中国科学 E 辑：信息科学 2008 年 第 38 卷 第 6 期：843 ~ 863
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE IN CHINA PRESS

# 从基于知识的软件工程到基于知件的软件工程

陆汝钤[①②], 金芝[①②]

① 中国科学院数学与系统科学研究院, 北京 100080;
② 中国科学院计算技术研究所, 北京 100080
E-mail: rqlu@math.ac.cn, zhijin@math.ac.cn

收稿日期: 2008-01-09; 接受日期: 2008-03-02
国家自然科学基金(批准号: 60496324, 60603002, 60736011)和国家杰出青年基金(批准号: 60625204)资助项目

**摘要** 首先回顾了从 90 年代初开始的 10 年 PROMIS 技术研究. PROMIS 的核心是基于领域知识建模的应用系统自动开发, 它强调要把领域知识的开发与应用和系统架构本身的开发区分开来并分别进行. 但是, 当时并没有确定这些单独分出来的领域知识应当采取何种形式. 在随后的研究中, 找到了这样的形式, 这就是知件. 知件是领域知识的商品化形式. 简述知件、知识中间件和知件工程的基本定义, 简述知件工程的 3 个生命周期模型, 然后设计这 3 个生命周期模型的知件化实现, 并推广到 PROMIS 系统本身的知件化实现, 其中体现了全部 3 个生命周期模型. 进一步讨论了 J2EE 背景下的领域知识建模和应用系统自动开发问题, 把 PROMIS 技术、知件技术和 J2EE 技术三者结合起来, 提出了 PROMIS/KW**的应用系统开发和部署框架, 并研究了其中的相关技术问题.

**关键词**
PROMIS
基于知识的软件工程
知件
J2EE 技术
PROMIS/KW**
基于知件的软件工程

## 1 基于领域建模和类自然语言技术的应用系统自动生成

软件工程的实践告诉我们, 软件开发失败的原因往往在于需求分析没有做好. 而需求分析没有做好的原因又往往在于用户和软件工程师之间缺乏良好的合作. 用户往往不能确切而清楚地说明他的需求, 甚至还要保留在软件开发过程中随时修改需求的自由. 这一来, 软件工程师就很难做出恰当的需求分析, 从而也就很难保证软件开发任务的顺利完成.

目前软件工作者在需求分析中研究和使用的主要方法是形式化或半形式化方法. 这些方法使用了不同的需求说明语言. 形式化方法的优点是它能引导软件工程师或程序员按照严格

的步骤写出需求说明, 但使用任何形式化方法的前提是已经确切和完整地收集到了用户的需求信息. 如果做不到这一点, 则不管形式化方法在理论上如何严密, 它是不能帮助我们完成软件开发任务的. 也就是说, 形式化方法不能解决我们在上面提到的用户不能很好合作的问题. 针对这个问题而提出的一个解决方案是: 尽可能多地把用户吸引到软件开发过程中来, 使用户能尽早发现正在研制中的软件与他的需求之间的偏差, 甚至是发现已经写好的需求说明和他的真实需求之间的偏差, 从而及时修改需求说明和设计. 但是, 形式化方法很难被直接用作软件工程师和用户之间合作的基础, 形式化的需求说明语言也很难在这两部分人中间起沟通作用, 因为他们的知识背景太不相同了. 更何况在研制和运行软件的过程中, 需求信息可能改变, 修改设计或重新设计都会给这两部分人带来沉重负担.

我们认为, 不仅要把用户吸引到软件开发过程中来, 而且在可能情形下, 最好是把开发软件的钥匙交到用户手里, 让用户自己来定义、设计、开发、维护和修改他的软件. 这对某些特定类型的软件, 例如管理信息系统之类, 是可能的. 为达此目的, 必须免去用户学习和掌握软件开发知识的负担, 还要免去用户按形式化方法做需求分析的负担. 而做到这一点的关键是以强大的知识支持作为后盾. 为了实现这个目标, 我们在多年前提出了一种基于知识的软件开发方法KISSME(knowledge intensive software system manufacture engineering), 据此开发的技术称为PROMIS(PROtotyping MIS)[1~14], 这个方法的要点是以一个大容量的知识库来支持软件的开发, 使用户不必自己掌握软件开发知识, 也不必自己掌握有关应用领域的需求分析知识. 该方法的另一个要点是提供一种面向领域用户的类自然语言BIDL(business information description language)[1~3]. 该语言非常接近领域用户常用的业务用语(因此称为类自然语言), 而不含软件技术用语. 用户只需把本单位的组织、人员和业务情形用BIDL语言描述清楚, 计算机即可在上述知识库的支持下编译BIDL说明, 并为用户自动生成一个他所需要的软件.

设计一个类自然语言形式的 BIDL 语言大致可按如下步骤进行:

1. 针对某个应用领域, 选定一组基本语义元素. 例如, 如果该领域是供应链管理, 则基本语义元素应该是一组描述供应链管理中涉及的仓储、采购、制造、物流、服务、核算等各个环节的元素, 例如, “规定最少存货量”就是仓储的一个基本语义元素;

2. 选定一个自然语言, 例如中文, 作为待设计的类自然语言的背景语言;

3. 对于每一个基本语义元素, 选定该自然语言中至少一个语句模式与之对应. 例如, 对语义元素“规定最少存货量”可选定语句模式“*库存不应少于*”, 它可以生成例如“焦炭库存不应少于 5 万吨”的语句;

4. 这些语句模式所能生成的语句全体称为类自然语言, 把所有语句模式组织成一个语法, 称为该类自然语言的关键结构;

5. 计算机对类自然语言程序进行分析时, 只分析其关键结构并提取语义, 凡是与*号对应的部分视为参数, 不进一步分析.

这种知识自动转换(或自动编译)的原理还需要两项技术的补充, 才能做到比较完备. 第 1 种技术是逐步求精, 即不断用实际的例子去考核和测试知识库, 使知识库能够更加准确地反映客观世界. 第 2 种技术是把从多个来源(多本教科书, 或多批技术资料, 或多位专家, 以及它

们的组合)获取的知识综合起来, 甚至融合成一体.

经过了将近10年的研究和开发, 上述思想逐步完善, 开发了PROMIS系统[11]. 它的基本结构如图 1 所示. 其中DKDL(domain knowledge definition language)是类自然的领域知识描述语言, 领域专家用此语言写下有关该领域的知识; DOKB(domain ontology knowledge base)是基于本体的领域知识库, ONONET(ONtology-Object-NET)是一个领域本体描述语言, 是DOKB中领域知识的表示形式, 由(许多)DKDL程序经过编译和融合而成[12,13]; BIDL(business information description language)是一个企业情形描述语言, 领域用户用它描述本企业的基本情形[1~3]; NEWCOM是一个应用系统体系结构描述语言, 针对客户/服务器两层结构给出该应用系统在网上的分布[8].

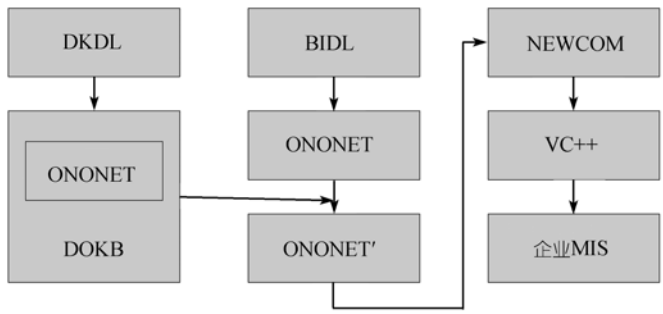


**图 1　PROMIS 系统基本结构**

PROMIS 系统的运行过程大致分为两个并行的进程. 一个是领域知识获取进程, 在此进程中领域专家不断地以 DKDL 形式给出领域知识, 并以 ONONET 的本体形式存于知识库中. 另一个是应用系统自动生成进程, 在此进程中领域用户以 BIDL 形式给出他的企业的基本情形, 由 BIDL 编译程序加工成 ONONET 形式的企业本体, 一个 ONONET 分析程序利用领域知识库 DOKB 对之进行分析, 并用 DOKB 中的知识对之进行补充, 使之在领域知识库意义下完备化, 完备后的领域本体称为 ONONET′. 然后再利用信息系统知识和网络知识把 ONONET 转换为 NEWCOM 形式的应用系统体系结构描述, 再往下就可以直接转换成高级语言编程了, 其中和数据库的接口通过 ODBC 完成.

根据这个技术路线实现的系统也称为《天鹰》. 它可以帮助人们快速地构造各行各业的管理信息系统. 根据用户的需要, 有如下 3 种版本可供使用.

《小天鹰》由一个应用系统自动生成器和一个应用系统知识库构成. 用户用 BIDL 语言写清楚本单位的组织、人员和业务情形后,《小天鹰》即可为他自动生成应用系统.《小天鹰》适合于不懂计算机软件, 但熟悉本行业业务情形的人使用.

《中天鹰》由一个应用系统自动生成器、一个应用系统知识库和一个领域知识库构成. 用户可以像使用《小天鹰》那样使用《中天鹰》, 也可以只指明本单位属于哪种类型的企业(例如三星级宾馆),《中天鹰》即可利用领域知识库的知识为他自动生成应用系统, 用户再在此基础上稍加修改就成了适合本单位使用的软件.《中天鹰》适合于不懂计算机软件, 也不太熟悉

本行业业务情形的人使用. 它的领域知识库是独立于应用系统知识库的, 用户可以根据自己的需要配备, 因此,《天鹰》软件是组合式的.

《大天鹰》由一个领域知识分析和建模器、应用系统自动生成器、一个应用系统知识库和一组领域知识库构成.《大天鹰》除具有《中天鹰》的全部功能外, 还可以用来对一个应用领域作分析, 建立新的领域知识库.《大天鹰》适合专业软件开发人员使用.

无论是在软件开发方法学方面, 还是在软件产业发展方面,《天鹰》项目的实践都给了我们很多启示. 总结起来有 3 点. 第 1 是要在开发应用软件时尽可能把软件知识和领域知识区分开来. 第 2 是在组建开发队伍时要把软件开发人员和领域知识开发人员区分开来. 第 3 是在发展产业时要把软件产业和知识产业区分开来.

一个应用软件往往含有大量领域知识. 这决定了在开发应用软件时领域专家和软件工程师必须很好合作. 由知识背景不同而造成的文化壁垒使这种合作不容易成功. 如果我们能让软件工程师和领域专家各司其职, 软件工程师只把软件工具做好, 总结和开发领域知识的事则交给领域专家去做, 然后用适当的技术把两者结合起来, 就可以达到扬长避短的目的. 用户从软件开发者手里购买软件工具, 而从知识开发者那里购买知识模块. 就好像微机用户从 IBM 公司那里购买硬件, 而从兼容软件开发商那里购买微机的配套软件一样. 我们相信, 循这条路发展, 必能使软件产业和知识产业同时发展成两个独立而繁荣的产业.

KISSME 方法带来的另一个好处就是它为解决软件工程中的“演化”难题提供了一个可行的解题思路. 有些软件, 如管理信息系统, 是不可能不演化的, 因为企业情形随时在变, 当然它的管理信息系统也得跟着变. 任你需求说明写得再好也不能完全解决问题. KISSME 方法把开发和修改软件的钥匙都交给了用户, 从而用户不必为经常需要修改软件而去召唤软件工程师, 他可以随时根据变化了的企业情形自己动手修改以 BIDL 形式书写的需求说明, 让系统为他生成一个新的管理信息系统.

## 2 知件和知件工程

如上所述,《天鹰》的特点是要把软件的开发和软件中所含知识的开发区分开来. 但是在《天鹰》的研究中并没有对这部分区分开来的知识给予定位: 知识的开发应该采取什么方式? 开发出来的知识产品又应该呈现为什么形式? 科学家和工程师们在软件和软件工程方面的丰富研究成果是否也适用于知识开发和知识产品? 如果不完全适合, 那么它们的区别在什么地方? 特别地, 既然软件是一种商品, 那么知识产品的商品化形式是什么? 对这一系列问题的研究引导我们得出了知件的概念[15~20]. 知件是独立的、计算机可操作的、商品化的、符合某种工业标准的、有完备文档的、可被某一类软件或硬件访问的知识模块. 知件和软件的主要区别如表 1 所示.

通过知件的形式, 我们可以把软件中的知识含量分离出来, 使软件和知件成为两种不同的研究对象和两种不同的商品, 使硬件、软件和知件在 IT 产业中三足鼎立. 明确地说, 知件和软件的主要区别如表 1 所示. 专家系统和通常的知识库都在某些方面类似于知件, 但是它们都不是知件. 专家系统是传统意义上的软件, 因为它包括以推理程序为核心的一系列应用程序模块. 通常的知识库也不是知件. 首先是因为它至少包含一个知识库管理程序, 从而不满足知

件的基本条件(只含知识). 其次是因为这些知识库的知识表示和界面一般不是标准化的, 难以用即插即用方式和任意的软件模块组合使用. 而且一般的知识库还没有商品化. 我们的目的是使知件成为一种标准的部件, 更换知件就像更换计算机上的插件一样方便.

**表 1 知件和(传统意义上的)软件的主要区别**

| | (传统意义上的)软件 | 知件 |
|---|---|---|
| 技术内容 | 既含领域知识, 又含软件技术 | 只含领域知识 |
| 可操作性 | 可以独立运行 | 不能独立运行, 只用于支撑软件运行 |
| 生命周期 | 主要取决于用户需求的变化 | 主要取决于知识积累和发展 |
| 种类划分 | 习惯上分为系统软件和应用软件两类 | 只有应用知件, “系统知件”是一种软件 |
| 开发人员 | 以专业的软件工程师为主(有时需要领域专家的支持) | 以各行各业的领域专家为主(有时需要知识工程师的支持), 一般无需专业队伍 |
| 知识产权 | 软件著作权 | 知识专利权 |

对软件开发过程施以科学化和工程化的管理, 就形成了软件工程. 类似地, 对知件开发过程施以科学化和工程化的管理, 就形成了知件工程. 两者有某些共同之处, 但也有很多不同. 计算机发现知识, 或计算机和人合作发现知识已经成为一种产业: 知识产业. 而如果计算机生成的是规范化的、包装好的、商品化的知识, 即知件, 那么这个生成过程(包括维护、使用)涉及的全部技术之总和可以称之为知件工程. 它与软件工程既有共同之点, 也有许多不同的地方, 从某种意义上可以说, 知件工程是商品化和大规模生产形式的知识工程.

在对软件工程的研究中, 学者们提出了多个不同的开发模型, 它们与软件的生命周期密切有关. 知件工程也有自己的开发模型和生命周期. 并且, 与软件工程一样, 知件工程的开发模型和生命周期也不会只有一种, 因为知识的获取、加工和应用本来就有多种不同的模式.

根据知识获取和建模的 3 种不同方式, 我们为知件工程建议 3 种开发模型.

第 1 种是熔炉模型, 它适用于存在着可以批量获取知识的知识来源的情形. 回顾 PROMIS 获取领域知识的机制. 在那里我们采用类自然语言理解(PNLU)技术, 让计算机把整本教科书或整批技术资料自动地转换为一个知识库, 也可以把一个专家的谈话记录自动地转换为知识库. 这个知识库就称为熔炉. 由于成批资料中所含的知识必须分解成知识元后在知识库中重新组织. 特别是当这些知识来自多个来源(多本教科书, 或多批技术资料, 或多位专家, 以及它们的组合)时, 更需要把获取的知识综合起来, 这种重新组织的过程就是知识熔炼的过程. 我们把熔炉中的知识称为知识浆. 熔炉模型的基本结构如图 2 所示.

第 2 种称为结晶模型, 它适用于从分散的知识资源中提取和凝聚知识. 结晶模型的基本构思是: 在知件的整个生命周期中, 新的、有用的知识是不断积累的, 它需要一个获取、提炼、分析、融合、重组的过程. 从这个观点看, 我们周围的环境更像是一种稀释了的知识溶液, 提取知识的过程就像是一个结晶过程. 由于其规模之大, 我们称它为知识海. 而熔炉模型中的知识浆则是浓缩了的知识溶液. 对知识的需求就像一个结晶中心, 围绕这个中心, 海里的知识不断析出并向它聚集, 使结晶越来越大. 知识晶体的结构就是知识表示和知识组织的规范. 蕴含于因特网上的知识就是一种典型的知识海.

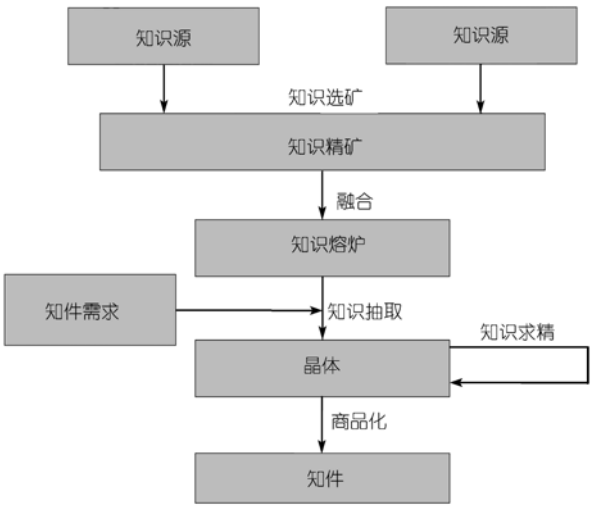


**图 2　知件工程的熔炉模型**

我们需要两个控制机制来控制知识晶体的形成和更新过程. 第 1 个机制称为知识泵. 它的任务是从分散的知识源中提取并凝聚知识. 上面提到的类自然语言就是这样的一种知识泵. 它不仅可以控制知识析取的内容, 还可以控制知识析取的粒度. 已经获得的知识晶体可以作为新的知识颗粒进入知识海中, 以便在高一级的水平上重用. 类自然语言的使用在某种程度上体现了知识结晶的方式. 第 2 个机制称为知识肾. 由于知识是会老化和过时的, 旧的、过时的知识不断被淘汰, 它表现为结晶的风化和蒸发. 知识肾的任务是综合分析新来的和原有的知识, 排除老化、过时和不可靠的知识, 促进知识晶体的新陈代谢.

综合这两者, 知识泵和知识肾合作完成知识晶体的知识析取、知识融合和知识重组. 知件的演化有赖于作为它的基础的知识晶体的演化和更新. 从理论上说, 这是一个无穷的过程. 结晶模型如图 3 所示.

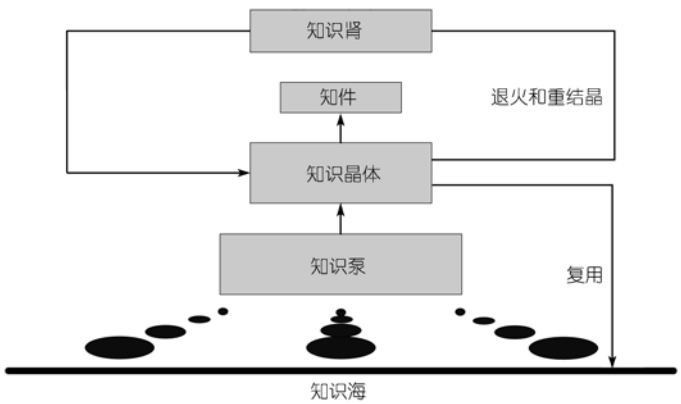


**图 3　知件工程的结晶模型**

结晶模型的典型例子是第 2 代浏览器, 这类浏览器不但具有广泛搜集网页的功能, 而且具有从网页中提取知识, 并综合从海量网页中获取的知识, 形成知识晶体, 最后还可以生成知件, 或以其他方式出版这些知识的功能, 例如提供某一领域的发展动态报告, 或某一领域的百科辞书.

第 3 种模型借用Nonaka 和Takeuchi在 1995 年作为知识的创建过程提出的螺旋模型概念[21], 它适用于获取通过反复实践积累起来的经验知识. 该模型反映了学术界区分显知识和隐知识的观点, 认为知识创建的过程体现为显知识和隐知识的不断互相转换, 螺旋上升. 它包括如下 4 个阶段: 外化(通过建模等手段使隐知识变为显知识)、组合(显知识的系统化)、内化(运用显知识积累新的隐知识)和社会化(交流和共享隐知识). 我们认为这个模型可以用来描述经验知识的形成过程, 它也是形成知识晶体的一个渠道(见图 4).

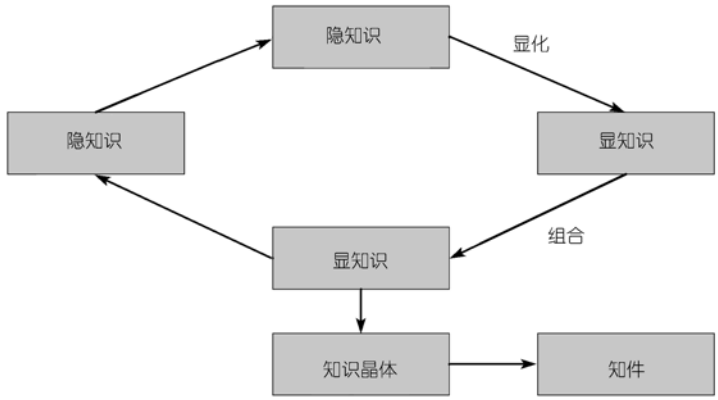


**图 4　知件工程的螺旋模型**

我们把 3 种知件工程模型生成的知识模块统称为知识晶体. 从应用的角度看, 知识晶体还只是一个半成品, 需要经过进一步的加工才能成为知件. 知识晶体和知件的主要区别如表 2 所示.

**表 2　知识晶体和知件的区别**

| | 知识晶体 | 知件 |
|---|---|---|
| 包含原则 | 某一领域或问题范围内知识自身的相对完整性 | 某一应用需求指导下知识使用的相对完整性 |
| 划分原则 | 不同的领域或问题的知识基本上组织成不同的晶体 | 多个不同的晶体可以组成一个统一的知件 |
| 表示原则 | 以说明性表示为主 | 以过程性表示为主 |
| 界面原则 | 强调科学性 | 强调规范性和使用方便性 |
| 生产原则 | 由知识的发展和积累所驱动 | 由市场需求所驱动 |
| 产权原则 | 不一定有知识产权 | 一定有知识产权 |
| 更新原则 | 通过知识晶体自身的演化, 包括聚晶、风化、蒸发、退火、再结晶 | 由演化后的知识晶体重新生成知件新版本 |
| 更新周期 | 可以不间断地连续进行 | 相对稳定. 知识晶体演化积累到一定程度时再更新知件 |

我们在前面说到不存在系统知件, “系统知件”即是软件. 我们把起“系统知件”作用的软件

称为知识中间件. 知识中间件是能够全面实现上面提到的知件生命周期的一大类软件工具. 表 3 列出了它们的主要功能及分类.

**表 3　知识中间件分类**

| 类别 | 功能 | 主要技术举例 |
|---|---|---|
| 知识编译中间件 | 把文字知识源批量转化为可操作的知识 | PNLU、知识抽取、知识融合 |
| 知识析取中间件 | 从知识溶液中析出知识 | 数据挖掘、文本挖掘、机器学习 |
| 知识结晶中间件 | 把析出的知识形成结晶 | 本体结构、知识建模 |
| 知识更新中间件 | 维持知识晶体的一致性、完备性和表示规范性 | 真值维护、知识约简、封闭世界假设 |
| 知件生产中间件 | 提取知识结晶的成分加工成知件 | 构件工程、快速原型 |
| 知件组合中间件 | 根据用户需要获取一组知件并装配成功能更复杂的知件 | 知识建模、自动体系结构设计、网构软件 |
| 知件管理中间件 | 登记、注册、发布、保护知识产权、收费 | 需求分析、电子商务、电子服务 |
| 知件协作中间件 | 以不提供知件本身, 只提供分布式知识查询的方式完成在线问题求解 | RPC、知识网格、语义网 |

## 3　知件自举

知件自举(knowware bootstrapping)是把知件工程的实现建立在知件本身的基础上. 我们知道, 知件工程的生命周期模型刻画了知件的开发、演化和维护过程, 其中大量地涉及知识的获取、转化和利用. 它本身是基于知识的. 如果把所有涉及知识处理和利用的地方采用知件作为部件, 那么就得到一个基于知件的知件开发模型和开发环境. 自举作为一种技术在计算机科学和软件的实践中并不少见. 一个程序设计语言的编译程序用该语言自己来写, 这就是自举, 也称自编译程序. 北大开发的“新型构件化应用服务器”PKUAS把服务器平台构件化[22~25], 用构件化的平台来开发软件构件, 这实际上是构件的自举. 这说明, 自举技术有利于把一种技术贯穿于开发和维护过程的始终.

图 5 分别表示熔炉模型、结晶模型和螺旋模型的知件化实现. 这 3 类模块起着完全不同的作用. 在熔炉模型中, 熔炉中的知识表示成本体形式, 构成本体的元素是一些知件对象和知识中间件对象, 统称知识元. 从知识源(例如书籍、报纸、杂志)中获取知识元时需要一个知件作为知识元的模版. 对于基于类自然语言的技术来说, 这个模版就是类自然语言的语法. 通过该语法获取的大批知识元构成一个集合, 此时它们还只是一些知识的素材. 下一步是对它们作面向对象的分析, 其分析结果是两类对象: 知件对象和知识中间件对象. 此时需要另一个中间件, 本体融合中间件, 把它们组织成本体. “融合”是指和知识库中原有的本体融合, 因为知识获取不是一次性能完成的过程. 每当有生产知件的需要时, 根据知件需求说明, 在结构模型知件指导下, 从知识库中萃取一个子本体并进行必要的检查, 包括: 此子本体是否一致(无矛盾), 是否充分(满足需求), 是否完备(知识的内在结构有无空缺?). 通过这些检查的子本体由包装中间件包装后成为知件. 严格地说, 包装成知件之后还必须通过网上评测机构的评测, 才能够成为商品.

在结晶模型中(见图6), 获取知识的对象是海量知识源(如万维网), 这里同样需要一个描述知识元模版的知件和一个描述知识晶体结构的知件. 这里使用一类特定的知识元描述语言WKPL(万维网知识泵取语言)[26], 它是类自然语言和标记语言XML的综合. 与熔炉模型不同的

是, 获取的知识不作为知识浆存进知识库而直接形成知识晶体, 因此, 必须检查新的知识元与老的知识元集合的一致性. 此检查由知识维护中间件执行, 但不一定要随着每个新的知识元的到达立即进行. 一般来说, 在知识元获取过程中一面形成晶体, 一面检查知件需求是否满足, 待到发现需求满足时再检查它的一致性.

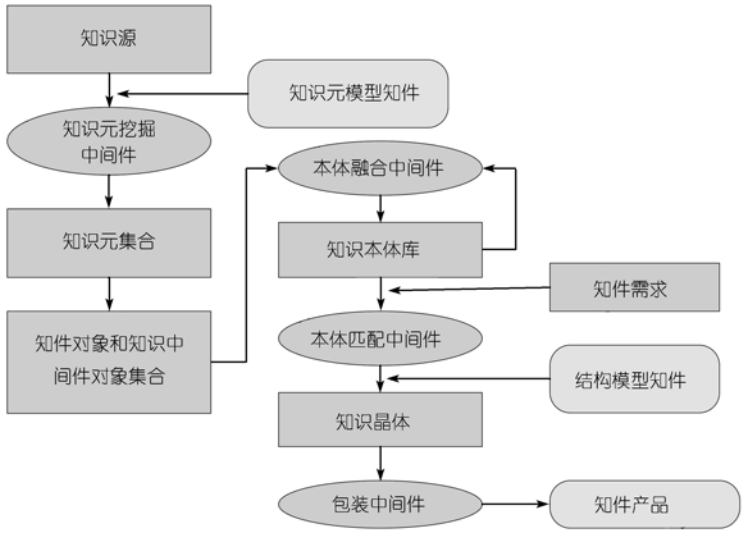


**图 5　熔炉模型的知件化实现**

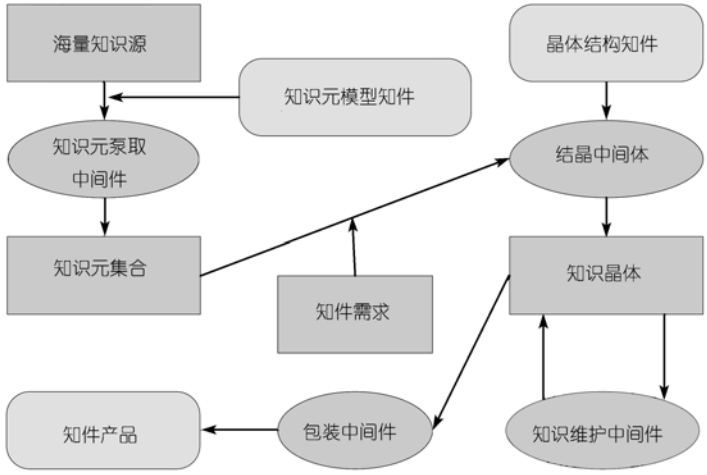


**图 6　结晶模型的知件化实现**

对于螺旋模型, 我们以企业应用系统为例. 每个企业应用系统都包含一个或多个企业管理知件, 这是企业应用系统的灵魂. 这些管理知件在应用系统循环运行过程中不断更新, 更新

可以是来自外部(例如购买新知件)或内部(在本身经验的基础上更新), 图 7 描述的是后者. 通过对企业运行数据进行数据挖掘, 从中获取规律性的知识, 再用这个知识修改原有的知识晶体, 形成新的知识晶体, 再加工成知件, 并用新知件置换原有知件.

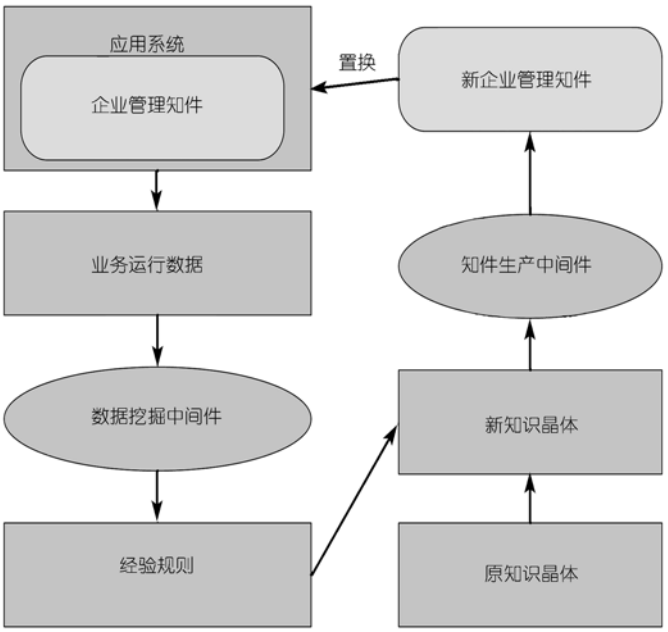


**图 7　螺旋模型的知件化实现**

我们在上面多次提到“知识元”这一概念. 应该说明, 在从知识源获取知识时使用的知识元概念和存储在知识库(熔炉)中的知识元形态是不一样的. 前者是知识元的原始状态, 后者是知识元的加工后状态. 例如, 在基于类自然语言的知识获取中, 被关键结构覆盖的每一个类自然语言语句都是(原始状态的)知识元. 如:

涨价 **的 原因 是** 货物供应不足

涨价 **的 类型 包括** 持续涨价、季节性涨价、临时性涨价 **等**.

其中的粗体字部分属于关键结构, 这两个句子都是被关键结构覆盖的, 都是知识获取时的知识元. 如果它们以语义网络形式存在知识库中, 则加工后可能取如下三元组形式:

(原因(涨价) **is** 货物供应不足)

({持续涨价、季节性涨价、临时性涨价} **sub-class-of** 涨价).

但是如果取文献[26]中引进的知件对象形式, 则加工后的形态为

Knowware Class: 涨价

原因: 货物供应不足

Sub-class: 持续涨价、季节性涨价、临时性涨价

……

方法: 空集

任何知件对象和知件对象类都不含方法, 相关的方法在知识中间件中. 系统运行时要把知件对象和知识中间件对象组合起来, 才可以在面向对象系统支撑下运行. 下一节要引进基于知件的 PROMIS 技术, 其中的加工后知识元就取这种知件对象的形式.

## 4 基于知件自举的 PROMIS 技术

本节讨论知件自举的一个特例, 就是我们的PROMIS技术的知件化, 这里用到全部3类知件工程模型. 为把基于知识的软件工程技术 PROMIS 移植到基于知件的环境中, 需要对 PROMIS 技术作一番解剖. 从图 1 来看, PROMIS 系统大致上可以分解为 3 个主要部分: 领域知识库生成部分、基于领域知识库的应用系统自动生成部分和应用系统运行部分. 其中第 1 部分又可分成领域知识的外部描述(DKDL)、内部描述(ONOET)、外部描述(专家输入)向内部描述的转换, 以及领域本体库(DOKB); 第 2 部分可以分为企业情形描述 BIDL(用户输入)及其本体表示 ONONET 以及系统结构模型 NEWCOM; 第 3 部分包含可执行系统和运行支撑.

我们把 PROMIS 的知件自举称为 PROMIS/KW. 它和 PROMIS 一样也分为 3 个部分, 其中前面两个部分与 PROMIS 基本相同, 只是其中所有的知识支持都被替换为知件. 第 3 部分则加上系统演化的内容. 从知件的观点来看, 第 1 部分对应于领域知件的生产, 第 2 部分对应于领域知件的应用和应用系统的生成, 第 3 部分对应于领域知件的演化.

对于第 1 部分, 领域知件的生产适用熔炉模型, 其中知识源是领域专家口授或编写的领域管理知识, 也包括企业经营规范. 仍采用类自然语言方法输入, 知识编译中间件去掉其中关键结构以外的部分, 剩下的就是可供使用的领域知识. 这些领域知识以组织成本体的领域语义元素形式存在知识库中. 同时, 软件工程师或程序员编写一批基本的应用小程序(Applets 和 Utilities)加进知识库中, 构成一个完整的领域知识库. 需要生成领域模型时由模型生成中间件根据需求说明提取相应的领域子本体(包括领域语义元素和应用小程序), 分别生成知件、知识中间件和软件. 因此, 此处的熔炉模型不同于前一节讲的熔炉模型, 它生产的不仅是知件, 而且还有知识中间件和软件与之配套. 另一方面, 知识分析中间件对这些领域知识做面向对象分析, 获得一批知件对象和知识中间件对象. 我们在上面说过, 知件对象是不含方法的, 为了实际使用知件对象, 需要在系统运行前把知件对象和相应的知识中间件对象组合起来. 是生成模型时就组合, 还是临到运行时再组合, 取决于软件工程师的选择. 由此也可以知道, 基于知件的领域本体表示与原来 PROMIS 中用的表示是不同的.

但是, 熔炉模型所描述的是生成单个知件的全过程. 每次生成知件时, 都由知识库中提取相应的知识元集合, 然后组装完成. 但是从系统开发的实践来看, 知件是可以复用的. 我们并不需要每次使用知件时都从库中的知识浆开始. 我们可以有一个做好的知件库, 每次需要使用知识时由库中提取恰当的知件即可. 但是应用系统的需求差别很大, 这种知件往往不是现成可以得到的. 一个解决办法是把库中的成品知件组合成新的知件. 更严格地说, 是从已有的成品知件中各提取一部分组成新的知件. 这就产生了一个跨本体搜索和推理问题. 为此可以有两种组合方式: 编译方式和解释方式. 在编译方式下, 组合中间件根据库中已有的知件生成一个新知件, 而解释方式并不对知件库做物理变动, 只是由有关的中间件(在图 8 中是功能视图中间件和结构视图中间件)提供一个新知件的视图. 前者有利于知件库的有效复用, 但是每

重构一个知件就对知件库重新组织一次, 效率受影响. 后者在每次建构新知件时要消耗时间和空间资源, 但是在某些情形下会带来较好的效果. 特别是对于正在运行的系统是如此(参见下节).

第 3 部分对原 PROMIS 系统做了丰富与扩充, 主要是添加了系统演化. 对于应用系统来说, 这是非常重要的. 其内容可以分为系统运行数据生成、运行数据挖掘、经验规则生成、新知识晶体生成、新知件生成等. 原则上对应于螺旋模型, 它是领域知识(知件)的不断演化和改进. 但是在螺旋模型之中又包含了一个结晶模型, 它刻画了从运行数据中挖掘知识、形成经验、改进知件的过程. 新知件用来置换原有知件.

图 8 是基于知件的 3 阶段 PROMIS/KW 框架结构, 其中矩形框代表知识、数据或信息, 长椭圆形框代表知识中间件, 圆角矩形框代表知件或知件的集合, 三维梯形框代表程序(软件). 模块“知件生成系统”浓缩了熔炉模型的主要部分, 模块“知件更新系统”浓缩了结晶模型的主要部分. 业情描述是领域用户提供的信息, 这些信息不再以需求描述的形式提供. 针对业情描述构建新知件时采用了解释方式, 因此而有了两个提供视图的中间件.

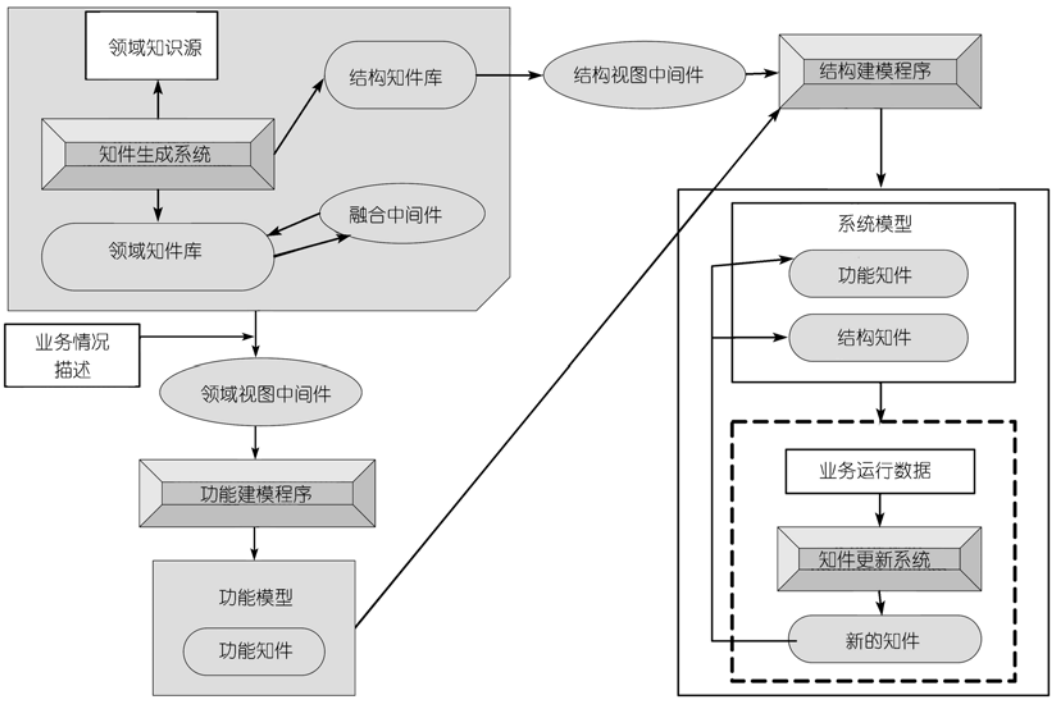


**图 8　PROMIS 系统的知件化实现**

以上几点说明, 知件工程生命周期模型并不是一种固定不变的模式, 并不是在某一种场合只能使用某一种特定的生命周期模型, 也不是只限于使用一种生命周期模型. 我们在刚才已经看到, 在一个应用系统中可以同时使用多个知件工程生命周期模型, 并且一个生命周期模型还可以是另一个模型的组成部分. 这说明生命周期模型还可以嵌套使用.

## 5 后 J2EE 时代的知件工程快速原型技术

文献[27]指出了J2EE出台的时代背景: 一是基于构件的软件开发方法的实践, 其中, “基于构件的软件开发, 有时也称为基于构件的软件工程, 是一种基于对象技术、强调通过可复用构件设计与构造软件系统的软件复用途径”. 二是企业计算的深入. 这些企业“需要支持分布、目录、事务、安全、持久、并发等等服务”, Java的方案是“通过容器将这些企业服务集成起来, 并按照配置自动调用这些服务以满足应用的业务需求”. 三是Internet的普及, “促使传统的客户/服务器两层结构演化为……表示层、逻辑层和数据层的三层体系结构”. 实现J2EE框架的例子可见有关PKUAS的资料[22~25].

在这个形势下, 我们的应用软件自动生成技术必须有本质的提升以适应新时期企业信息处理的需要. 我们要把(虚拟设计, 仅作对照用的)PROMIS/KW 技术改进为基于 J2EE 的技术, 暂时称之为 PROMIS/KW*技术. 它的主要含义是让 PROMIS 生成的应用系统自动形成 J2EE 结构.

PROMIS/KW*保持原 PROMIS/KW 的如下原则不变:

1. 基于类自然语言的领域知识自动获取和领域本体建模;
2. 基于类自然语言的用户业务情形描述;
3. 基于领域(知件)本体库和用户业情描述的应用系统功能建模;
4. 基于环境知识库和系统功能模型的应用系统体系结构建模;
5. 基于体系结构模型的应用系统自动生成;
6. 基于经验数据挖掘和领域知识更新的应用系统自动演化.

这 6 项原则全部是围绕应用系统自动生成和自动演化而制定的, 其中没有一项与 J2EE 框架直接有关. J2EE 涉及的只是应用系统的部署和运行, 既不考虑自动生成, 也不考虑自动演化. 它的位置在 6 项原则的原则 5 与 6 之间. 尽管如此, J2EE 的规定对这 6 项原则还是有影响的. 我们现在看看 J2EE 框架的核心内容:

1. 全部软件组织成构件;
2. 把系统的部署分成 3 个层次: 表示层、业务层和数据层;
3. 表示层存放客户端应用程序, 业务层存放服务器端应用程序, 数据层存放后援系统(遗产系统、数据库等);
4. 每个构件用容器存放, 容器是构件的运行环境.

考察 J2EE 框架、PROMIS 技术和知件技术三者之间的相互影响, 有如下几点是值得注意并必须解决的新问题:

1. J2EE 的 3 层结构容纳不下 PROMIS 的运行机制. 严格地说, 还需要另外 3 个层次才能体现 PROMIS 技术. 它们是: 基础层(领域知识库及其自动生成)、开发层(基于领域知识库的应用系统自动生成)、演化层(应用系统在运行过程中的自动演化). 因此, PROMIS/KW*是一个 6 层结构. 如果把应用系统的生成和运行部分分开, 则或许可以免去基础层和开发层, 但是演化层是不能免除的.

2. 由于出现了知件和知识中间件, 如何将软件、知件和知识中间件组织成构件是一个大

问题. 是把它们都组织成构件?还是部分组织成构件? 如果组织成构件, 是联合起来组成构件, 还是各自独立形成构件?

3. 各有关软件、知件和知识中间件在多层结构上的部署也是一个大问题.

下面概要回答这几个问题.

### 5.1 PROMIS/KW*框架的层次结构

这是 PROMIS/KW*结构相对于 PROMIS/KW 结构改变最大的部分. 在 PROMIS(包括 PROMIS/KW)系统中, 所有应用最终被映射为一个“客户-服务器”2 层结构, 其设计由 NEWCOM 语言编写的程序确定. 在 J2EE 的背景下, PROMIS/KW*对 PROMIS/KW 最大的改进是从“客户-服务器”2 层结构改进为采纳 J2EE 的表示层、逻辑层和数据层的 3 层体系结构. 但这还不是全部. 事实上, J2EE 的 3 层结构并没有考虑到系统演化的问题, 而我们的 PROMIS 系统考虑到了. 所以 PROMIS/KW*体现的至少应该是一个 4 层结构, 其中前 3 层结构对应于 J2EE 的 3 层结构, 第 4 层是演化结构. 需要说明的是, 尽管演化的内容在 PROMIS 和 PROMIS/KW 中已经有了, 但是在那里并没有分出明确的层次来. J2EE 以 3 层结构为其特色, 我们就相应地把系统演化表示为一个单独的层次, 并称加了这个层次的框架为 J2EE/KW*框架, 见图 9 的“应用系统部署区”部分.

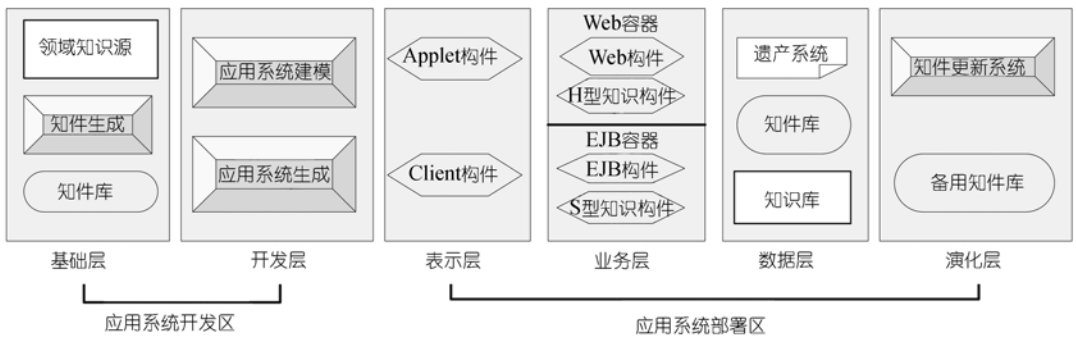


**图 9 PROMIS/KW** 6 层体系结构**

从该部分可见, 演化层有一个备用知件库, 还有一个知件更新系统, 其内容与图 8 中的系统演化部分基本相似. 知件更新的依据来自数据层, 那里有原有的知件库和系统运行数据库, 知件更新系统中的监视中间件不断地分析运行数据库中新数据的走向, 从中总结规律, 提供给知件更新系统的知识晶体维护中间件, 后者根据这些信息把原有的知识晶体改造成新的晶体, 并进一步加工成新的知件. 此时有两种选择, 或者由用户决定是否用新知件替换老知件, 或者应用系统自己决定知件的替换. 如果是后者, 则应由用户事先授权, 把知件演化模式设定为“自动”. 无论是哪一种模式, 都可以以热部署的方式进行. 系统首先阻断所有构件对更换知件的访问, 保存所有访问的状态, 在完成更换知件后再恢复系统的运行.

演化层的领域知件库不一定只组织成一个知件, 而是可以含有多个知件. 这是因为领域知件库的更新一般不是整个知识库同时更新, 而是一部分、一部分地更新, 需要更新哪一部分

就更新哪一部分. 而每次实现更新时不一定要重新组织整个知识库. 这不仅是因为重新组织需要代价, 而且也因为知识库的部分更新有一个热部署的问题, 每当有一部分知识需要更新时, 应用系统的运行不能停下来等待整个知识库的重新组织, 因此, 在许多时候知识库可能处于多个知件分立的情形. 运行系统对领域知识库的访问一般要通过一个领域浏览知识中间件进行, 该中间件以虚拟的形式把知识库中可能是分立的多个领域本体看成统一的本体, 在此基础上进行功能建模.

但是 4 层结构还不足以完整地描述 PROMIS 的基本思想. 要做到这一点, 就要把领域知识库自动生成部分和应用系统自动生成部分也包括在内, 即还要加上两层: 基础层和开发层. 这两层的功能不能被演化层所代替, 因为演化层只能在企业经营策略, 即商业智能方面改进应用系统的性能, 它不能对应用系统的架构做大的修改. 另一方面, 演化层修改的主要是知件, 知件以外的部分, 包括软件和知识中间件, 它都不能修改, 也不能增加知件. 所以 6 层结构才是完整的, 我们把它称为 PROMIS/KW**.

### 5.2 构件的组织和部署

基于知件的开发方法和基于构件的开发方法在技术原理上是一致的. 知件技术并没有破坏基于构件的软件开发机制, 相反地, 它是软件构件技术的进一步细化和发展. 其中的一个关键问题是: 在有知件和知识中间件存在的前提下, 如何设计 PROMIS/KW** 的构件? 我们有 5 种可以考虑的方案.

第 1 种方案把知件当作(只读)数据处理, 放在数据逻辑部分, 而把知识中间件放在业务逻辑部分, 其中知识中间件的插入点要根据它们的类型来区分. 把作为软件-知件桥梁的(S 型)知识中间件(采用 EJB 容器)放在业务逻辑的后端; 把作为用户-知件界面的(H 型)知识中间件(采用 JSP 或 Servlet 容器)放在业务逻辑的前端, 而演化层的知识构件则称为 R-型知识构件. 这种做法的缺点是知件和作用于其上的知识中间件分开了. 知识中间件每次访问知件时都要越过业务逻辑和数据逻辑的中间地带, 还要越出知识中间件所在的容器, 在效率上会有所影响, 因此只适用于知件很大并且访问次数很少的情形.

第 2 种方案是把知件也看作构件, 这样就把原来单一的软件构件分化为 3 类构件: 软件构件、知件构件和知识中间件构件. 这种方案有两个缺点. 首先, 单纯的知件不含程序, 与数据属于同一个类型, 应该属于数据逻辑部分, 把它作为构件放在应用逻辑里并不合适. 其次, 任何知件要想发挥作用, 就一定要有知识中间件的配合. 把知件和知识中间件处理为不同的构件, 势必影响效率. 尤其是在“每个容器一个构件”的框架下, 每次调用知件都要跨越(知件和知识中间件的)容器边界, 非常不合算.

第 3 种方案是把知件和相关的知识中间件绑在一起, 放在同一个构件中. 根据实现系统功能的需要, 还可以把部分知件对象和知识中间件对象合并成新的软件对象, 这样可以改善效率. 但是一般来说, 知件和知识中间件并非是一一对应的, 也不是一对多, 而是多对多. 这就产生了一个问题: 按什么原则把它们绑在一起? 对此还有如下几种可能性:

1. 以知件和知识中间件为节点, 它们之间的访问关系为边, 构成一个图. 取这个图的每个连通子图为一个构件;

2. 以每个知件为核心, 所有作用于它的知识中间件为外围, 组成一个构件, 任何知识中间件用于多个知件的, 制作该知识中间件的多个拷贝, 分发到各个构件中, 此方案的构件数就是知件数;

3. 方案 2 的一种实现方式是把所有这些知识中间件并进核心知件中, 构成一个大对象;

4. 以每个知识中间件为核心, 所有被它作用的知件为外围, 组成一个构件, 任何知件被多个知识中间件作用的, 制作该知件的多个拷贝, 分发到各个构件中, 此方案的构件数就是知识中间件数;

5. 方案 4 的一种实现方式是把核心知识中间件并进每个知件中, 构成一组兼有知件和知识中间件的对象;

对每一个具体的应用系统, 知件构件的解决方案很可能是在综合考虑各种效率因素的前提下对上述几种方案的折衷和组合.

第 4 种方案是组织包括部分软件构件和部分知件(和知识中间件)构件在内的混合构件. 对于软件功能模块和知件模块联系紧密的情形, 可以考虑采用这个方案.

第 5 种方案是根据具体情形, 选择上述 3 种方案的组合, 例如, 受访问次数较少的知件放在数据逻辑部分, 受访问次数较多的知件和知识中间件绑在一起, 放在业务逻辑部分, 这样可以减少管理的开销.

我们原则上采用第 5 种方案.

在图 9 中, 我们把 H 型(S 型)知识中间件或包含 H 型(S 型)知识中间件的构件统称为 H 型(S 型)知识构件.

### 5.3 描述构件的组织和部署的规划语言

在原来的PROMIS系统中, 在从类自然语言BIDL程序到应用系统目标代码的转换过程中有一个中间层次, 称为NEWCOM语言[8], 它是PROMIS将要生成的应用系统的体系结构描述语言. PROMIS 并不从应用系统的功能模型直接生成可运行系统, 而是生成NEWCOM形式的应用系统体系结构描述, 然后才由NEWCOM编译程序把这个结构描述转换成某种目标代码. 应用系统开发者及其用户可以在NEWCOM这一级看到PROMIS为他生成的应用系统的体系结构和功能结构, 并根据实际情形作必要的调整, 而不必陷入编码细节. 这样也为用户提供了一个对PROMIS的生成机制进行干预的手段. 其次, NEWCOM程序编译后生成的是VC++代码, 其中有关数据处理的部分采用ODBC编程, 这就大大减少了对特定数据库的依赖, 使得PROMIS可以运行在任何支持ODBC的数据库平台上. 第三, PROMIS生成的是分布于网络上的应用系统. NEWCOM是一个面向客户/服务器网络结构的语言, 适合用作这类系统的描述工具. 第四, PROMIS 生成的应用系统加强了对数据安全性的管理, 不但严格了对数据的访问和操作权限, 而且采用了数据封装的原则. 当然, NEWCOM的作用绝不限于作为PROMIS的后端, 它是一个独立的应用程序开发工具, 可以由计算机软件人员直接用来编写应用系统.

在 PROMIS/KW**中, 我们也需要这样一个体系结构描述语言, 它会和 NEWCOM 有较大的区别, 因为现在采用的是在 J2EE 的 3 层结构基础上发展出来的 4 层结构(虽然 PROMIS/KW**本身具有 6 层结构, 但是它生成的应用系统只在 4 层结构上运行, 不包括基础层和开发

层), 称这个语言为 NEWCOM/KW. 表 4 是 NEWCOM/KW 程序语法公式的核心部分.

表 4 中的节点都是逻辑节点, 每个逻辑节点的内容分为多个部分. 节点上各部分的物理分布在系统部署时动态决定. 可以是一个逻辑节点分布在多个物理节点上, 也可以反过来, 多个逻辑节点分布在一个物理节点上. 数据保护通过允入和允出说明来指定. 每个部分的允入名单指明了允许访问该部分的其他过程以及允许访问的程度(读、写等). 每个部分的允出说明指明了该部分中哪些内容是可以访问的. 举例来说, 如果过程甲访问数据库乙中的关系丙, 则必须同时满足下列 2 个条件: 甲的名字出现在乙的允入名单中, 丙的名字出现在乙的允出名单中.

**表 4　NEWCOM/KW 基本语法**

```
〈网络部分〉: := NETWORK (〈网络名〉)“;”
                    [ PROTOCOL = 〈网间协议名〉; ]
                    {〈子网部分〉}1n
                END_OF_NETWORK;
〈子网部分〉: : = NETWORK (〈子网名〉) “; ”
                    [ PROTOCOL = 〈网络协议名〉; ]
                    {〈节点部分〉}1n
                END_OF_NETWORK;
〈节点部分〉: :=〈表示层节点〉 | 〈业务层节点〉| 〈数据层节点〉| 〈演化层节点〉
〈表示层节点〉: : ＝REPRESENTATION(〈表示层节点名〉)
                APPLET (〈应用小程序名〉)
                    [〈允入部分〉]
                    [〈过程部分〉]
                CLIENT (〈用户程序名〉)
                    [〈允入部分〉]
                    〈用户过程部分〉
            END_OF_REPRESENTATION
〈业务层节点〉: : ＝BUSINESS(〈业务层节点名〉)
                [ WEB  ＝〈WEB 构件名〉
                        [〈允入部分〉]
                        〈过程部分〉]1n
                [ JSP  ＝〈JSP 构件名〉
                        [〈允入部分〉]
                        〈过程部分〉]1n
                [ EJB =〈EJB 构件名〉
                        [〈允入部分〉]
                        〈过程部分〉]1n
            END_OF_BUSINESS
〈数据层节点〉: : ＝DATA(〈数据层节点名〉)
                [DBMS＝〈数据库管理系统名〉
                        [〈允入部分〉]
                        〈允出部分〉
                        [〈数据库操作部分〉] 1n
                        〈数据部分〉] 1n
                [LEGACY＝〈遗产系统名〉
```

表 4(续)

```
                    [<允入部分>]
                    <允出部分>
                    [<遗产操作部分>]1n
                    <遗产部分>]1n
              [KNOWWARE=<知件名>
                    <允出部分>
                    [<知识中间件部分>]1n
                    <知件部分>]1n
          END_OF_DATA
<演化层节点>::=EVOLUTION(<演化层节点名>)
          RUNDATA=<系统运行数据库名>
                    [<允入部分>]
                    <允出部分>
                    <数据部分>
          MONITOR =<监视中间件部分>
          MINING =<挖掘中间件部分>
          K-CRYSTAL =<知识晶体名>
                    [<允入部分>]
                    <允出部分>
                    <数据部分>
          WRAPPING = <包装中间件部分>
          KNOWWARE = <知件集合>
      END_OF_EVOLUTION
```

### 5.4 PROMIS/KW**的服务组织

相比于 J2EE 应对的企业信息系统需求, 原 PROMIS 技术解决的问题只是针对小型企业的信息服务需求. 在新的需求下, 上述支持企业计算的各种服务, 如分布、目录、事务、安全、持久、并发等等都应该考虑进去. 好在我们已经有类似于 PKUAS 这样的 J2EE 支撑技术, 许多事情无需在知件工程的层面上操心. PROMIS 技术的一个重要基础是领域建模, 而更加关注面向特定领域的应用服务器也是 PKUAS 的一大特点. 然而, PKUAS 只是在公共服务层面上较多地考虑照顾不同领域的特殊需求, 而 PROMIS/KW 引入的知件则从根本上解决了应用服务器的领域特色问题. 原则上, 我们可以使所有的服务知件化, 从而使得系统的在线演化变得十分容易.

## 6 相关工作及研究反思

基于知识的软件工程已在文献中研究得很多, 它几乎涉及软件工程的所有领域和所有生命周期阶段. 它们的一个共同特点是要以知识库为基础. 例如DAIDA项目[28]利用一个软件工程数据库, 其中包含了从需求说明到系统实现各个阶段的知识, 它能把数据和事务的约束转换成抽象机形式, 最终成为用数据库语言DBPL编写的程序, 从而能支持信息系统的设计和实现. IRIS系统[29]从面向对象的需求说明生成一个信息系统. CADDY[30]则是一个计算机辅助软

件工程工具, 可以从一个扩充的实体-关系模型[31]出发来生成信息系统. 基于知识的软件工程的另一个特点是它们的大多数研究都从需求工程开始做起[32,33]. 软件建模是需求工程的一个重要手段. Yu[34]提出了一个面向Agent的建模框架i*, 着重分析软件工程需求中各有关方面的目标、信念、能力和承诺等要素, 以此识别诸如目标、任务、资源(包括软资源, 如优化)之间的相互依赖关系. 之后有很多人把i*框架和其他软件工程技术结合起来[35,36], 这方面比较近的工作是Castro等人[37,38]在Tropos方法学方面的工作, 把软件开发过程分为早期需求分析(i*框架)、后期需求分析、体系结构设计和详细设计 4 个阶段, 通过Jack编程平台实现. Bhuiyan等人[39]把用于需求分析早期的i*框架和用于需求分析后期的UML技术结合起来, 形成一套前后呼应的需求建模技术. 基于知识的软件工程的另一个方向是基于软件复用, 有人称这项技术为"用组装代替编程"[40]. 近年来, 软件复用逐渐移到网上, Web服务的发现和组合就是在新形势下出现的技术[41~45].

比起以上的工作来, 我们的 PROMIS 技术有 5 个主要的特点: 一是 PROMIS 不是从需求分析做起, 不需要用户给出需求, 而是直接从用户给出的业务情形描述做起, 自动生成企业信息系统的模型; 二是 PROMIS 支持应用系统实现的全过程自动化, 原则上不需要编程; 三是特别强调领域知识的作用, 领域知识单独获取、单独存放和管理, 不和软件工程知识混在一起; 四是 PROMIS 采用类自然语言技术, 便于领域专家给出领域知识, 更便于领域用户给出业情描述, 有利于把软件开发和维护的主动权交到领域用户手里; 五是从业情描述, 到企业信息系统功能模型, 到企业信息系统结构模型, 以至最后的高级语言程序, 由粗到细每一层模型都具备, 以后要修改、维护或演化, 甚至为其他类似企业开发信息系统, 都可以任意利用其中的每一层模型, 做必要的修改后重新生成信息系统.

我们在PROMIS工程中强调把领域知识和软件的开发区分开来, 并且认为和软件一样, 领域知识本身也应该商品化. 但是并未找到它的商品化的合适形式. 后来终于找到了这样的形式, 这就是知件. 2005 年有关知件的文章第 1 次发表[16]. 在后续的工作中, 我们陆续研究和完善了知件工程[17]、知件/软件协工程[19, 20]、面向对象的知件开发方法学[26]、知件作为服务[26]、知件技术在信息系统开发中的应用[18]等等[20]. 本文工作就是进一步发展在文献[46]中提出的框架, 把PROMIS的软件自动生成技术、知件技术和J2EE技术三者结合起来, 形成一个在J2EE背景下的基于知识的软件/知件协同开发框架.

基于知件的软件工程在技术上还不完善, 还没有经过工程化和产业化的考验, 无论是从方法学、工具、语言、形式语义还是实现技术等方面来看, 要做的事情都很多. 从方法学来说, 软件和知件的协同开发仍有大量的问题需要研究, 其中的一个重要挑战是: 传统软件和传统知识系统的开发各有自己的显著特点, 数十年间都已经积累了许多独特的技术和经验, 能否把这些历史遗产吸收、融合到软件/知件协工程中去, 将是对知件研究的一大考验. 其次, PROMIS 项目的历史经验表明, 没有一套系列化的工具作为支撑, 一个完整的方法学就难以实现. 对我们来说, 系列化的工具主要体现在系列化的语言(领域知识描述语言和软件/知件技术描述语言)及其实现上. 我们要重新开发一套基于 J2EE 技术的、系列化的 PROMIS/KW**工具和语言, 以实现 J2EE 背景下的应用软件自动生成和演化. 至于知件的形式语义, 以及软件/知

件协工程的形式语义, 则更是一个挑战性极大的研究课题, 是今后的重点探索方向.

## 参考文献


1 Lu R Q, Jin Z, Wan R L. A knowledge-based tool for automatically prototyping management information systems. AI Watch, 1994, 3(2): 10—11

2 Lu R Q, Jin Z, Wan R L. Requirement specification in pseudo language in PROMIS. In: Proceedings of the 19th International Computer Software and Applications Conference. Washington: IEEE Computer Society, 1995. 96—101

3 陆汝钤, 金芝, 万荣林, 等. 基于领域知识的需求信息获取. 软件学报, 1996, 7(3): 137—144

4 Lu R Q, Jin Z, Xia Y. An intelligent CASE tool for information management systems. In: Proceedings of the First International Conference on Future Computer Systems. Xi' an: Software Engineers Association of Japan, 1996

5 Lu R Q, Jin Z. A multi-agent and pseudo-natural language approach for intelligent information service. In: Proceedings of the International Conference on the Software Engineering and the Knowledge Engineering. Madrid, 1997. 422—429

6 Lu R Q, Jin Z, Liu L, et al. PROMIS 2.0: an intelligent case tool for MIS in the client/server application. In: Proceedings of ISFST-97. Xiamen: Software Engineers Association of Japan, 1997. 399—405

7 Lu R Q, Jin Z, Liu L, et al. OSNET-a language for domain modeling. In: Proceedings of the Technology of Object-Oriented Languages and Systems. Washington: IEEE Computer Society, 1998. 83—92

8 陆汝钤, 金芝, 刘璘, 等. 客户机/服务器风格的体系结构描述语言 NEWCOM. 计算机学报, 1998, 21(12): 1103—1111

9 Lu R Q, Jin Z. Knowledge based hierarchical software reuse. Chinese J Adv Softw Res, 1999, 6(1): 1—11

10 陆汝钤, 金芝, 陈刚. 面向本体的需求分析. 软件学报, 2000, 11(8): 1009—1017

11 Lu R Q, Jin Z. Domain Modeling based Software Engineering. Dardrecht: Kluwer Academic Publishers, 2000

12 Lu R Q, Jin Z. Formal ontology: foundation of domain knowledge sharing and reusing. J Comput Sci Technol, 2002, 17(5): 535—548

13 Lu R Q, Jin Z. Domain knowledge representation: using an ontology language. Appl info, 2003: 1302—1307

14 Lu R Q, Jin Z. Automating application software generation. Expert Syst, 2003, 20(2): 71—77

15 Jin Z, Lu R, Bell D, Automatically multi-paradigm requirements modeling and analyzing: an ontology-based approach. Sci China Ser F-Inf Sci, 2003, 46(4): 279—297

16 Lu R Q. From hardware to software to knowware: IT' s third liberation? IEEE Intell syst, 2005: 82—85

17 Lu R Q, Jin Z. Beyond knowledge engineering. J Comput Sci Tech, 2006, 21(5): 790—799 [DOI]

18 Lu R Q. Towards a software/knowware co-engineering. In: Knowledge Science, Engineering and Management, LNAI 4092. Berlin/Heidelberg: Springer, 2006. 23—32

19 Lu R Q. Knowware, knowware engineering and knowware/software co-engineering. In: Proc ICCS, 2007

20 Lu R Q. Knowware research and fourth industry. In: Economic Globalization and the Choice of Asia. Shanghai, 2005

21 Nonaka I, Takeuchi H. The Knowledge Creating Company: How Japanese Companies Create Dynamics of Innovation. Oxford: Oxford University Press, 1995

22 Mei H, Huang G. PKUAS: an architecture-based reflective component operating platform. In: Proceedings of the 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS). Suzhou: IEEE Computer Society, 2004. 163—169

23 Huang G, Liu T, Mei H, et al. Towards autonomic computing middleware via reflection. In: Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC). Hongkong: 2004. 122—127

24 Huang G, Mei H, Yang F Q. Runtime software architecture based on reflective middleware. Sci China Ser F-Inf

Sci, 2004, 47(5): 555—576
25 Huang G, Mei H, Yang F Q. Runtime recovery and manipulation of software architecture of component-based systems. Int J Autom Softw Eng, 2006, 13(2): 251—278
26 Lu R Q. Knowware, the Third Star After Hardware and Software. Monza Mi: Polimetrica Publisher, 2007
27 梅宏, 黄罡, 曹东刚, 等. J2EE 技术剖析与评价. 中国计算机学会通讯, 2005, 1(4): 26—35
28 Jarke M, Jeusfeld M, Mylopoulos J, et al. Information systems development as knowledge engineering: a review of the DAIDA project. Technical Report MIP-9010. Passau: University of Passau, 1990
29 Vassiliou Y, Marakakis M, Katalagarianos P, et al. IRIS: a mapping assistant for generating designs from requirements. In: Proceedings of the Conference on Advanced Information Systems Engineering. Berlin/Heidelberg: Springer, 1990. 307—338
30 Engels G, Lohr-RichteP R. CADDY-A highly integrated environment to support conceptual database design. In: Proceedings of the 5th International Workshop on CASE. Montreal: IEEE, 1992. 19—22
31 Engels G, Gogolla M, Hohenstein U, et al. Conceptual modelling of database application using an extended ER model. Data Knowl Eng, 1992, 9(2): 157—204 [DOI]
32 Mylopoulos J, Chung L, Yu E. From object-oriented to goal-oriented requirement analysis. CACM, 42: 31—37
33 Kundu S. Structuring software functional requirements for automated design and verification. COMPSAC' 07. 127—134
34 Yu E. Towards modeling and reasoning support for early phase requirement engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering. Washington: IEEE, 1997. 226—235
35 Wang X Y, Lespérance Y. Agent-oriented requirements engineering using ConGolog and i*. In: Proc AOIS 2001. 59—78
36 Vilkomir S, Ghose A K, Krishna A. Combining agent-oriented conceptual modeling with formal methods. In: Proc ASWEC. Washington: IEEE Computer Society, 2004. 147—157
37 Castro J, Kolp M, Mylopoulos J. Towards requirements-driven information system engineering: the Tropos project. Inf Syst, 2002, (27): 365—389
38 Fuxman A, Liu L, Mylopoulos J. Specifying and analyzing early requirements in Tropos. Requirements Engineering, 2004, 9(2): 132—150 [DOI]
39 Bhuiyan M. Integration of agent-oriented conceptual models and UML activity diagrams using effect annotations. COMPSAC07, 2007. 171—178
40 Monroe R J, Gailen D. Style based reuse for software architectures. In: Proceedings of the 1996 International Conference on Software Reuse, 1996. 84—93
41 McIlraith S A, Son T C, Zeng H L. Semantic Web services. IEEE Intelligent Syst, 2001, 16(2): 46—53 [DOI]
42 Bansal S, Vidal J M. Matchmaking of Web services based on the DAML-S service model. In: Proceedings of International joint conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03), 2003. 926—927
43 Maamar Z, Mostefaoui S K, Yahyaoui H. Toward an Agent-Based and Context-Oriented Approach for Web Services Composition. IEEE Trans Knowl Data Engin, 2005, 17(5): 686—697 [DOI]
44 Hou L, Jin Z, Wu D, Modeling and verifying Web services driven by requirements: An ontology based approach, Scin China F-Inf Sci, 2006, 49(6): 792—820 [DOI]
45 Wang P W, Jin Z, Liu L, Cai G J. Building towards capability specifications of Web services based on an environment ontology. IEEE Trans Know Data Engin, 2008, 20(4): 547—561 [DOI]
46 Lu R Q, Jin Z. From knowledge based software engineering to knowware based software engineering. In: COMPSAC' 07, 2007