多粒度的面向对象软件估算模型的研究及应用

毛明志 陈 立
(中山大学信息科学与技术学院 广州 510275)

摘 要 软件估算方法学是软件工程学科中重要的研究方向,也是软件成本和进度控制的重要手段。面向对象作为现今的主流软件开发方法,对其估算方法的研究成为当前的热点。到目前为止,现有的面向对象估算方法大多是功能点方法的变种,这些方法对于统一的估算体系研究不足且估算的误差较大。针对上述问题,从经典估算方法入手,提出了一种多粒度的面向对象软件估算模型,并对模型的4个层次作了详细的描述,然后利用最小二乘法回归分析探讨了规模与工作量的关系,最后对实验结果做出了评估。

关键词 面向对象,面向对象估算模型,面向对象估算工具,回归分析

Multi-granularity Object-oriented Software Estimation Model

MAO Ming-zhi CHEN Li
(School of Information Science and Technology, SUN YAT-SEN University, Guangzhou 510275, China)

Abstract Software cost estimation is not only a significant researching aspect of software engineering, but also a good means to control software cost and schedule. Object-Oriented (OO), as a leading metric of software development, now its estimation is a hotspot of software cost estimation. Until now, most OO estimation methods are variations of FP method. These methods are lacking in uniform estimation system and have a comparative relative error. Aiming at the above problems, this paper presented a multi-granularity estimation model and gave a minute description according to classical metrics, then analyzed the relationship between the software size and effort by Ordinary Least-Squares (OLS) regression analysis. Finally this paper evaluated the result of experience.

Keywords Object-oriented, Object-oriented estimation model, Object-oriented estimation tool, Regression analysis

1 引言

软件估算是确定建造一个特定基于软件的系统或产品需要多少钱、多少工作量、多少资源以及多少时间的意图,即软件估算涉及到诸如成本、工作量、资源、进度和规模等[1]。软件估算是软件成本与进度控制的一个重要方面,在软件开发中占据举足轻重的地位。Robert L. Glass 在其《Facts and Fallacies of Software Engineering》(《软件工程的事实与谬误》,严亚军、龚波译)一书中曾经提到:造成软件项目失控最普遍的两个原因之一就是软件估算不足[2]。因此,无论是产业界还是学术界,越来越多的人认识到,做好软件成本估算是减少软件项目预算超支问题的首要措施之一,因为它不但直接有助于做出合理的投资、外包、竞标等商业决定,也有助于确定一些预算或进度方面的参考里程碑,使软件组织或管理者对软件开发过程进行监督,从而更合理地控制和管理软件质量、人员生产率和产品进度[3]。

从上个世纪90年代初以来,面向对象(Object-Oriented,简称OO)技术成为主流的软件开发方法,与此相对应,面向对象软件估算的方法和理论也获得了比较大的发展,它们大多是利用不同的估算依据来推测系统的规模与工作量。Fetcke[4],Uemura[5],Antoniol[6]的估算依据分别是用例图、序列图和类图,他们都设定了一组规则,把各自的图形映射到功能点的计算上。受到了功能点方法的启发,许多学者也提出许多类似于功能点的方法来估算面向对象系统。1993年,Karner 提出了一种用例点[7](Use Case Point,UCP)的估算方法,随后,Braz 对 UCP 方法提出了改进,取得了较好的效果[8]。2005年,Costagliola 提出了类点[9](Class Points,CP)的方法,建议充分使用类的信息来进行估算。其他著名的方法还包括对象点[10]、3D功能点[11]、预言性对象点[12]等。

尽管现有的估算方法大多已经成形并已经得到发展,但是仍然存在以下的不足:

(1)现有的估算技术大部分都是针对结构化编程而非针对面向对象技术的,例如上文提到的 Fetcke,Uemura,Antoniol 的方法,其本质上是属于功能点方法的,尽管这样可以充分利用功能点方法已经取得的研究成果,但使用为结构化编程而设计的功能点方法来估算面向对象系统会产生比较大的误差。

(2)统一的估算体系研究不足。目前大部分的研究只集中在某一方面,例如有些方法只集中于用例,有些方法只集中于类,更有些方法只集中于函数。这些方法没能从多角度进

到稿日期:2008-03-20 本文得到国家自然科学基金项目(No. 60773201)资助。
毛明志(1966—),男,副教授,研究方向是智能规划、软件度量和软件测试等,E-mail: Mcsmmz@mail.sysu.edu.cn;陈 立(1984—),男,硕士,研究方向是软件工程与CMM。

行估算考察，未能把握面向对象系统的总体特性。

2 多粒度的面向对象估算模型

在用例点[7]和类点[9]方法的基础上，结合一些传统的估算方法，建立以下多粒度估算模型：函数层的估算、类层的估算、模块层的估算和系统层的估算。估算模型如图 1 所示。

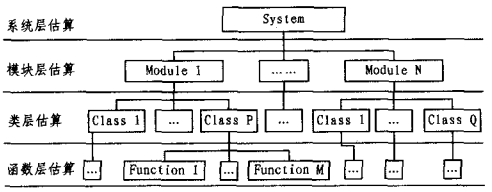

图 1 四层的面向对象估算模型

第一层是函数层，它是多粒度估算模型的基础，该层主要从程序员的角度出发，结合 Antoniol 的方法进行改进，对底层的函数的复杂性进行估算。第二层是类层，它建立在函数层的基础上，该层主要从软件设计师的角度出发，结合 Costagliola 类点的方法进行改进，对类的复杂性进行估算。第三层是模块层，它建立在类层的基础上。每一个模块可能由一个或多个类组成，所以该层主要从系统分析师的角度出发来考虑一群类，对模块的复杂性和规模进行估算。第四层是系统层，该层主要从项目经理的角度出发，结合 Karner 的 UCP 方法进行改进，提供系统级别的估算，帮助项目经理把握项目的进度。四层面向对象估算模型分别对应软件开发的不同层级，是一个逐渐递进的关系。

3 多粒度估算模型的描述

3.1 函数层

函数层的主要任务是估算类中每一个函数的复杂度，估算的依据是函数的参数和返回类型。首先定义两个概念[13]：

(1) 数据元素类型(Data Element Types，DETs)，指基本的数据类型，例如 bool，int，float，double 等。

(2) 记录元素类型(Record Element Types，RETs)，指复杂的数据类型，例如用户自定义的结构体和类等。

根据函数的参数列表和返回类型所包含的 DET 和 RET 的数量，可对 Antoniol 的函数复杂度评估矩阵[6]进行改进，引出表 1 来估算函数的复杂性。

表 1 函数的复杂性评估矩阵

	0～4 DET	5～10 DET	>10 DET
0～1 RET	低	中	高
2～3 RET	中	中	高
>3 RET	中	高	高

例如对于函数：public bool separate(structA a，classB b，int c，double d)。这个函数的 DET 和 FTR 的数量分别为 3 和 2，故复杂性为中等。

3.2 类层

类层的主要任务是估算系统中类的复杂度，估算的依据是类中含有的方法和属性。首先定义两个概念：

(1) 函数点(Method Points，MPs)。规定：一个简单的函数的 MP 为 1，一个中等复杂的函数的 MP 为 2，一个复杂的函数的 MP 为 3。

(2) 属性点(Attribute Points，APs)。规定：若类中的有一属性是 DET，则此属性的 AP 为 1；若类中的有一属性是 RET，则此属性的 AP 为 2。

根据一个类所含有的 MP 和 AP 的数量，可对 Costagliola 的类复杂度评估矩阵[9]进行改进，引出表 2 来估算类的复杂性。

表 2 类的复杂性评估矩阵

	0～7 AP	8～15 AP	≥16 AP
0～15 MP	低	低	中
15～30 MP	低	中	高
≥31 MP	中	高	高

一般来说，系统中的类还可以分为以下 4 种类型[9]：

(1) 业务逻辑类型(Problem Domain Type，PDT)，表示系统中用于描述真实世界实体的类。例如，FiledOfficer，EmergencyReport 就是典型的 PDT 类。

(2) 人机交互类型(Human Interaction Type，HIT)，表示系统中用于人机交互作用的类。例如，EmergencyReportForm，ReportEmergencyButton 就是典型的 HIT 类。

(3) 数据管理类型(Data Management Type，DMT)，表示系统中用于数据库查询和修改的类。例如，StoreReportSQL 就是典型的 DMT 类。

(4) 任务管理类型(Task Management Type，TMT)，表示系统中用于任务控制与管理的类。例如，ManageEmergencyControl，ReportEmergencyControl 就是典型的 DMT 类。

由此可引出表 3 来计算未调整的类点(Unadjusted Class Points，UnCP)。

表 3 UnCP 矩阵[9]

类型	描述	复杂性		
		低	中	高
PDH	Problem Domain	…＊3=…	…＊6=…	…＊10=…
HIT	Human Interaction	…＊4=…	…＊7=…	…＊12=…
DMT	Data Management	…＊5=…	…＊8=…	…＊13=…
TMT	Task Management	…＊4=…	…＊6=…	…＊9=…

3.3 模块层

由于每个模块可能由一个或多个类组成，因此模块层主要从一群类出发，估算系统中模块的规模和工作量，步骤如下：

(1) 计算模块的技术复杂度。技术复杂度因子如表 4 所列。

表 4 技术复杂性因子[9]

序号	描述	DI 值	序号	描述	DI 值
1	数据交换	＊＊＊	10	可重用性	＊＊＊
2	分布式功能	＊＊＊	11	易于安装	＊＊＊
3	响应或吞吐量性能	＊＊＊	12	易于操作	＊＊＊
4	高性能配置	＊＊＊	13	多站点开发	＊＊＊
5	传输速度	＊＊＊	14	易于修改	＊＊＊
6	在线数据输入	＊＊＊	15	用户适应性	＊＊＊
7	终端用户效率	＊＊＊	16	快速原型法	＊＊＊
8	在线更新	＊＊＊	17	多用户交互	＊＊＊
9	复杂的内部处理	＊＊＊	18	多重接口	＊＊＊

技术复杂性因子的值取在 0～5 之间，其相关程度随着取值的程度而增加，0 表示完全不相关，5 表示强烈相关。那么

$TFactor=(\sum_{i=1}^{18}DI_i*0.01)+0.64$。

(2) 对所有类的未调整类点求和，然后乘以模块技术复杂度，得到总规模：$Module\_Size=TFactor*\sum_{i=1}^{n}UnCP_i$，其中 n 为模块中类的数量。

(3) 估算此模块的工作量：$Module\_Effort=A*(Module\_Size)+B$，单位是人时。其中 $A=0.910$，$B=37.878$，这两个数是在下一节对数据库中实际项目的数据做回归分析得出的，可以通过实际情况进行校准。

3.4 系统层

系统层主要从项目经理的角度出发，结合传统的 UCP 方法进行改进，提供系统级别的估算，步骤如下：

(1) 计算技术复杂度(TCF)和环境复杂度(ECF)。技术复杂度因子和环境复杂度因子用来体现项目的非功能性需求对项目的影响，技术复杂度因子共含有 13 个，环境复杂度因子共含有 8 个，如表 5 和表 6 所列。

表 5 技术复杂度因子[7]

技术因子	描述	权重
T1	分布式系统	2
T2	响应或吞吐量性能	1
T3	终端用户效率	1
T4	复杂的内部处理	1
T5	可重用的	1
T6	易于安装	0.5
T7	易于使用	0.5
T8	可移植	2
T9	易于修改	1
T10	并发性	1
T11	特殊的安全特性	1
T12	提供对第三方的直接访问	1
T13	特殊的用户培训设施	1

表 6 环境复杂度因子[7]

环境因子	描述	权重
E1	UML 的熟悉程度	1.5
E2	应用程序开发经验	0.5
E3	面向对象编程经验	1
E4	领导分析能力	0.5
E5	积极性	1
E6	稳定的需求	2
E7	兼职工作人员	−1
E8	编程语言的难度	2

每个因子 T_i 与 E_i 的取值在 0～5 之间，0 表示完全不相关，5 表示强烈相关。若 W_i 表示权重，则 $TFactor=\sum_{i=1}^{13}W_i*T_i$，技术复杂度 $TCF=0.6+(0.01*TFactor)$；$EFactor=\sum_{i=1}^{8}W_i*E_i$，环境复杂度 $ECF=1.4+(-0.03*EFactor)$。

(2) 计算单个用例的未调整用例点。在计算系统中第 i 个用例的未调整用例点时，可把它拆分成以下 7 个小步骤。

a. 计算执行者复杂度。假设第 i 个用例中共有 n 个执行者，可根据它与系统交互数据的数量(如表 7 所列)判断第 j 个执行者的复杂度 CA_{ij}。

表 7 执行者复杂度分类[8]

复杂度	数据交换数量	值
简单	<=5	2
中等	6～10	4
复杂	>10	6

那么第 i 个用例执行者的复杂度之和 $TCA_i=\sum_{j=1}^{n}CA_{ij}$。

b. 计算前置条件复杂度。可根据用例前置条件的逻辑表达式数量(如表 8 所列)判断其复杂度 $CPrC$。

表 8 前置条件复杂度分类[8]

复杂度	逻辑表达式数量	值
简单	1	1
中等	2～3	2
复杂	>3	3

c. 计算主事务流复杂度。可根据用例的实体数量和主事务流数量之和(如表 9 所列)判断其复杂度 PCP_i。

表 9 主事务流、替代事务流复杂度分类[8]

复杂度	实体数+主事务流数量	值
非常简单	<=5	4
简单	6～10	6
中等	11～15	8
复杂	16～20	12
非常复杂	>20	16

d. 计算替代事务流复杂度。假设第 i 个用例中共有 n 个替代事务流，可根据第 c 步的计算方法(如表 9 所列)判断第 j 个替代事务流的复杂度 PCA_{ij}。那么第 i 个用例替代事务流复杂度之和 $TPCA_i=\sum_{j=1}^{n}PCA_{ij}$。

e. 计算异常事务流复杂度。假设第 i 个用例中共有 n 个异常事务流，可根据测试逻辑表达式的数量(如表 10 所列)判断第 j 个异常事务流的复杂度 CE_{ij}。那么第 i 个用例异常事务流复杂度之和 $TPE_i=\sum_{j=1}^{n}CE_{ij}$。

表 10 异常事务流复杂度分类[8]

复杂度	测试逻辑表达式数量	值
简单	1	1
中等	2～3	2
复杂	>3	3

f. 计算后置条件复杂度。可根据用例后置条件的实体数量(如表 11 所列)判断其复杂度 $CPoC_i$。

表 11 后置条件复杂度分类[8]

复杂度	实体数量	值
简单	<=3	1
中等	4～6	2
复杂	>6	3

g. 求和。综合以上 6 步的结果，可得到第 i 个用例的未调整用例点 $UUSP_i=TPA_i+CPrC_i+PCP_i+TPCA_i+TPE_i+CPoC_i$

(3) 计算全部用例的未调整基本工作量。假设待估算项目共有 m 个用例，重复步骤(2)可分别计算出这 m 个用例的未调整用例点，那么此项目所有用例的调整用例点为：$System\_Size=\sum_{i=1}^{m}(UUSP_i*TCF_i*ECF_i)$，其中 TCF_i 和 ECF_i 是系统中第 i 个用例的技术和环境复杂度。

(4) 计算系统的工作量：$System\_Effort=C*(System\_Size)+D$，单位是人时。其中，$C=1.814$；$B=1006.179$。这两个数是在下一节中对数据库中实际项目的数据做回归分析得出的，可以通过实际情况校准。

4 估算模型规模与工作量关系

在模块层和系统层中已经得出模块和系统的规模，那么如何将规模转化为工作量呢？为了研究规模与工作量之间的关系，本文从实际的项目数据出发，利用最小二乘法回归分析来研究它们之间的关系。

4.1 模块层规模与工作量的关系

表 12 是从 G 公司中提取的 40 个模块的数据，表 13 是利用 SPSS 对表 12 做最小二乘法回归分析得出的结果，图 2 是预测的模型图。

表 12 40 个模块的数据

ID	Module_Effort	Module_Size	ID	Module_Effort	Module_Size
1	47	18.43	21	61	43.79
2	66	40.42	22	157	104.60
3	78	74.43	23	80	59.77
4	169	126.83	24	135	98.40
5	202	207.10	25	114	71.36
6	43	30.14	26	106	46.81
7	165	107.60	27	300	286.54
8	92	34.76	28	61	17.42
9	166	150.83	29	73	26.11
10	30	15.84	30	81	41.16
11	80	41.93	31	80	40.23
12	180	127.71	32	80	68.85
13	34	10.77	33	143	123.12
14	141	103.35	34	69	39.01
15	140	123.92	35	44	33.06
16	235	224.23	36	78	43.84
17	46	12.38	37	72	21.06
18	103	69.78	38	71	24.72
19	100	79.16	39	72	33.35
20	113	73.15	40	59	18.45

表 13 统计结果

	Value	Std. Err	t-value	p-value
Coefficient	0.910	0.040	22.807	0.000
Intercept	37.878	3.789	9.996	0.000

Prediction Model	R^2	R	Std. Err	F	Singif F
Module_Effort = 0.910 * Module_Size + 37.878	0.932	0.965	15.380	520.145	0.000

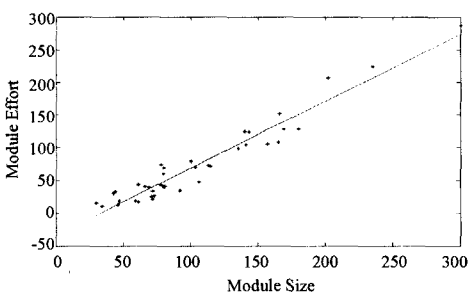

图 2 模块规模与工作量的预测模型图

4.2 系统层规模与工作量的关系

表 14 是从 G 公司中提取的 10 个系统的数据，表 15 是利用 SPSS 对表 14 做最小二乘法回归分析得出的结果，图 3 是预测的模型图。

表 14 10 个系统的数据

ID	System_Effort	System_Size	ID	System_Effort	System_Size
1	2560	967.34	6	4085	1495.42
2	3786	1669.24	7	4294	1936.91
3	4526	1858.62	8	5195	2402.53
4	5264	2205.45	9	2475	882.21
5	3722	1327.61	10	4933	2218.95

表 15-a 统计结果(A)

	Value	Std. Err	t-value	p-value
Coefficient	1.814	0.163	11.149	0.000
Intercept	1006.179	287.715	3.497	0.000

表 15-b 统计结果(B)

Prediction Model	R^2	R	Std. Err	F	Singif F
System_Effort = 1.814 * System_Size + 1006.179	0.940	0.969	256.584	124.291	0.000

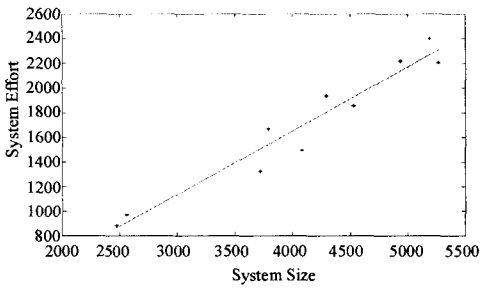

图 3 系统规模与工作量的预测模型图

5 实验结果

为了验证多粒度估算模型的有效性，本文额外选取了 G 公司中 20 个模块和 10 个系统的数据来研究和验证，得出的结果如表 16-a 和表 16-b 所示。

表 16-a 模块层实验结果

	$Module\_Effort_{real}$	Module_Effort = 0.910 * Module_Size + 37.878		
		Module_Size	$Module\_Effort_{pred}$	MRE
1	42	14.43	51.00	0.21
2	76	74.43	105.61	0.39
3	198	202.10	221.79	0.12
4	169	102.60	131.24	0.22
5	165	150.83	175.14	0.06
6	177	124.71	151.37	0.14
7	137	101.35	130.11	0.05
8	233	224.23	241.93	0.04
9	105	65.78	97.73	0.07
10	108	73.15	104.44	0.03
11	152	101.60	130.33	0.14
12	132	97.40	126.52	0.04
13	108	44.81	78.65	0.27
14	57	16.42	52.82	0.07
15	76	37.16	71.69	0.06
16	139	123.12	149.91	0.08
17	50	31.06	66.14	0.32
18	68	18.06	54.32	0.20
19	70	31.35	66.41	0.05
20	59	16.45	52.84	0.10
		MMRE		0.13
		PRED(0.25)		0.85

表 16-b 系统层实验结果

	$System\_Effort_{real}$	System_Effort = 1.814 * System_Size + 1006.179		
		System_Size	$System\_Effort_{pred}$	MRE
1	3268	1649.07	3997.59	0.22
2	3752	1338.37	3433.98	0.08
3	2106	807.02	2470.11	0.17
4	5352	1975.03	4588.88	0.14
5	2345	862.35	2570.48	0.10
6	1668	698.58	2273.40	0.36
7	2092	947.57	2725.07	0.30
8	4311	1685.08	4062.91	0.06
9	2392	1416.85	3576.34	0.22
10	2543	1382.55	3514.12	0.38
		MMRE		0.23
		PRED(0.25)		0.70

由实验的结果得出以下结论：(1) 模块层工作量的估算值平均相对误差为 0.13，被估算的 20 个模块中相对误差在 0.25 以内的有 17 个，占 85%；(2) 系统层工作量的估算值平均相对误差为 0.23，被估算的 10 个系统中相对误差在 0.25 以内的有 7 个，占 70%。

结束语 面向对象估算方法的研究从 20 世纪 80 年代开始，经过多年的研究与发展，理论上取得了比较大的进步，但是也存在一些比较大的缺点，例如统一的估算体系研究不足以及估算误差较大等。本文在传统面向对象软件估算方法的基础上，提出了一种多粒度的面向对象软件估算模型并对模型的 4 个层次作了详细的描述，然后利用最小二乘法回归分析探讨了规模与工作量的关系。最后通过实验数据的验证，证明了此估算模型是有效的。

参 考 文 献

[1] Pressman R S. Software Engineering : A Practitioner ' s Approach. Fifth Edition. McGraw-Hill, 2001

[2] Glass R L. Facts and Fallacies of Software Engineering. Boston: Addison-Wesley, 2003

[3] LI Ming-shu, HE Mei, YANG Da, et al. Software Cost Estimation Method and Application. Journal of Software, 2007, 18(4): 775-795

[4] Fetcke T, Abran A, Nguyen T-H. Mapping the OO-Jacobson approach into function point analysis // Proceedings of IFPUG 1997 Spring Conference. 1997: 134-142

[5] Uemura T, Kusumoto S, Inoue K. Function-point analysis using design specifications based on the unified modelling language. Journal of Software Maintenance Evolution-Research Practice, 2001, 13: 223-243

[6] Antoniol G , Fiutem R , Lokan C . Object - oriented function - points: an empirical validation. Empirical Software Engineering, 2003, 8: 225-254

[7] Karner G. Resource estimation for objectory projects. Objective Systems SF AB, 1993

[8] Braz M R, Vergilio S R. Software effort estimation based on use cases // Proceedings of the 30th Annual International Computer Software and Applications Conference

[9] Costagliola G, Tortora G. Class Point: An Approach for the Size Estimation of Object-Oriented Systems. IEEE Transactions on Software Engineering, 2005, 31(1)

[10] Banker RD, Kauffman RJ, Kumar R. An empirical test of object-based output measurement metrics in a computer aided software engineering environment. Journal of Management Information Systems, 1991-1992, 8(3): 127-150

[11] Whitmire S A. 3D Function Points: Applications for Object-Oriented Software // Proc. ASM'96 Conf. 1996

[12] Minkiewicz A F. Measuring Object Oriented Software with Predictive Object Points // Proc. Conf. Applications in Software Measurements (ASM'97). 1997

[13] Zivkovic A, Rozman I, Hericko M. Automated software size estimation based on function points using UML models. Information and Software Technology, 2005, 47: 881-890

(上接第 281 页)

的影响。此外，这些研究往往依赖于特定的分布对象平台，缺乏对于分布对象应用的通用性支持，因而限制了这类研究在遗留系统中的应用。

我们的方法采用的混合型监控模式与 OGS[3] 类似。本文的方法与 OGS 关键的区别之一在于我们按照域模型对分布对象环境进行了划分，并通过部署层次型的故障检测服务，实现了对域内不同粒度实体的故障检测，而 OGS 仅能提供对象级的故障检测功能。与 OGS 的另一区别是，本文按照功能角色的划分，明确提出了一种新的故障提供者-监控者-使用者概念模型。

FTBUS[4] 的设计思想在采用层次型故障检测机制与我们的方法类似，但它未采用混合型故障监控模式，同时引入 CORBA 事件服务来传播故障信息，相应地，对 CORBA 平台的依赖过强，可能导致单点故障。

结束语 故障检测对于分布对象环境中的容错系统具有非常重要的作用。然而，已有的系统和研究工作大多集中于对故障恢复问题的研究，较少考虑故障检测问题，而且多数系统采用集中式的故障检测方法，机制上普遍比较单一，因此故障检测服务在伸缩性、扩展性和灵活性方面往往存在问题。本文在基本的故障提供者-监控者-使用者概念模型基础上，引入了域的概念，将整个分布对象环境划分为若干域。在域内，是集中式的层次型故障检测方法，采用了混合型的故障监控模式，弥补主动和被动故障监控方法的不足；引入异步消息通知机制实现了故障信息的及时传播；层次型故障检测部署方式则能对不同粒度实体执行故障检测。

目前，我们提出的故障检测服务是适用于同一域内不同粒度实体的层次型故障检测方法，并未考虑域之间的全局范围内的故障信息传播问题。因此，下一步，我们打算引入更加灵活的移动 Agent 模型，实现全局范围内的故障信息传播。

参 考 文 献

[1] Tanenbaum A S. Distributed Operating System. Prentice Hall Inc., 1995

[2] Natarajan B, Gokhale A, Yajnik S, et al. DOORS: Towards High-performance Fault Tolerance CORBA // Proceedings of the 2nd Distributed Applications and Objects (DOA) Conference. Antwerp, Belgium, Sep. 2000: 21-23

[3] Felber P. The CORBA Object Service : a Service Approach to Object Groups in CORBA. PhD thesis. Switzerland: Swiss Federal Institute of Technology at Lausanne, 1998

[4] 周明辉. 面向对象的容错中间件的研究与实现. 博士学位论文. 国防科技大学, 2002

[5] Alvisi L, Malkhi D, Pierce E, et al. Dynamic Byzantine Quorum Failure Systems // Proceedings of the International Conference on Dependable Systems and Networks. June 2000: 283-292

[6] Schonwalder J, Garg S, Huang Y, et al. A Management Interface for Distributed Fault Tolerance CORBA Services // Proceedings of the IEEE 3rd International Workshop on System Management. Newport, RI, Apr. 1998