50-55 软件工程 集成化 CASE (12)

计算机科学1993 Vol.20 No.5

集成化CASE

TP311.5

张 莉 （北京航空航天大学软件所 北京100083）

摘 要

近几年来，随着计算机辅助软件工程(CASE)工具的发展，CASE的研究和开发重点已从确保各CASE工具的正常工作转移到了确保各个不同厂家的CASE工具的协同工作，因而集成成为开发和维护CASE工具的关键技术。

目前，国外许多著名的软、硬件公司都纷纷投资于集成CASE的研究，并提出了各种各样的CASE策略和框架，随之也应运而生了一些国内外的CASE标准。本文根据Ronald J. Jorman和Minder Chen发表在IEEE Software 1992.3上的两篇文章编写而成，以有助于国内集成化CASE。

一 CASE的发展

当前CASE环境中的工具和方法，其设计都是为了支持应用程序的开发，因此，CASE的发展在很大程度上受到各个时期所开发的应用程序和开发这些应用程序所用到的方法的影响。

	应用程序	方法	工具
1970	• 批处理系统 • 联机处理系统	• 结构化编程 • 结构化设计 • 结构化分析	• 高级语言编译程序 • 交互式编程环境 • 结构化 基于文本的(Upper CASE)
1975			
	• 决策支持系统	• 信息系统规划	• 代码生成器
1980	• 实时系统 • 专家系统	• 软件度量 • 原型(化)	• 第四代语言 • 基于结构化方法和图形的CASE
1985	• 决策信息系统 • 可执行信息系统	• 信息工程 • 合作设计	• 支持集成项目的环境 • 基于信息工程的 CASE
1990	• 多机处理系统	• 形式化方法 • 面向对象的方法 • 软件过程管理	• 逆向工程CASE • CASE外壳 (Shells) • 基于Repository的CASE
	• 集成化系统		• 智能 CASE
1995		• 集成化方法	• 合作 CASE • 集成化CASE 环境

图1 CASE技术的发展

张 莉 博士生，主攻方向：CASE集成标准。

新的应用程序又推动了新的系统开发方法的产生。由于这些新的开发方法又比较复杂，因此为了有效地使用这些方法又开发出各种工具，并随着工具的功能越来越强，对新的应用程序和工具的开发也越来越容易。它们就是这样相互作用，相互推动而发展的，见图1。

·应用程序(Application)。七十年代，大多数商业应用程序都是用第三代语言编写的成批事务处理系统。之后，随着数据库技术的成熟，开发出了更为复杂的、数据集中的在线事务处理系统。七十年代后期，又建立了决策支持系统，它通过与决策模型的交互作用来帮助用户进行数据的分析。

进入八十年代，随着实时软件在控制与通信设备中的使用，产生了Ada。到了八十年代中叶，专家系统和基于知识的应用程序引起了广泛的注意。八十年代后期，组织者们为了能在竞争中保持优势开始使用决策信息系统，而高级管理者则已开始使用可执行的信息系统来检索信息。

九十年代，要求这些信息系统必须被集成，以覆盖所有的商业功能、组织层次和总体分配。创建这样的系统将需要组合多种可用技术。如用户-服务器构造、图形用户界面等。正是创建这样的系统在组合技术的复杂性以及时间上的要求驱动了集成CASE的发展。

·方法。结构化编程是最早的支持系统开发的系统化方法之一。由于要修正软件生命周期早期所产生的错误，花销很大，随后又分别为设计、分析和规划开发出了一套相应的结构化技术。

从八十年代中期到后期，为了弥补结构化技术的不足，开发了实时系统的设计、面向对象的分析和设计方法。这期间还着眼于开发一些与整个软件生命周期有关的方法和技术(如信息工程等)。这些方法的严密性和复杂性表明在其中已开始引入了对CASE的使用。

由于代码生成器和第四代语言在某种程度上简化了后一阶段的工作，因而开发的瓶颈转移到了前一阶段的工作，包括系统规划、企业模型和需求工程。

前阶段产品的质量取决于系统允许用户和管理人员介入的程度。象联合应用设计(Joint Application Design)的参与设计方法(Participatory Design Method)对用户参与设计很有帮助，它是对其它方法的一个有益的扩充，并在近五年内开始为人们所接受。

结构化方法开发于七十年代，流行于八十年代，并加以改进，加入了图形表示并更面向最终用户。但是许多结构化方法只能处理信息系统模型的一到几个侧面，批评家们认为它们太不严密也太模糊了。

形式化方法要更严密一些，但通常又不适用于处理用户和开发者间的通信。形式化方法和结构化方法恰好能够互相补充，能否将二者结合起来呢？这是一个极有吸引力的问题。

如果我们希望改进开发过程，那么就有必要对效率和质量进行客观的测量。当过程管理完全被集成进CASE环境时，由于测量数据可以被自动地收集，从而也推动了测量的发展。由于在工具、方法和应用统计的过程控制中使用测量有助于我们更好地管理开发过程，因此进入九十年代，软件的测量方面将进一步发展。

当前许多为支持某些方法而设计的CASE工具使得这些方法走向实用，而这些CASE工具间的协作则表明集成化CASE的进展将依赖于将软件生命周期中各阶段的或横跨整个生命周期的以及横跨多个应用领域的模型方法集成起来的集成方法的开发。

·工具(Tools)。系统开发生成了大量必须被获取并要求被分析的开发信息。CASE环境允许系统开发者从最初的用户需求开始，到设计和实现进行文档编制和模型制作，然后再将其用于一致性和完整性测试，并要求其符合标准。CASE技术的使用也将成为成功地开发大规模系统项目的关键技术。

七十年代早期，第一代CASE工具(PSL/PSA[3])基本上是基于主机和文本的。这些工具的作用在于它们激发了结构化方法的开发，进一步实现了自动工具的必要收集，并有助于分析使用结构化方法所产生的大量开发信息。

在PC机和工作站上图形用户界面的尝试使得可以用图形的前端工具来支持结构化方法。但是，在这些早期的工具中所获取的信息是存贮在工具内部的，且通常在工具间不能被传送。

第二代CASE工具开发于八十年代早期，主要目的是支持结构化方法中图形表示的使用，如结构化分析中用到的数据流图、结构化设计中用到的结构图等。这些工具所获得的详细开发信息被存贮在一个项目字典中，可被同一环境中的其它CASE工

具所共享。但是，这类集成只限制为同一厂家生产的工具，通常还限制为同一项目中的数据。通过增加一些从项目字典中输入或输出引发信息的工具可以达到CASE环境的松集成。偶尔情况下，甚至不同厂家的产品，只要它们在数据格式或专用的应用程序界面上是一致的话，也可以进行链接。

八十年代后期出现了基于信息仓（Repository）的CASE工具，提供了企业范围和项目级的局部信息仓，该信息仓集成了一套用于规划、分析、设计、编程、测试和维护的工具。但是，这类产品仍然在很大程度上依赖于所采用的方法，并只能支持应用程序开发的某些方法。

为满足更快地开发高度集成和更为复杂的决策信息系统，九十年代将通过开放系统使系统集成进入一个新的阶段。集成化CASE环境使得IS结构能够在开放的系统平台间及时地传送系统和移动系统。

要满足这些要求，一个集成CASE环境必须基于灵活的框架结构、提供成本经济的工具集成机制，鼓励可移植性工具，能进行开发信息交换的设施，并能适应未来的方法，向别的工程学科靠近。

二 集成CASE框架

本节将介绍基于由国家标准与技术研究所（NIST）和欧洲计算机制造商协会（ECMA）开发的参考模型的一种集成CASE环境的技术框架，以及R.J.Norman和M.Chen提出的一种构造框架，它们能将该技术放入企业、项目等级别的信息系统开发和管理的过程中。

1 技术框架

一个集成CASE环境必须是可适应的、灵活的且能动态地支持企业、项目和人员。在这样的环境中，用户可以找出支持所选方法的最佳工具，然后将这些工具插入环境中，便能用它们进行工作。

这里采用了NIST/ECMA参考模型，见图2，作为描述集成化CASE环境中各种技术的根据。该参考模型中所定义的设备可分为三种形式的集成：

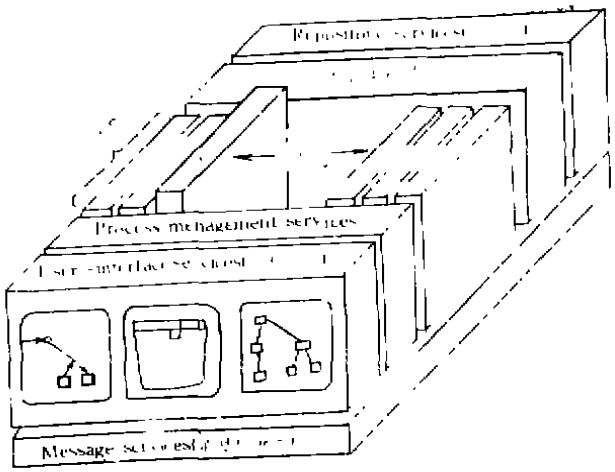

图2 NIST/ECMA参考模型

• **数据集成**—由repository和data-integration services支持

• **控制集成**—由process-management和message services支持

• **表达集成**—由user-inteface services支持

（1）**数据集成**：集成工具的关键在于共享设计信息的能力。根据IEEE标准草案，有四种信息共享方法：（*i*）两个工具间的直接信息转换。这在实时集成时很有效，但当有多个工具集成时难以实现。（*ii*）基于文件的转换是最易于实现的，由*EIA*开发的*CASE Data Interchange Format*是最为成熟的基于文件的转换标准。（*iii*）基于通信的转换。适用于开放系统和分布式系统。（*iv*）基于*Repository*（信息仓）的转换，支持紧耦合的一致性环境，并且是一些集成化*CASE*产品的基础，一个（数据）信息仓提供了许多基本的服务器，包括对象/事项和链接/关系的存贮和管理；版本和配置控制；服务器命名；安全性和事务处理控制。

（2）**控制集成**：由消息服务和进程管理支持。要求工具能注意到其它各个事件，激活程序控制下的其它工具，并能共享函数。

控制集成机制包括显式的消息传送、[illegible]间或存取启动设备及消息服务器。要取得控制集成，在消息服务器上调用的工具必须能提供3种类型的通信：工具到工具、工具到

服务器以及服务器到服务器的通信。

进程管理服务能显式地表示工具调用序列和策略，因此用户可以了解工具的细节，也能自由地处理抽象的任务和进程。

由于控制集成的发展，提高了自动化水平，因此该环境能支持全局进程管理、项目级任务管理和工具的调用。PCTE(Portable Common Tool Environment)是ECMA采用的工具界面标准，以支持工具的可移植性。在PCTE下，工具被存贮在信息仓中，并被作为进程执行，这使得改善工具的运行、组成、通信和同步成为可能。

(3) **表达集成**：用户界面服务器使得CASE用户能与工具进行一致性交互作用，并使得新工具更容易学习。基于窗口的工具有四个层次的表达集成：窗口系统、窗口管理程序、用户界面开发工具箱和视感(look-and-feel)指南。

对于用户界面，大多数CASE框架和环境都采用了通用的标准，例如Motif是开放系统中大多数CASE工具所使用的表达标准。有的环境对此作了一些扩充。

(4) **其它**。NIST/ECMA参考模型直接通过工具层提出了两种方向的CASE集成，即构造垂直和水平工具，并将其插入工具层便可以支持垂直和水平集成。(i)垂直(整个生命周期)集成确保软件生命周期各个阶段所产生信息的完整性与一致性。支持垂直集成的机制有：正向、逆向工程、构造和修改管理以及需求跟踪工具等。(ii)水平(方法论上)集成维护当使用多种模型方法(如数据过程、事件驱动、面向对象)时生命周期各阶段中信息设计的完整性。支持水平集成的机制有信息仓元模型、完整性检查规则等等。

2 构造框架(Organizational framework)

上节所描述的技术框架不考虑具体工具的功能，而是如图2所示，将工具插入工具层，它再请求框架服务器来支持一具体的系统开发功能。

图3中所示的构造框架企图将CASE工具放入开发和管理范围中。在图的左边，我们将一个环境的服务器和工具分为三层，每层中的构件支持右边相应的活动。

在图的右边，该框架将系统开发和管理分为三个活动层次：

(1) IS基本结构规划和设计，由企业层(enterprise level)负责

(2) 系统项目管理和决策，由项目层(project level)完成

(3)软件开发过程，由individual和team level完成

高层活动控制低层活动，而低层活动又反过来影响高层活动。一个集成化CASE环境必须支持所有各层的活动。

该构造框架是对技术框架的补充，有助于集成化CASE环境的开发和推广，指导将来的研究，并能帮助CASE用户选择和构造集成化CASE环境中的工具。

(1) Enterprise Level—IS基本结构的规划和设计。在此最高层的活动决定了开发

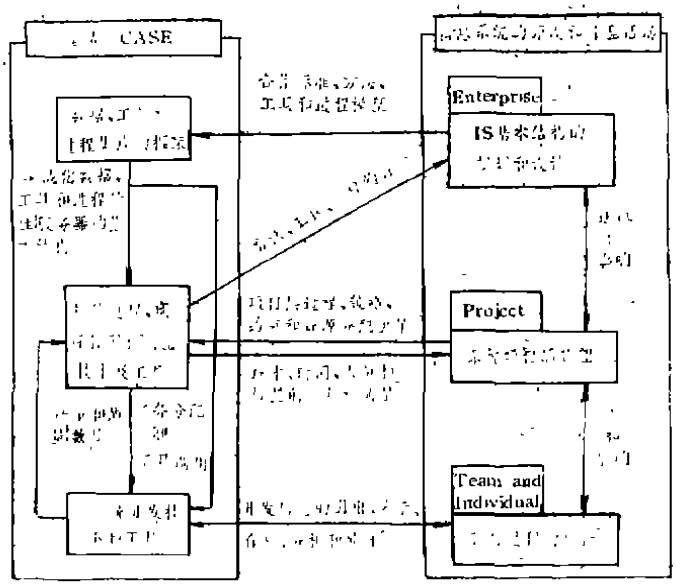

图3 构造框架

过程的管理方式。该层最关心的是建立企业范围的IS基本结构。该层的活动包括：

- 建立计算机平台的配置指标。
- 估算并决定要采用的方法和过程模型。
- 将项目管理和协调的过程与策略形式化。

在企业级提供数据、工具和过程集成基本结构的集成化框架，不依赖于具体的语言、方法、工具和平台，因此有较强的生命力。

(2) Project Level—系统项目管理。第二层包括跨越整个软件生命周期大部分或所有阶段的管理和决策过程。其活动包括项目和过程管理、效果分析、对管理与文本的修改以及重用。例如，由工具支持的项目管理活动有制定计划和跟踪、人员分配、成本估算和协调工作。

过程管理工具能够调用某些CASE工具，并根据所定义的过程模型检索设计对象，此外，项目间的协调还可以使用通信工具，如电子邮件等。

被CASE工具获取并存入信息仓中的设计信息主要包含了许多关于过程和产品的属性。在集成化CASE环境中，测量工具能使用这些信息来获得关于生产率和质量的度量。一些CASE产品能够自动收集数据并计算度量结果。重要的是开发过去的项目数据库，调整已有的度量和模型，并引伸出新的模型。这些度量结构又反过来为进度的制定、成本、人员估算、以及过程和产品的质量控制提供新的依据。

IS人员也可以使用测量结构来评价工具、方法和过程模型，以确保能成功地监控和改进正在使用中的过程。

(3) Team&individual level—软件过程的执行。在此最低的一层中，集成化CASE应能支持软件生命周期中各阶段的活动：计划、分析、概要设计、技术设计、编码、测试、产品和维护。高级或前端CASE工具支持前阶段活动；而低层或后端CASE工具支持后阶段活动。垂直工具支持引导、表达、存贮、分析和开发信息的转换。

需求引导工具便于开发者和用户在定义系统规范时进行交互。这些工具可以以各种形式引导和表达信息，并将它们存于局部或企业域的信息仓中，并根据所用方法的一些规则检查该规范的一致性和完整性。

为了进一步辅助设计信息的映射，一些CASE工具包含了转换功能，它们能将规范从生命周期的上一阶段送入下一阶段。要得到垂直集成，各阶段间应有二种转换方式，即正向和/或逆向工程。逆向工程主要用在当开发信息只用于低层形式（如代码）时，而其余大部分时间几乎不用，因为集成化CASE中开发的系统是通过高层规范来维护的。

不同的CASE环境要求不同的集成形式和层次。例如，科学和工程应用程序要求更多的控制集成，而商业应用程序则侧重于数据集成。CASE用户应根据其开发的基础和实际情况选用适当的集成形式。

三 小结及未来

1. 未来的发展。将通过组合各种技术（如基于数据库和知识的系统、面向对象的技术和超媒体）来开发更复杂的、集成化程度更高的应用程序。目前CASE环境仅能使用一、二种指定技术，在有限的区域中支持应用程序的开发。将来的集成化CASE环境应以开放系统环境的形式支持更广泛的应用程序。

2. 虽然象版本控制、配置控制和多用户信息仓等机制能以多种方式支持程序设计，但是目前CASE技术仍鼓励专用的开发方法。将来在质量和生产率上取得的进展将引导人员间的交互（包括开发人员之间，以及开发人员和用户之间）。未来的CASE环境应组合各协作工具（分组软件）来支持共同合作的开发。

3. 虽然当前CASE环境中的信息仓包

含了对管理人员十分有价值的有关组织结构、商业目标和过程的信息，但是该信息仓仍然只能被系统开发的IS人员使用。将来的CASE环境应同时支持系统开发和信息传送。用户和管理人员可以根据存贮在信息仓中的元数据，取得关于所访问的应用程序数据一些前后关系。

4. 设计和选择一个集成化CASE环境的关键之处在于在集成化和灵活性间取得平衡。通常，集成化程度越高，开放性就越差。CASE外壳(也称为元系统、CASE工具生成器和元CASE) 是一类裸CASE环境，允许CASE开发者和用户根据其新的或专用的方法制定CASE工具和环境。目前已有的CASE外壳有CADWare的Foundry，Intersolv的XL/Customizer，以及Systematica的Virtual Software Factory。

5. 重用技术是提高生产率的最有效途径之一。使用可重用构件，不仅能减少开发所需的费用，而且能提高开发速度和产品质量。重用的概念包括代码重用，以至更高层的规范和过程的重用。面向对象的技术为创建可重用构件提供了有效的机制、继承和封装。人工智能技术，如类比推理和情景推理也有助于确认和选择重用构件。

6. CASE标准在开发开放的CASE系统中起着重要的作用。集成化CASE环境应以这些标准为基础。但是，目前许多CASE标准交叉重叠，甚至互相冲突。如果我们要实现切实有用的CASE标准，就必须协调各形式化标准组织，取得用户和厂家的支持，并根据其它相关的标准开发CASE标准。

7. 技术转换和组织行为的学习是与集成化CASE有关的另一个必要的研究领域。

8. 最后，AI技术使得集成化CASE环境能包含具体域的知识，以帮助终端用户可以用高级语言或图形工具来开发和维护自己的系统。最终，使得终端用户得以检索或购买高层可重用模型，并加以修改，然后将其插入一个集成化CASE环境，生成他们自己的应用程序。

(接第17页)

• syntolic构造子：它把任意的一个超平面进程结构规定为脉动阵列；

• 树构造子：形成静态或动态进程树；

• 一般并行构造子：允许不相干的进程能相互独立地并行执行。

控制结构模型独立于系统结构并且是描述简单的，但计算开销一致性则取决于构造子或控制原语的选择。

5. 结束语

上面我们指出对一个并行计算模型的基本要求有：系统结构独立性，使模型上开发的软件易于移植，能充分利用物理机器的性质；计算开销一致性，能在模型层次上分析并行算法的复杂性；描述简洁性，便于程序设计者开发并行软件。我们认为，为了使并行计算能得到广泛应用，必须把软件工程、语言设计、智能编译和系统结构设计作为一个整体加以考虑。

参考文献

[1] D.B.Skillcorn, Architecture-Independent Parallel Computation, IEEE Computer 23(12):38-51, 1990

[2] D.B.Skillcorn, Models for Practical Parallel Computation, Inter, Jour. of Parallel Programming, Vol20, No.2, 1991

[3] W.F.Mcloll, Parallel Algorithms and Architectures, LNCS, 384, 1-22,

[4] L.G.Valiant, A Bridging Model for Parallel Computation, CACM 33 (8): 103-111, 1990

[5] S.Peyton-Jones, Implementation of Functional Programming Languages, Prentice-Hall, 1987

[6] K. M. Chandy & J. Misra, Parallel Program Design:A Foundation, Addison-Wesley, 1988