计算机科学 1994 Vol. 21 №. 4

可视语言：把思想变成程序的工具

强 军 王兵山 李舟军

（国防科技大学计算机系 长沙 410073）

摘 要 可视语言是计算机科学和技术的结晶，是一个新兴的研究领域。它与传统的程序设计语言相比较具有明显优势。本文介绍可视语言的发展背景，探讨其主要研究内容，并介绍几个较典型的实现系统。

关键词 可视语言，语法，语义，实现系统，程序设计环境。

一、引 言

在计算机科学研究中，一个最重要的问题是向人们提供什么样的方法或工具，能方便且准确地描述客观世界中要解决的问题并用计算机进行问题求解。人们意识到，对问题进行规范说明的语言和工具是软件开发过程中的关键，同时还希望在正确的规范基础上能实现某种程度上的软件生产自动化。

十多年来，人们开发出了多种规范说明语言，如 VDM Meta-N，Z，Larch，Clear 等，由于它们需要较深的数学基础，难于普及应用。可执行规范、快速原型，程序变换等软件工程新范型取得了一定的成功，推动了软件生产自动化的研究，但还没有解决自然直观地描述客观世界的问题。针对于此，当前有两种途径：一是开发一种类似于（受限制的）自然语言的规范说明语言；二是借助于图形引喻，进行可视语言（Visual Languages）的研究和实现。由于人类的思维是强烈地倾向于图形的，后者可望具有光明的前景。

二、可视语言的发展背景

几十年来，计算机科学基本上是以计算机为中心而发展的，现在其主流转移到怎样使计算机系统更适应用户上来。

可视语言来源于图形界面设计和程序可视化（Program Visualization）。用图标（icon）代替命令作为操作系统用户界面（如早期的 MAC 机）和软件工具用户界面（如 MAC DRAW），其富有语义暗示的图形令人耳目一新。另外，为了更直观地把握程序，人们做出这样一些工具，用图形把数据结构和程序控制结构显示在屏幕上，当程序执行时，对正在活动的程序对象的对应图形作明暗变化和色彩变化，从而可以“感知”到程序的动态行为。实践证明这种对程序的动画模拟尝试取得了极好的效果。有了这些成功，人们进一步联想到，既然我们经常用有限状态自动机、框图、DFD 表示算法和程序，然后再把它们翻译（细化）成某种计算机可以接受的语言，那么为什么不直接用这些图形作为程序设计语言呢？

任何一种新事物都是时代的产物。可视语言也不例外。我们可以看到以下几条促使可视语言诞生和发展的原因：

1. VLSI 技术、微处理器技术、视频技术和计算机图形学的发展使高速、高分辨率工作站为越来越多的人在经济上能够接受。以彩色显示器和指示设备（鼠标，触摸屏等）为媒体，人机之间可以进行图形方式的交互，这是硬件方面的基础。

2. 多窗口系统（如 X Windows，Microsoft Windows）提供的覆盖式多窗口、图形函数模板、icon 编辑器等工具使得可视程序设计环境的开发十分方便，友好界面的生成也很容易。构造可视程序设计环境所需的工程数据库和面向对象数据库技术也达到了实用程度。这是软件方面的物质基础。

3. 八十年代以来，软件工程领域和程序设计自动化领域对快速原型、可执行规范和程序变换进行了较深入的研究，并在实践中作了许多尝试，取得了许多经验，特别是 O-O 语言的思想和技术为可视语言的实现提供了方法论上的指导。

强军 工程师，硕士，研究兴趣是可视语言和软件规范语言。王兵山 教授，博士副导师，研究兴趣是计算机科学理论和软件生产自动化。李舟军 讲师，硕士，研究兴趣是形式语义、逻辑和面向对象技术。

4."软件危机"多年未决，促使人们积极地探索新途径。可视语言以崭新的面貌和易于形象理解，易学易用的优点，受到越来越多的青睐。这是可视语言的内在魅力，也是它的内在发展动力。

目前，可视语言的研究和应用都在向深度和广度不断发展。据权威人士预测，在本世纪剩下的几年里，硬件方面主要发展方向是多处理器(MPP)和多媒体(Multi-media)，软件设计的主要方法是面向对象(O-O)。可视语言正好顺应这一潮流，可以充分利用它们的成果，并把它们推向更广泛的应用，与之相得益彰。

三、可视语言的优势

相对于传统程序设计语言而言，可视语言具有明显的优势。下面就三个方面进行比较。

1. 语言的表示能力

①表示维数　正文语言是一维的字符流，通过上下文语法结构表示其意义。可视语言至少是二维的(有人把带有 zoom in/out 机制的可视语言看作是2.5维的)并可以用形状、大小、颜色、文字注解、指向、距离等表示其意义。图形能够自然地提供丰富直观的引喻(metaphors)，从而可以方便地反映客观世界及其抽象；正文只能在某种程度上描述客观世界。我们每个人都有过这样的体验：在阅读一个正文程序的时候，往往费尽周折，最终要到在头脑里形成一个形象的概念模型时才认为是把握住了它。可视语言程序可以就是问题的概念模型的图解，或其细化。表示维数的不同决定了表示级别的差异。

②信息熵　俗话说"一图顶千字"，可见人类的思维是强烈地倾向于图形的。正文程序中有意义的基本单位是字符或字符串，前后之间的上下文关系和相互间的逻辑关系需要反复互相参照，认清一个对象或问题要读大段正文程序，熵值较低。可视程序中有意义的基本单位是方框、圆、线段、箭头等图素(graphical elements)，以及少量的文字注解，通常直接对应于概念模型中的对象及对象之间的关系，具有更高级别的语义，能以较少的符号表达较多的信息，熵值较高。

③运行情况　程序本质上是动态的，要掌握一个程序的细节，最简单的办法就是观察它的执行情况。正文程序的运行情况一般是通过 debuger 程序来观察，根据指明的当前执行的语句行和断点环境数据进行判断。可视程序运行时，可以在程序本身之上作动画模拟，用图形对象的色彩变化和亮度变化表示程序的控制流程和当前活跃情况，并把各个对象的有关数据同时显示出来，一目了然。这种动画模拟的突出优点是可以形象表示并行程序的执行过程。

2. 对概念程序设计的支持

在以往的软件开发过程中，设计者弄清了需求以后，首先在头脑中形成一个待开发系统的概念模型，然后再逐步求精、细化，直至翻译成某种高级语言。在软件调试和维护过程中必须在这种语言的程序和设计者的概念模型之间进行多次的来回翻译，大量的精力被浪费在与设计无关的工作中。在可视语言程序设计环境中，设计者可以用图形把头脑中的概念模型"画"一张图表示出来，这张图就是可视程序。可视程序再由系统支持变换成可执行映象。对设计的修改是通过对可视程序的修改来完成的。用可视语言进行软件开发的过程中，主要关心的是设计待开发系统的概念模型并把这个模型用恰当的图表示出来，不必陷入高级语言的细节之中。从某种意义上说，可视程序设计语言实际上起的是规范描述语言的作用，而把更多的工作交给机器去做。

3. 对抽象的支持程度

要用计算机解决现实世界中的问题，必然要用到抽象手段。无论是写程序还是读程序，关键是要搞清抽象的层次结构。正文程序由多个过程和函数构成，这些过程和函数顺序写下来，从中较难看出抽象的层次结构。可视程序可以直接反映抽象层次结构，用方框、圆等可视对象表示现实世界中的实体或抽象对象，用连线或转移表示实体或抽象对象之间的联系，用 zoom in/out 表示抽象的层次结构。事实上，采用多窗口技术可以同时在几个层次上观察程序。系统中还可以支持不同抽象角度对软件的视图。

可见，可视语言易理解、易掌握、易编程、易维护，具有正文语言无可比拟的优越性。国际上可视语言研究者们认为，可视语言的研究与发展必将对计算机软件产生重大影响，甚至带来根本性变革。

四、主要研究内容

目前，可视语言研究领域涉及的范围还在不断扩展之中，对"可视语言"这一概念还没有统一的定义，但有一点共识，即该领域是在软件工程、程序设计自动化、计算机图形学、认知心理学等领域的基础上发展起来的，是交叉学科。大致上说，可视语言是以可视的形式表示计算过程中的对象、概念和过程的计算机语言。

可视语言的实现，一般是指用这种可视语言写的程序可以被计算机解释执行或生成可执行映象。这要求对可视语言进行严格的语法描述和语义定

义，并定义从语法结构到语义的映射。我们认为，一个较完备的可视语言实现系统有如图 1 所示的几大模块：

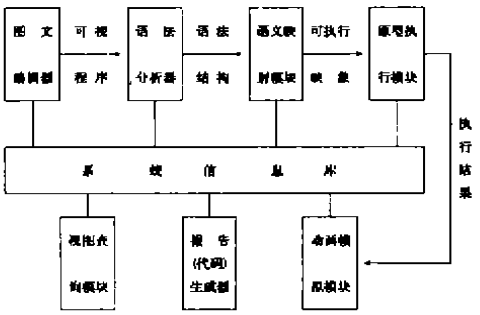

图 1　可视语言实现系统总体结构

这样就构成了一个可视语言程序设计环境，或者说实现了一台可视语言的虚拟机。图 1 的上半部分是系统内部对可视程序进行处理，下半部分是给用户的反馈信息处理。

1. 可视语言的基础理论研究

可视程序设计语言要在将来的软件生产中起重要作用，还要解决如下问题：

①为该领域建立一个坚实的科学基础。例如可视语言的复杂度理论这一公开理论问题：在设计出满意的对可视语言进行规范说明的语言及其相应的可视程序识别系统后，我们缺少讨论这种规范说明语言的表示能力的方法[3]。希望有类似正规文法和有限状态自动机那样的结果。

②对大型程序设计的支持。可视程序设计对程序设计初学者和小型程序的支持是没有疑问的，但人们会有这样的疑问：可视语言程序设计环境对熟练的程序设计人员是否有帮助？可视语言程序设计环境是否支持大型程序设计？文[2]中认为，只要环境中有足够的工具支持，那么答案是肯定的。

③为程序、数据类型和数据结构开发一种共同遵循的图文混合表示方法。

④定义和确认度量标准(metrics)，以便在不同的可视语言程序设计环境之间进行比较评价。

2. 语法说明规范

可视语言是二维的，其语法描述比正文语言要复杂得多。一般地是先定义构成可视语言的基本图形元素集合和图形之间的空间关系(spatial relation or spatial operator)集合，然后再以这两个集合为基础用恰当的文法描述可视语言的各语法单位。目前，各种规范百家争鸣，比较有代表性的有 Picture Layout Grammer[4]、Relational Grammer[5]、Positional Grammer[6]等。我们还不知道，可视语言的语法说明规范应具有哪些性质。

3. 语法分析

早期的可视语言的实现大多用语法制导编辑器，现在几乎所有的可视语言实现系统都强调通过语法分析得到程序的内部结构的重要性。

语法分析器的任务是识别出可视程序的语法结构。完成这一任务相对于正文程序的语法分析来说要困难得多，主要表现在以下几个方面：

①从图文编辑器得来的输入是原始图素的集合，语法分析器需要根据这些原始图素的物理信息(空间位置座标、色彩等)，判定相互之间的关系，识别出具有一定逻辑意义的 token 集合。这个工作相当于正文语言程序的词法分析。

②token 之间不存在明显的序关系。正文语言程序中，无论是字符之间还是 token 之间都存在着线性序关系，这是应用下推自动机构造语法分析器和句柄概念的基础。对于二维可视语言，为了处理这种无序性，有人通过限定 operator 集合为水平邻接、垂直邻接和覆盖来方便地构造 token 之间的线性序关系，以利用 LR 分析器；有人寻找构造 token 集合上的偏序关系的一般方法，以利用 Earely 算法；还有人利用人工智能技术，用文法描述的重写规则作为驱动，进行组合、归约和回溯，直至分析成功或发现错误。

③一般地，可视语言(如 Petri 网、StateChart 等)是上下文相关的，要对特定的可视语言构造高效的分析器，需要有足够的上下文信息甚至全局信息，这就需要预处理过程，以及系统信息库的支持。

④不同于正文语言程序以分析树作为语法分析的结果，可视语言程序的语法分析的结果是一张分析图。

4. 语义表示方式

由于语法说明规范未取得统一，影响到了语义表示的研究。目前可视语言有多种语义表示方式，大致分为传统的形式语义(主要用操作语义)和非形式语义两大类，根据不同的可视语言和系统目标进行选择。

操作语义适用于面向 icon 集合的可视语言，对每个 icon 给出一个语义子程序，该 icon 被激活时就调用它的语义子程序，完成原型执行的任务或生成一段目标语言代码。操作语义也适用于一些用原始图素构成的可视程序设计语言，主要用于生成目标语言代码。

近年在实现中用得较多的非形式语义有说明语义(declarative semantics)，求值语义(evaluation semantics)。在实现系统中强调把可视程序作为可执行规范时，可将该程序映射成某种已实现的高级语言写的语义解释程序：如 Prolog 程序，Lisp 程序，C++程序等，然后利用相应的执行机制方便地完成解释执行。

如果系统既要求做原型执行又要求生成目标代码，则可以同时用上述两种语义表示方法，或加以合成。

面向对象的语义表示方式，可以作为在一个系统中实现多种可视语言时一致的内部语义框架[7]。面向对象的表示方式，以其与客观世界的自然对应这一特点，符合概念程序设计的要求，易于建立从语法结构到语义的映射。

5. 可视程序设计支持工具

友好的用户界面(窗口，菜单，按钮，命令 icon 等)；

支持多种可视语言的通用图形编辑器，以及可视对象实例化模板(通常以对话框的形式给出)；

可视程序的多种静态视图和动态视图，如语法分析图，内部数据结构视图(名，类型，值等)，程序的抽象表示视图，运行时的动画模拟，debuger 跟踪和断点检查等。这些视图都要求系统中有一个信息(数据)库作为支持；

支持大型程序设计的项目管理、模块机制、文档生成等 CASE 工具；

对有并发性的可视程序提供模型验证工具，等。

五、可视程序设计环境和可视语言应用系统

一种在计算机上表示人的思想的语言，只有在机器上实现了并得到了广泛的应用才是有意义的。国外已有许多实现可视语言的研究和工程，对实现技术和应用领域进行了探索。右表列出一些较有代表性的系统。

下面简要介绍其中的几个系统。

1. PECAN，GARDEN 和 VPW。这三个系统是 Brown 大学在程序可视化和可视程序设计方面十多年努力的结果。分析它们可以看到可视程序设计环境的发展过程和努力方向。

PECAN 是 Brown 大学早期(84 年)的尝试。它实际上是一个 PV 系统，正文语言程序给出多种语义视图，如表达式树、数据类型图、流图等，并在程序执行的时候通过其数据结构中内容的变化给出其动态视图。在程序被修改或程序在执行时，系统自动维护各视图的一致性。

为了支持概念程序设计，必须在一个环境中甚至一个程序中能同时支持多种可视语言，如 DFD、框图、有限自动机等。同时在一个统一的内部语义框架之上维护可视程序多种视图的一致性，从而使 VP 和 PV 相辅相成，得到淋漓尽致的发挥。

系统名称	系统特性	可视语言	研制者
PECAN	提供程序的多种视图	……	[美]Brown 大学
GARDEN	多种可视程序设计语言环境	多种可视语言	[美]Brown 大学
VPW	可视程序设计环境开发平台	多种可视语言	[美]Brown 大学
SIL-ICON	面向 icon 的可视语言环境	icon 集合	[美]Pittsbug 大学
HI-VISUAL	面向 icon 可视程序设计环境	icon 集合	[日]广岛大学
VLG	面向 icon 的系统开发环境	icon 集合	[意]Salerno 大学
ECASET	可视规范和原型生成系统	E-R 图，DFD	[意]米兰研究院
ENVISAGER	实时系统规范环境	icon/时态逻辑	[美]Southwesten Louisiana 大学
* Layout	根据流程图生成 C 代码	Flow Chart	[美]Matrix 公司
* STATEMATE	复杂反应式系统的开发环境	Model-Chart Activity Chart StateChart	[以]Welzmann 研究院 [美]i-Logix 公司
Miro	系统安全性可视规范工具	Miro 语言	[美]Carnegie-Mellon 大学
KEATS	构造基于知识系统的软件支撑环境	语义网络	[英]Open 大学
* Explore	科学计算可视化工具界面	icon 集合	[美]SGI 公司

注：带 * 者已商品化。

GARDEN 就是以上思想的一个实现。该系统的主要特点是为支持多种可视语言而提供一个面向对象的内部语义框架 Object Graph[7]，并为对象定义一种执行机制——求值(evaluation)，即该对象对给定的参数活动一次的结果，改变内部私有变量，向其它对象发送消息等。这样，该系统鼓励对问题进行抽象，达到较高程度的软件重用，并特别适合原型方

法。

VPW(Visual Programming Workbench)是可视语言程序设计环境的快速综合工具包[8]。它集中了PECAN和GARDEN的优点，并采用了许多先进的软件技术。VPW的一个突出特点是力图用PLG(Picture Layout Grammar)作为多种可视语言的统一的语法说明规范。VPW功能上主要由四大部分组成：

语法结构：用PLG文法给出可视语言的规范，对每个可视程序用系统的语法分析程序分析得到可视程序的语法结构——分析图(parse graph)；

抽象结构：把分析图转变成一致的抽象结构——对象图，作为对该可视程序进行静态语义处理和动态语义处理的基础；

静态语义：对可视程序静态性质的抽取，分析或综合，产生各种静态视图；对并发程序进行模型验证；

动态语义：可视程序的解释，模拟和动态验证(可达性，不确定性等)。VPW还支持多处理机的分布式计算。

2. SIL-ICON Compiler，是一个对面向icon系统进行规范、解释、原型和生成的软件系统[18]。系统由icon库ID、操作子库OD、ETAG文法和解释程序Icon INTERPRETER组成。用户可根据应用自己定义新的icon(图形表示，逻辑语义子程序)。Icon之间的空间关系仅限于水平邻接、垂直邻接、覆盖三种。语法分析是把icon之间的二维空间关系转变成一维形式，再用LR分析法完成，得到分析树。可视程序的原型执行是通过调用icon的语义子程序来完成的。

3. KEATS(Knowledge Engineer's Assistant)系统。是一个应用VP和PV技术的、支持知识工程生命周期的集成软件环境[19]。该环境中有多种工具，可支持知识工程的主要阶段：

知识获取阶段：采用可以产生代码的超级文本，配以多种视图，用语义网络使专家知识可视化；

知识编码阶段：多种工具之间紧密耦合，有助于对知识进行编码；

调试阶段：运行时可以从不同的侧面看到粗略的视图，也可以看到小粒度视图。

六、结束语

自1984年第一届IEEE Workshop on Visual Languages召开以来，可视语言正式成为计算机科学的一个相对独立的研究领域。人们逐渐认识到了可视语言对软件界可能产生的巨大而深刻的影响。国际上可视语言的研究和实现正在向深度和广度进军。国内尚未见有关可视语言研究和实现的报道，亟须我们作出努力。

两位著名的计算机科学家，D. Harel和A. Pnueli在文[10]的结束语中的两句话值得我们深思：

"我们相信，在对计算机科学的不断研究和探索过程中，可视形式(visual formalisms)将成为至关重要的、自然而有力的方法。"

"我们相信，(在竞争淘汰中)存在下来的方法将是具有图形本质的，同时是形式化的且具有严格的语法和语义的方法。"

参考文献

[1] G. Reader, "A Survey of Current Graphical Programming Techniques", IEEE Computer, Vol. 18, No. 8, 1985, pp11-25.

[2] Proceedings of COMPSAC'86, Panel Session on Visual Programming: Possibilities and Limitations, pp404-411.

[3] A. L. Chow et al., "Toplogical Composition Systems: Specifications for Lexical Elements of visual languages", 1991 IEEE Workshop on Visual Languages, pp118-124.

[4] E. J. Golin et al., "The Specification of Visual Language Syntax", 1989 IEEE Workshop on Visual Languages, pp105-110.

[5] C. Crimi et al., "Relational Grammars and their Application to Multi-dimensional Languages", Journal of Visual Languages and Computing 2, pp333-346.

[6] S. K. Chang, "Principles of Visual Programming Systems", Prentice-Hall, 1990.

[7] S. P. Reiss, "Working in the Garden Environment for Conceptual Programming", IEEE Software, Vol. 4, No. 6, NOV. 1987, pp16-27.

[8] R. V. Rubin et al., "Early Experience with the Visual Programming Workbench", IEEE Transactions on Software Engineering, Vol. 16, No. 10, OCT. 1990, pp1107-1121.

[9] M. Eisenstant et al., "Visual Knowledge Engineering", IEEE Transaction on Software Engineering, Vol. 16, No. 10, 1990, pp1164-1177.

[10] D. Harel et al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", Proceedings of the tenth IEEE International Conference on Software Engineering, 1988.