操作集 $\Omega(T) = \{op_1^T, op_2^T, \cdots, op_n^T\}$，定义它的操作名集为 $\Psi(T) = \{\gamma_1^T, \gamma_2^T, \cdots, \gamma_n^T\}$，其中 $\gamma_i^T = Opname(op_i^T)$。$\Psi$（T）与 Ω（T）具有一一对应的关系，函数 Opname 是一一映射。

方法适配过程可表示为映射 f，其中，obj 是类型为 T 的对象，用 Impl（obj）表示实现 obj 操作的实现方法集，是对应对象实现方法集的子集。

$$f: \Psi(T) \rightarrow Impl(obj)$$

为进一步分析方法适配，基于功能分离原则，为对象的每个操作定义对应的骨架方法，骨架方法将对应操作请求中的无语义参数映射为具有操作语义的参数，激活实现方法，编码返回值。类型 T 的对象的骨架方法集记为 S（T），S（T）与 Ω（T）具有一一对应的关系；对应于对象的 *self*（T），有与之一一对应的自定义骨架方法集，记为 *self-S*（T）。因此，映射 f 可分为如下两步完成：①将请求中的操作名映射为对应的骨架方法；②骨架方法完成解码后，映射到实现方法，激活对应的实现方法。骨架方法到实现方法的映射依赖于实现环境所提供的支持，如面向对象编程语言提供的多态机制。因此，方法适配过程则主要取决于请求的操作名向对应骨架方法的映射，可简化为如下的一一映射 g。

$$g: \Psi(T) \rightarrow S(T)$$

2 适配框架

2.1 适配策略效率和定制时机

对于映射 g，集合 Ψ（T）、S（T）是一个静态集。对于静态集，目前有较多较为高效的策略。包括：二分法策略（Binary Search），策略算法实现简单，其平均比较次数为 O（$\log_2 n$），对于静态集具有最佳的空间效率。动态散列策略（Dynamic Hash），这种策略在最坏的情况下，其比较次数为 O（n），需要动态注册搜索集中的关键字，其空间效率取决于关键字集。二分发和动态散列策略的查找算法对于所有的关键字集是一致的。最佳散列策略（Perfect Hash）是静态搜索集在时间和空间效率上的最佳实现。然而，在实际应用中，搜索集的最佳散列函数的生成既费时又困难，因此通常使用非最小的最佳散列技术，这种技术将会占用较大的空间，导致频繁的调页，从而影响适配效率。另外，最佳散列策略对于不同的关键字集，会产生不同的较为复杂，包含有较长代码和表的最佳散列函数，不仅代码的重用性不好，而且对系统空间的要求较高。

上述策略对于不同的搜索集和不同应用环境，其时空效率不同，甚至在实际应用中理论上高效策略的效率在某些平台上不如其它较为低效的策略。单一的方法适配策略显然不能满足不同应用和环境的时空效率需求。另外，方法适配所需的数据可从接口描述中获得，并且应用的规模和特点、系统资源的限制、时空效率的要求，在设计时就可确定。因此，应用根据搜索集静态定制适配策略以达到时空效率的较好结合成为可能，并显得尤为重要。下面我们结合上述的分析给出框架和相关算法。

2.2 适配框架

基于灵活性和方法适配模型，我们建立了“框架 + 行为控制对象”形式的方法适配框架，如图 1 所示。其中，框架封装了方法适配基本过程的完成结构，定义了 Agent 行为的基本过程和基本信息，

包括基本方法适配的策略模板和骨架方法类以及相互之间的接口调用关系，框架中无具体的行为策略，如图 1（左）所示；行为控制对象封装了应用级数据，按照用户的需求控制方法适配行为的完成。行为控制对象既可继承相应的适配策略模板，也可不必继承相关的策略模板，直接实现对应接口中的方法，定制用户自定义的方法适配策略，如图 1（右）所示。适配算法如下：在适配框架中，对象的对应骨架类中包含了骨架方法对和方法适配方法 dispatch；对象适配器在完成对象适配后，激活对应对象实现骨架类的 dispatch 方法，该方法激活行为控制对象的 find 方法，使系统按照所定制的适配策略将操作名映射为对应骨架方法，骨架方法则通过解码、使用环境提供的功能激活实现方法，将输出参数编码返回，完成整个方法适配的过程。

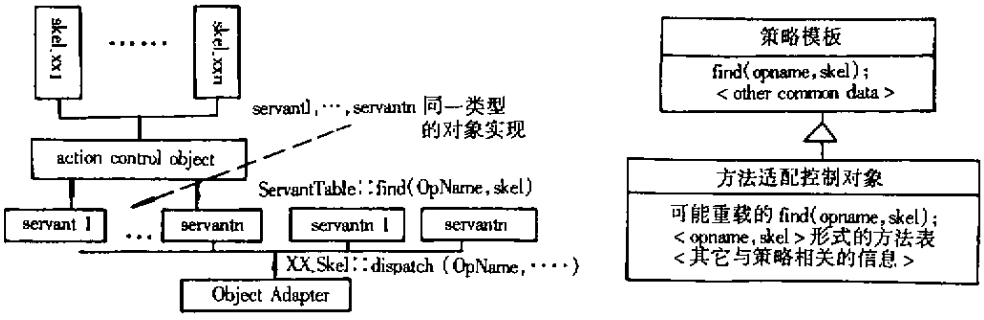

图 1　适配框架和行为控制对象

Fig.1　Framework for dispatching method and strategies pattern

行为控制对象作为应用级控制数据的封装被嵌入到对应框架中，控制方法适配的过程。对于一些典型策略，如二分法策略、动态散列策略等都有典型算法，在框架中提供了这些典型策略的策略模板，支持代码的重用；对于实现形式不一致的适配策略，注释预处理器将根据当前对象的 S（T）和适配策略生成对应行为控制对象的 find 方法的实现，如最佳散列策略；对于一些需要用户定制或添加的策略，则可直接实现对应方法，完成策略的定制。基于适配框架，方法适配策略是一种类级（class-level）的策略，可为每一类型的对象定义相应的适配策略。根据行为控制对象只读的特性，同一类型的对象可以共享一个行为控制对象，并且在并发模式下无需访问控制，如图 1 所示。

方法适配框架基于方法适配模型抽象了方法适配的共性，开放了方法适配的实现，易于集成多种适配策略，允许用户定制和修改适配策略，具有良好的可扩展性和灵活性。

2.3　分层算法

然而，随着应用复杂度和规模的增加，对象的继承变得更为复杂，对应 S（T）也会越来越大，造成了频繁地调页，影响方法适配的效率，尤其当很少一部分操作被请求的概率较大时。另外，较大的骨架方法表对系统的资源提出了较高的要求，不能满足某些系统资源的限制，如嵌入式系统。为满足时空效率的最佳结合，应对上述的框架和算法进行优化。本文基于方法适配模型，通过对骨架方法集 S（T）的合理分块，以减小搜索集。

基于骨架方法的定义和公式（1）和（2），有如下公式（3）成立：

$$S(super_i(T)) \subseteq S(T) \wedge self\text{-}S(T) \cap S(super_i(T)) = \Phi(1 \leqslant i \leqslant m) \quad (3)$$

由式（3）可推出对象可共享其父类对象的骨架方法。因此，基于适配框架可按照对象的继承层次对骨架方法集 S（T）进行自然分层。当前对象所对应的骨架方法表中只包含 self-S（T），对于 S（T）-self-S（T）中的方法，则共享其父类对象的方法表完成。由于方法表是按照适配策略组织的，所以方法表的共享还包含了适配策略的共享。分层算法适配过程如下：按照继承层次结构，逐层自底向上地调用对应父对象的行为控制对象的相关方法，直到适配出对应的骨架方法。例如：对于图 2（左）中的继承层次结构，基于上述算法则有图 2（右）中访问顺序，类型为 D 的对象通过各行为控制对象依次访问 D、B、C、A 的方法表，直到适配出对应的骨架方法。分层算法不要求对象的所有父对象都使

用分层结构。

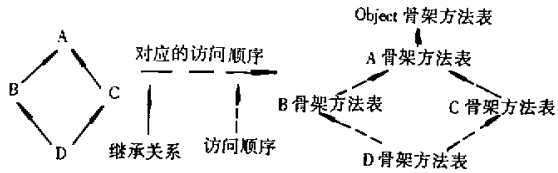

图2 对象的继承层次和对应的骨架方法表访问顺序

Fig.2 Object's inherit hierarchy and corresponding access hierarchy

对象通过共享其父类的骨架方法表减小了骨架方法表和重用了行为控制对象，不仅解决了环境限制的问题，而且对于方法集过大或者基于操作的调用频率定义接口中操作的应用具有较好的时空效率。分层算法还可解决遗留系统的封装，解决系统升级后原有后端形式与现有形式不匹配的问题。

3 方法适配行为的描述和实现

基于上述，适配策略和分层策略是一种静态可定制的策略。因此，应用需要一种描述对象方法适配语义的机制。为维护应用的可移植性和互操作性，简化应用的开发和维护，我们扩展 IDL 的注释来描述方法适配语义。

下面是基于 BNF 范式对方法适配语义的定义，如下所示：

< dispatching_policy_specification > ::= " /* " < interface_identifier > " Dispatching Policy ::" < dispatching_policy > " */"

< interface_identifier > ::= [< scoped_name >] < identifer >

< dispatching_policy > ::= " Linear_Search " | " Binary_search "
| " Dynamic_Hash " | " Perfect_hash " | " User_defined "

< method_table > ::= " /* " < interface_identifier > " Table ::" < table_policy > " */"

< table_policy > ::= " Part " | " Total "

< scoped_name > 和 < identifier > 的语法形式见 IDL 的语法规范[1]。

dispatching_policy_specification 用于描述对象的方法适配行为，interface_identifier 确定被描述的接口，dispatching_policy 确定方法适配所使用的策略，如 Linear_Search 表示定制线性搜索策略作为被描述接口所对应对象类型的适配策略，当集成了新的适配策略，其取值可增加；method_table 用于描述对象骨架方法表的组织形式，其中 table_policy 确定骨架方法表的组织形式，其值 Part 表示骨架方法表使用分层结构；Total 表示使用单一的骨架方法表，对应的骨架方法表包含对象的 S（T）。

为支持上述扩展 IDL 语言，IDL 编译器应集成相应的注释预处理器。注释预处理器以扩展 IDL 文件作为输入，将合法的注释描述转化为三元组 < 接口类型，适配策略，方法表组织策略 >。IDL 编译器在编译扩展 IDL 文时，将分别启动 IDL 编译器和注释预处理器进行 IDL 文件的前端分析和注释分析，分别生成前端分析结果和策略定制表。在注释处理器进行下一步处理前，必须与 IDL 编译器同步，等待编译器完成前端的分析，使用前端结果生成行为控制对象。IDL 编译器在生成 Skeleton 框架前，也必须等待注释预处理器完成策略定制表的建立，以便访问策略定制表，生成对

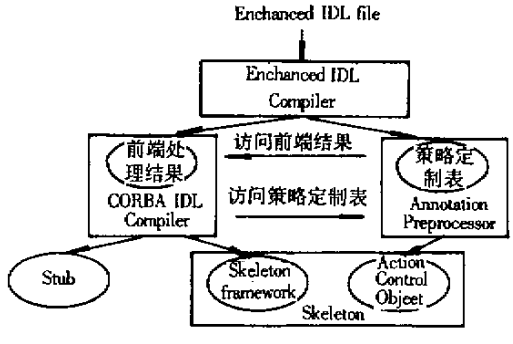

图3 扩展 IDL 编译器结构

Fig.3 Enchanced IDL compiler structure

应框架，如图 3 所示。IDL 编译器对未描述方法适配语义的接口使用缺省设置。编译后端形式请参见 StarBusv2.3 以上版本的编译结果。

4 扩展 IDL 机制的模板

通过扩展 IDL 的注释描述对象的适配行为为解决 CORBA 中类似的问题提供了一个设计模板。CORBA 规范未定义内部 API[1,5] 为该设计模板的实现提供了可能。在分布系统的设计过程中，对象的某些行为根据部署环境、应用需求和对象之间的交互约束就可确定，可通过 IDL 注释来说明，如本文中所分析的方法适配以及应用级访问同步控制、请求处理优先权的静态说明。在设计模板中，首先应确定对象行为约束关系的静态可确定性。根据这些行为抽象、设计描述语言的语法和语义，语法上应能与一般的注释区分，语义上应能表达这些行为的可能情况；其实现形式一般使用上述“框架 + 行为控制对象”的结构。模板的实现可完全参照图 3 的实现，使用一个预处理器处理注释，生成对应的控制对象，由标准 IDL 编译器生成框架。框架和控制对象完成用户对 Agent 行为的语义描述。

5 结论

本文研究了通过注释来增强 IDL 的描述能力，并且基于这种方法设计实现了一个方法适配策略的定制机制，支持应用以接口为粒度定制对象的方法适配策略，以达到较好的时空效率。本文还提供了一种基于扩展 IDL 解决类似问题的设计模板，该设计模板不会影响应用的移植性和互操作性，支持用户可直接描述对象的某些行为，使系统易于开发和维护。基于 StarBus 的实现，我们对方法适配的定制机制进行了测试。首先测试了各策略的基本效率，使用黑箱对比测试方法，系统中只有一个对象，操作无参数，操作数以 10 为步长递增，测定方法集对 1000 次请求的平均响应时间，结果如图 4。第二个测试基于最佳散列策略测试分层后的效率，结果如图 5。

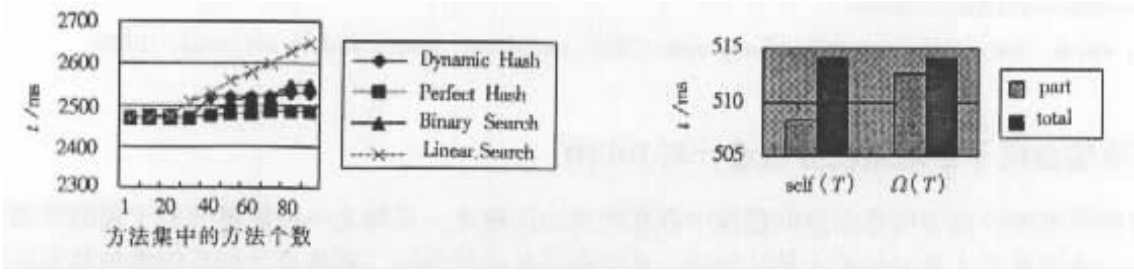

图 4 各策略效率比较

Fig.4 Efficiency of dispatching strategies

图 5 分层算法的效率

Fig.5 Efficiency of layered dispatching

参考文献：

[1] OMG. Common Object Request Broker：Architecture and Specification Revision2.2 [R]. February 1998.

[2] Puder A and Romer K. Mico – Mico is CORBA [M]. Morgan Kaufmann Publishers, 1998.

[3] IONA Technologies Ltd., Dublin, Ireland. Orbix programmer's Guide [M], 2. 0 edition, 1998.

[4] Inprise Corporation. VisiBroker for C + + Programmer's Guide [M]. 1998.

[5] Jacobsen H – A. Use extensible programming language interoperability with CORBA [C]. In CORBA Management Workshop, Dublin, Ireland, OMG, 22 September, 1997.