第 35 卷 第 2 期 计 算 机 工 程 2009 年 1 月
Vol.35 No.2 Computer Engineering January 2009
·软件技术与数据库· 文章编号：1000—3428(2009)02—0044—03 文献标识码：A 中图分类号：N945

语法制导生成的虚拟仪器开发平台

随阳轶[1]，林 君[1]，张晓拓[2]，王世隆[1]
(1. 吉林大学仪器科学与电气工程学院，长春 130026；2. 吉林大学通信工程学院，长春 130022)

摘 要：如何以一种与特殊的图形用户接口无关的形式一致地描述数据流可视化编程语言及其开发平台，成为虚拟仪器开发平台自动生成的主要问题。该文以自主研制的开发平台 LabScene 为例，形式化地描述其中最主要的 3 个部分：编辑，运行，调试，把这些形式化的描述翻译成 C#语言，实现自动生成，为以非数据流为核心的虚拟仪器开发平台的自动生成提供参考依据。
关键词：虚拟仪器；数据流可视化编程语言；自动生成

Virtual Instrument Development Environment Generated by Syntax-directed Method

SUI Yang-yi[1], LIN Jun[1], ZHANG Xiao-tuo[2], WANG Shi-long[1]
(1. College of Instrument Science and Electronic Engineering, Jilin University, Changchun 130026;
2. School of Communication Engineering, Jilin University, Changchun 130022)

【Abstract】The key problem of auto-generation of virtual instrument development environment is how to congruously describe DataFlow Visual Programming Language(DFVPL)and its environment in a way independent of a particular GUI implementation. With the example of virtual instrument development environment, LabScene formally describes three most important parts: edition, execution and debugging, and translates them into C# language to realize auto-generation of virtual instrument development environment. The method is the reference of auto-generation of virtual instrument development environment which is not build on dataflow.
【Key words】Virtual Instrument(VI); Dataflow Visual Programming Language(DFVPL); auto-generation

目前主流的虚拟仪器[1]软件开发是在以数据流可视化编程语言(Dataflow Visual Programming Language, DFVPL)[2]为核心的开发平台中完成的，如NI公司的LabVIEW与Agilent公司的VEE。以DFVPL为核心的虚拟仪器开发平台自动生成的关键问题有 2 个：(1)以某种简洁、准确的方式一致地描述DFVPL本身及虚拟仪器开发平台。(2)要求与特定GUI的实现无关[3]。

1 LabScene

LabScene是基于DFVPL的虚拟仪器(Virtual Instrument, VI)开发平台，它不仅提供了具有仿真能力的仪器面板，还将电路图的基本元素映射到数据流模型中，从而完整地表达了虚拟仪器的设计。电路图的基本元素可以抽象为 6 种[4]：基本元件，芯片，管脚，连线，交叉点，线路板，如图 1 所示。经过映射后，在LabScene中元件视为功能节点，芯片及线路板视为容器节点，其他仍然视为管脚、连线、交叉点。因此，硬件的电路图被映射成可用数据流模型表示的程序，即不同种类的节点以及连接节点的线组成的集合。

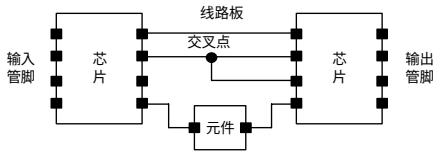

图 1 电路图模型

2 形式化描述

2.1 结构定义

对于 LabScene，一个 VI 包含 2 个部分：与用户进行数据输入输出交互的 UI(User Interface)和用户设计时使用的数据流可视化语言 DF。UI 由组件集和放置这些组件的面板组成，DF 除了自身的标志外，就是节点和线。组件由输入/输出控件和关联节点的 NodeFlag 组成。节点可以分为功能节点和容器节点。功能节点又可以分为完成某个特定功能的基础节点和包含另一个完整 DF 的子 VI 节点。功能节点包含输入/输出管脚的集合。容器节点具有控制功能，可以包含功能节点或另外的容器节点。Design 是一个特殊的容器节点，它是创建 VI 时生成的最上层的容器节点。线由输出管脚和输入管脚的有序集表示。节点和线都有自己的图形化表示，比如形状、位置。下面是相关的核心产生式：

<VI> ::= <UI>;<DF>
<UI> ::= <Components>;<Panel>
<Components> ::= {<Component>}
<Component> ::= <Control>;<NodeFlag>
<DF> ::= <DFFlag>;<Nodes>;<Wires>
<Nodes> ::= {<Node>}
<Wires> ::= {<Wire>}
<Node> ::= <FuncNode>|<ContainerNode>;
<NodeFlag>;<Graphic>;
<State>;<Attribute>
<FuncNode> ::= <PrimitiveNode>|<SubVI>;

基金项目：教育部科学技术研究基金资助重点项目(106060)
作者简介：随阳轶(1980－)，男，博士研究生，主研方向：可视化编程技术，虚拟仪器；林 君，教授、博士生导师；张晓拓，助理工程师；王世隆，硕士研究生
收稿日期：2008-09-01 **E-mail：**suiyangyi@163.com

```
<Icon>;<InPins>;<OutPins>
<PrimitiveNode> ::= <Name>
<InPins> ::= {<InPin>}
<OutPins> ::= {<OutPin>}
<InPin> ::= <I>;<PinID>;<Wire>;
<Data>;<DataValid>
<OutPin> ::= <O>;<PinID>;<Wire>;
<Data>;<DataValid>
<ContainerNode>::=<Design>|<For>|<While>|<Case>|
                  <Sequence>;<Panel>;<DF>
<Design> ::= <Name>
<For> ::= <LoopCount>
<While> ::= <LoopCondition>
<Case> ::= <CaseSelector>;{<DF>}
<Sequence> ::= {<DF>}
<Wire> ::= <OutPin>;<InPin>;<Graphic>
<State> ::= FIRED|RUNNING|COMPLETE
```

其中，分号表示聚合，即关联一端的对象实例是另一端对象实例的组成部分；大括号表示集合；产生式右边未在“ <> ”内的元素表示基本类型(如 INTEGER)、枚举(如 FIRED)或特殊类型(如 GUID)。这样可以很容易地描述信息，比如 DFVPL 的可视化部分要么是节点要么是线：

WholeDF(df:DF) == {x | x∈df.Nodes ∨ x∈df.Wires}

其中，参数表达式 df:DF 表示 df 是 DF 的一个实例；点号表达式表示一个组成部分的子组成部分。

2.2 编辑

编辑是指改变程序结构的操作，例如：增加/删除控件，节点或线，改变控件，节点或线的属性。一个语法制导的编辑操作就是把已有的 VI 进行编辑操作，产生一个新的、语法构造仍然正确的 VI。下面以一个最常用的编辑操作说明。

生成 VI：创建一个 VI 可以通过生成一个不包含组件的 UI 和包含唯一容器节点 Design 的 DF 来实现：

CreateVI == [vi = new(VI) ∧ ui = new(UI) ∧ uipl = new(Panel) ∧ df = new(DF) ∧ ds = new(Design) ∧ dspl = new(Panel) ∧ vi.DF'= df ∧ vi.UI'= ui ∧ ui.Panel' = uipl ∧ ds.Panel' = dspl ∧ df.Nodes' = {ds}]

其中，new(X)是一个函数，它返回由 X 表示的类型的实例；元素右上角的“ ' ”表示元素在操作前后的状态转换。

2.3 运行

运行谓词描述程序状态在执行中可能的改变，比如点燃节点。本文的例子只描述了执行中的少量细节。

点燃功能节点：如果某个功能节点所有输入管脚的数据都有效，则状态转换为点燃：

Fire(fn:FuncNode) == [∀ ip:fn.InPins.InPin [ip.DataValid = TRUE] => fn.State' = FIRED]

2.4 调试

调试就是在程序执行过程中把 VI 中各个部分的执行状态呈现给用户，帮助用户修改设计、检查错误。相关状态的改变一般体现为图形化属性值的改变，如字体、高亮、闪烁。

高亮所有点燃的节点：可以通过节点图形化的部分来反映节点的运行状态：

DisplayFired(df:DF) == [∀ node: df.Nodes. Node[node.State = FIRED ⇔ node. Graphic. Highlight = TRUE]]

3 语法制导的翻译

语法制导的翻译就是把 VI 的抽象语法转换成正确的数据类型，以及将谓词中给出的说明性描述转换到一个程序的形式。目标语言是 C#以及.NET 的框架类库(FCL)所提供的图形化接口。

3.1 产生式的翻译

产生式表示的 LabScene 的数据结构被转换到 C#的类中。通过分析产生式来产生需要的类和成员。产生式中的“ 或者(|) ”被映射为类和子类的关系，“ 聚合(;) ”映射为一个类的成员变量。举例来说，Node 就是 FuncNode 和 ContainerNode 的父类，Node 有 4 个成员变量，其类型分别是 NodeFlag, Graphic, State, Attribute。节点的类体系结构如图 2 所示。

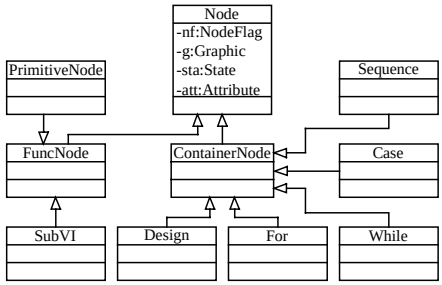

图 2 节点的类体系

3.2 编辑谓词的翻译

编辑谓词由连接的等式组成。每个等式被翻译为一个赋值语句。语句的执行顺序即谓词的连接顺序。如果谓词的连接顺序是任意的，那么在翻译过程中必须保证对象实例化在对象使用之前完成。集合的操作映射到 C#的列表操作上。下面以生成 VI 为例说明编辑谓词的翻译：

```
void CreateVI()
{VI vi = new VI();
      UI ui = new UI();
      Panel uipl = new Panel();
      DF df = new DF();
      Design ds = new Design();
      Panel dspl = new Panel();
vi.UI = ui;
vi.DF = df;
      ui.Panel = uipl;
            ds.Panel = dspl;
 df.Nodes.Add(ds);   }
```

3.3 运行谓词的翻译

运行谓词可以翻译为条件到行为的形式。条件部分测试各种状态，行为部分改变状态或执行函数，将某个集合中元素的遍历都翻译成 foreach 语句。下面以点燃功能节点为例进行说明：

```
void Fire(FuncNode fn)
{ bool fired = true;
  foreach(InPin ip in fn.InPins)
  {if(!ip.DataValid) {
fired = false; break;          }       }
  if(fired) fn.State = FIRED;}
```

3.4 调试谓词的翻译

调试谓词的翻译与运行谓词的翻译类似。只是“ ⇔ ”的翻译需要体现出当且仅当的唯一性。以高亮所有点燃的节点来举例说明：

```
void DisplayFired(DF df){
      foreach(Node node in df.Nodes)
      { if(node.State == FIRED)
node.Graphic.Highlight(true);
```

```
else node.Graphic.Hightlight(false);      }}
```

3.5 自动生成算法

以自动分析产生式、谓词等实现前面例子中说明的翻译：

(1)生成产生式的符号表T_o，T_o的每项包括元素名、父类名、成员变量的类型名。

(2)解析每个产生式，为左部和右部每个元素名在T_o中生成一项(已经生成的不重复生成)。分析产生式的结合方式，回填T_o中的项。如果元素通过“或(|)”结合，则产生式左边的类是元素的父类。如果元素通过“聚合(;)”结合，则元素是产生式左边的类的成员变量。遍历T_o可以完成类体系的建立和类的定义。

(3)生成谓词的符号表T_e，T_e的每项包括：谓词名，输入/输出变量集，局部变量集。每个变量又包括：指向T_o中类型名，变量名，输入还是输出。

(4)为每个谓词名在T_e中创建一个项。解析每个谓词，回填T_e中的项。进入谓词的变量为输入/输出变量，谓词中的其他变量为局部变量。遍历T_e以完成函数的定义，如果是等式连接则按连接顺序产生赋值语句。如果是蕴涵则生成测试表达式，并调用目标语言的函数。

4 实验

为了验证语法制导自动生成的正确性，设计的实验如图 3 所示。

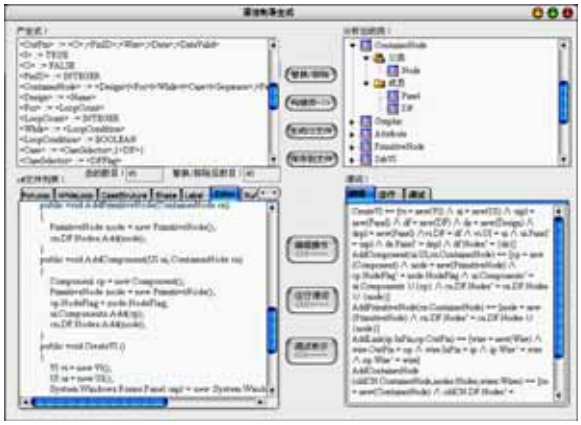

图 3 实验

实验过程如下：

(1)输入 45 个产生式，然后把产生式中复数形式的元素名用集合形式替换，比如替换后<DF>::= <DFFlag>; {<Node>}; {<Wire>}。如果元素名和 C#的关键字冲突，就修改元素名。这样得到 40 个产生式。

(2)分析得到的产生式，得到 18 个类以及它们的成员和父类。

(3)根据这些类的信息生成相应的.cs 文件，同时生成单独的 3 个.cs 文件：Editor.cs, Runner.cs, Debugger.cs。

(4)输入编辑、运行和调试的谓词集合，把分析这些谓词所得到的函数加入到相应的.cs 文件中。

(5)保存所有的.cs 文件，使用微软提供的 C#编译器进行编译，看能否通过。

按上述步骤进行实验，所有的.cs 文件都通过了编译器的验证，从而证明了本文的设计是可行的。

5 结束语

虽然自动生成虚拟仪器开发平台要考虑的细节远比给出的例子复杂，但是本文给出了自动生成该平台的高级抽象。这种与特殊 GUI 无关的抽象一致地描述了 DFVPL 和虚拟仪器平台。针对不同的语言或图形化包的翻译，可以得到外形和交互完全不同的虚拟仪器平台。修改 DFVPL 的定义也可以得到以非数据流为核心的虚拟仪器开发平台的高级抽象，从而帮助实现自动生成。

参考文献

[1] Goldberg H. What is Virtual Instrumentation?[J]. IEEE Instrumentation & Measurement Magazine, 2000, 3(4): 10-13.

[2] Johnston W M, Hanna J R P, Millar R J. Advances in Dataflow Programming Languages[J]. ACM Computing Surveys, 2004, 36(1): 1-23.

[3] Kleyn M F, Browne J C. A High Level Language for Specifying Graph Based Languages and Their Programming Environments[C]// Proc. of the 15th International Conference on Software Engineering. [S. l.]: IEEE Computer Society Press, 1995: 324-335.

[4] 谢宣松, 随阳轶, 林 君. G 语言的硬件虚拟模型[J]. 仪器仪表学报, 2006, 27(9): 1112-1115.

(上接第 43 页)

上述流程是对于正常时间闸数据交换处理流程的增强(为简洁起见，省略了多个时间闸检查环节)。其中增加了放行令牌的检查，包括数据业务宿主检查、令牌许可放行时间闸检查和令牌许可数据业务时间闸检查，并提供 2 种放行模式的处理，以满足绕开正常时间闸控制实施特殊放行的数据交换需求。

另外，对于放行时间闸($rt1$, $rt2$)过期的令牌(称为过期令牌)，数据交换系统将通过额外的机制定期予以清除。

6 结束语

数据交换技术是大型信息系统体系架构中一项重要的基础技术，其应用十分广泛。实际企业生产经营管理过程中的许多业务对数据交换提出了安全性更高灵活性更大的基于动态时间限制的数据交换控制和许可要求。

本文提出的基于时间闸和放行令牌的数据交换技术是有效解决这一问题的一种通用方法，可应用于各种具体的数据交换系统中。该数据交换技术已在南瑞 PX2000 数据交换平台中得以实现。

参考文献

[1] 胡良亚, 肖卫东, 何晓晔. 基于 XML 与 J2EE 技术的数据交换中心的设计与实现[J]. 计算机应用研究, 2003, 20(1): 70-72.

[2] 程 渤, 浮花玲, 杨国纬. 基于工作流及集成中间件技术的电力信息一体化设计及实现[J]. 电力系统自动化, 2004, 28(19): 80-83.

[3] 曾 伟, 杨微波, 杨绣勇, 等. 基于网络技术的变电运行信息管理系统的设计[J]. 电力系统自动化, 2003, 27(19): 91-93.

[4] 邓兆云, 张建平. 电力调度生产管理信息系统的工作流系统[J]. 电力系统自动化, 2003, 27(16): 78-80.

[5] 朱六璋, 李 晶, 黄太贵, 等. 安徽电网调度管理信息系统建设[C]//2004 全国电力系统自动化学术交流研讨大会论文集. 南京: 国电自动化研究院, 2004: 490-496.

[6] 孔 震, 林 峰, 俞 骏. PI2000 工作流引擎的设计与实现[J]. 电力系统自动化, 2003, 27(21): 75-78.