文章编号:1671-4598(2023)03-0187-06 DOI:10.16526/j.cnki.11—4762/tp.2023.03.027 中图分类号:TP314 文献标识码:A

基于图形化热工水力测控程序的解释性加载设计

向友洪,廖芳芳,付 玉,程 杰,王建宇
(中国核动力研究设计院,成都 610213)

摘要:在热工水力测控系统中,常采用图形化编程的方式构建编程控件之间逻辑关系,形成所需的系统功能;程序的部署和运行时,若采用程序实时编译的模式,会带来编译时间长、编译环境复杂等问题;为了提高测控程序的开发效率,采用控件组件化的思想,进行图形化程序逻辑关系的分解与重构机制设计;提出程序模板+轻量级配置文件的模式,利用程序模板对配置文件进行解释性加载;在热工水力测控系统的测试验证中,利用基于解释性加载机制的图形化编程平台完成了其中测控软件的开发及运行,功能及数据结果准确无误;该解释性加载机制真实、有效,可实现图形化热工水力测控程序的快速部署与加载运行。

关键词:热工水力;测控程序;图形化编程;解释性加载;程序模板

Explanatory Loading Design Based on Graphical Thermal Hydraulic Measurement and Control Program

XIANG Youhong, LIAO Fangfang, FU Yu, CHENG Jie, WANG Jianyu
(Nuclear Power Institute of China, Chengdu of Sichuan province, Chengdu 610213, China)

Abstract: In thermal hydraulic measurement and control system, graphical programming is often used to construct the logical relationship between programming controls to form required system functions. If the real-time compilation mode of program is adopted during the deployment and operation of program, it will bring problems such as long compilation time and complex compilation environment. In order to improve the development efficiency of measurement and control program, the idea of componentization is adopted to design the decomposition and reconstruction mechanism of graphical program logical relationships. a program template and lightweight configuration file mode is proposed, and the program template is used to explanatorily load the configuration file. In the test and verification of the thermal hydraulic measurement and control system, the graphical programming platform based on the explanatory loading mechanism was used to complete the development and operation of the measurement and control software, and the function and data results were accurate. The explanatory loading mechanism is real and effective, which can realize the rapid deployment and loading operation of the graphical thermal hydraulic measurement and control program.

Keywords: thermal hydraulics; measurement and control program; graphical programming; explanatory loading; program template

0 引言

随着计算机技术和编程技术的发展,图形化编程语言凭借其操作简便、易读性好等优点,形成了一系列有代表性的产品,如NI公司的LabView平台[1]和Wonderware公司InTouch等工控组态软件。针对热工水力测控系统,其一般具备测点众多、功能重复使用率较高的特点,因此,为了实现测控软件系统的快速搭建、运行,基于模块化思想的图形化编程[2-3]方式成为了重要的实现手段。

在热工水力测控领域中,不同测控程序之间往往具有分布式运行的部署特点,程序的编制和程序的运行一般处于不同的终端上。而将图形化测控程序从程序开发终端部署到程序运行终端,需要经历代码到跨平台可执行程序的编译转化过程[4-5]。不同运行终端环境需要对应的编译器去进行编译操作,否则会影响程序的正确运行[6]。在跨平台程序部署的常规实现方式上,通常采用程序开发终端上通过图形化界面生成源代码,然后在整体性能较好的开发终端上动态调用本地相应运行终端的交叉编译工具去对程序文件进行交叉编译[7-8],最终得到该运行终端的可执行测控程序。但该方式会带来以下几个弊端:

1)在测控程序开发过程中,往往存在对于测控程序进行反复修改和调试的工作需求,若采用实时代码编译的模式,每次编译、部署,均会带来较长的时间成本,特别是在软件代码量大、第三方组件多的情况下,会严重影响整个测控系统的调试、运行效率。

2)程序开发终端和程序运行终端之间通常是不同的平台环境,程序开发终端一般为工控机,采用基于x86架构的windows环境,而程序运行终端一般为现场可编程控制器,具体环境因现场终端而异,如MIPS架构下的linux环

收稿日期:2022-12-10; 修回日期:2023-01-04。

作者简介:向友洪(1994-),男,四川达州人,硕士,助理工程师,主要从事反应堆热工水力试验仪控系统设计、电气系统设计、传感器等方向的研究。

引用格式:向友洪,廖芳芳,付 玉,等.基于图形化热工水力测控程序的解释性加载设计[J].计算机测量与控制,2023,31(3):187-192,207.

境。因此，需要运行终端平台在 windows 上有对应的交叉编译器，但目前暂没用可兼容所有目标平台的完善交叉编译工具链。

3）为满足实时编译的设计需求，图形化开发工具会针对不同的目标程序部署平台，集成对应的程序编译环境，会造成编程工具整体组成极为臃肿。

为了解决上述问题，提出了一种基于解释性加载机制的程序部署－运行模式，在基于图形化程序的分布式软件系统应用中，该模式有利于提高程序调试、部署和运行的效率。

1 数据及逻辑关系建立及运行规则设计

本文解释性加载机制的设计实现所依托软件对象为自主开发的热工水力图形化测控程序编程平台，该平台基于 Qt 开发，其主要系统及功能架构如图 1 所示。该平台具备现场设备管理、图形化程序开发－调试－部署－运行、分布式协同开发等功能，可实现热工水力测控程序完整的开发、运维流程。

在图形化编程技术中，程序数据及逻辑关系的建立及运行是一个十分重要的特征，即通过可视化建模技术[9]，来构建图形化程序的运行逻辑。在自主开发的图形化编程平台中，主要是通过标准化的图形化编程控件和关系连线作为具象化元素，以控件拖拽、连线的交互方式，建立程序各编程功能控件之间的数据及逻辑关系。

在程序数据及逻辑关系建立及运行机制中，按关系所属范围的不同，主要分为两个层面：①单程序内部图形化编程元素（含编程控件和连线）之间的数据及逻辑关系；②多程序之间内部图形化编程元素（编程模块）之间的数据及逻辑关系。

1.1 单程序内部的数据及逻辑关系

单一图形化测控程序中，编程控件之间数据及逻辑关系建立及运行机制是程序编制、运行的关键核心。根据对于控件之间等级关系的不同可划分为两类：①控件之间平级关系：在平级控件之间，其相互之间逻辑关系类似于顺序关系，由源节点控件开始，按照图形化“连线 link”，顺序依次运行；②控件之间父子关系：由于“while”、“for”等结构型编程控件的需求，导致属于结构控件内部的编程控件运行顺序发生了变化，结构型控件和其内部所包含控件之间的关系类似于父子关系，运行至结构类控件时，进入该控件内部继续执行。

因此，单程序内部编程控件之间的逻辑关系主要分为两部分：①编程控件之间的连线关系；②编程控件之间的父子关系。

1.1.1 编程控件之间的连线关系

编程控件之间的连线关系为平级执行关系，在同一层级中，其执行关系为顺序型执行，其关系不仅适用于单个程序顶层的控件之间，也可适用于结构型编程控件内部的控件之间。

平级控件的执行逻辑：控件通过连线接收上游控件数

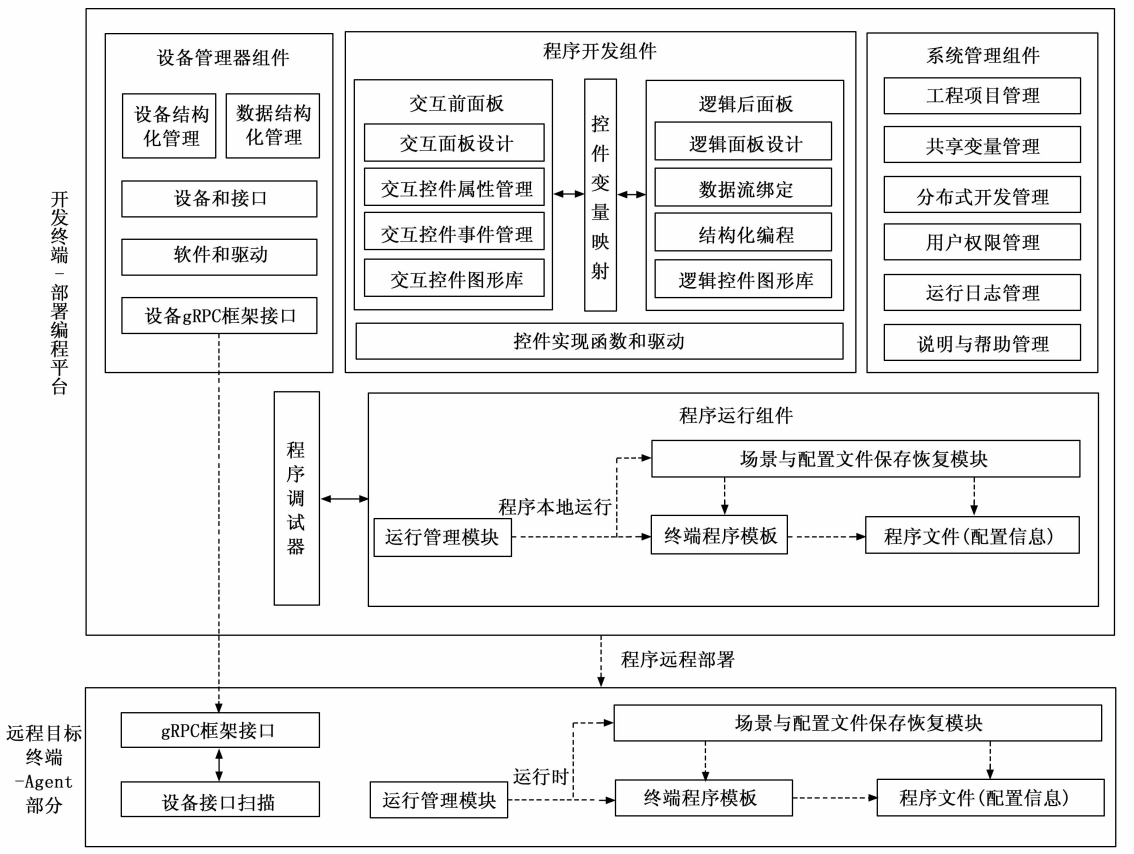

图 1 图形化编程平台系统及功能架构

据，当目标控件的所有输入端口全部收到数据后，会启动执行控件内置的运行逻辑，执行完毕后，控件会将执行的值发送到该控件的各指定输出端口上的连线，再通过该连线转发给下游控件，以此类推，逐级转发下去。

1.1.2　编程控件之间的父子关系

对于结构型控件，由于其本身没有固定的运行逻辑，只是一个容器性质的元素。因此，当结构型控件收到上游发送过来的运行信号之后，其会阻塞该控件同一层级的运行逻辑，然后按照平级关系之间的执行逻辑去启动结构内的控件运行；等到结构内的控件按照容器的规则（比如，条件结构就运行指定分支的逻辑，循环则重复运行指定次数）运行完成，这时容器就会解除阻塞状态，将所需结果从结构内输出并转发到下一级继续运行。

由于上述父子关系运行流程的通用性，针对父子关系中不同结构型控件多层嵌套的特殊情形，其运行机制为逐级阻塞，执行后再逐级解除。假设存在一个“for 循环”嵌套一个“条件结构”的情形，如图 2 所示，当运行逻辑来到 for 循环时，for 循环会先阻塞 L1 层逻辑，启动循环内部逻辑；当运行流程走到条件结构时，条件结构同样阻塞 L2 层逻辑，进入条件结构内部；等到条件结构运行完成，解除 L2 层阻塞，继续往下运行；当 for 循环内部逻辑运行完成之后，会通知 for 循环主体，然后 for 循环会根据循环次数决定是否继续新一轮内部逻辑运行。

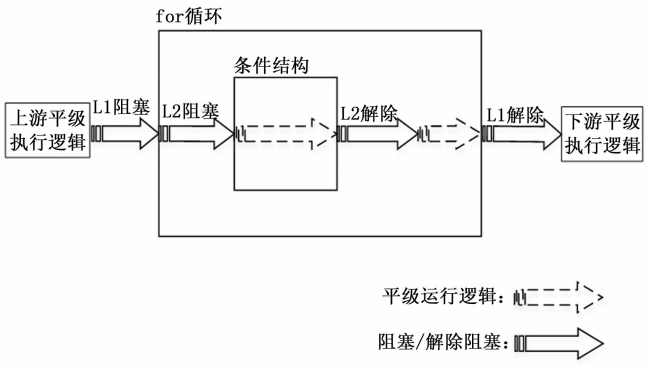

图 2　结构型控件嵌套运行流程示例

编程控件之间的父子关系在测控程序中的图形化实现依托于图形化编程平台。自主开发的图形化编程平台通过构建视图（view）—场景（scene）—元素（Item）三层架构的图形化编程体系模型，利用信号与槽机制实现对象之间的通信[10-11]。其中，视图（view）作为交互层，用来展示场景中的元素，实现编程元素的可视化；场景（scene）作为一个抽象的元素管理容器；元素（Item）作为图形元素的基本单位，是各类图形编程元素功能代码的实现的基础类。

在上述模型中，不同层级之间的消息传递，通过事件穿透机制实现。视图（view）接收用户产生的鼠标点击、拖拽、悬停等操作；然后将事件传递给场景（scene）；最后再将事件传递给元素（Item）进行指定控件功能的处理。

三层架构每一层都有属于自己的坐标系，其中，设计视图（view）和场景（scene）的坐标系均采用左上角作为坐标原点。而针对元素（Item），为了控件内部的绘图方便，设计控件内部的坐标系采用其中心点为坐标原点。在三层架构的事件穿透中，就需要注意不同坐标系下的坐标转换，不然会导致坐标混乱。

因此，在结构型控件和结构内的控件拖动时进行了联动移动设计，一旦建立了控件间的父子关系，在父控件产生移动操作时，其所有子控件均会收到父控件移动的信号，而该信号里面包含偏移量参数值，通过该参数实现同步。

1.2　多程序之间的数据及逻辑关系

1.2.1　MQTT 概述

MQTT（message queuing telemetry transport，消息队列遥测传输协议）是一种基于发布/订阅（publish/ subscribe）模式的“轻量级”通讯协议，其最大的特点是，以极少的代码和有限的带宽，为连接远程设备提供实时可靠的消息服务。MQTT 协议的实现需要客户端和服务器端，而在 MQTT 协议中有 3 种身份：发布者（Publish）、代理（Broker）、订阅者（Subscribe）。其中，消息的发布者和订阅者都是客户端，代理是服务器，消息发布者也可以同时是订阅者。

MQTT 传输的消息分为：主题（Topic）和负载（payload）两部分。Topic 为消息的类型，订阅者（Subscribe）订阅后，就会收到该主题具体的消息内容（payload）[12-13]。

1.2.2　共享变量机制

在多测控程序的分布式应用时，设计共享变量远程传输机制，实现不同测控程序之间的数据交互。该机制从功能设计上，通过变量管理器和共享变量编程控件两部分实现，其中，共享变量管理器作为隐性的全局性变量容器，其中的变量可被网络域中所有程序进行访问，采用 Tag 机制[14]设计；共享变量控件作为显性的图形化程序编程控件，存在“读取”、“写入”两种执行逻辑，实现图形化编程。

共享变量底层实现通过 MQTT 服务完成，MQTT 服务在远端服务器上部署，由远端服务进行管理，MQTT 服务器作为共享变量中转服务器。

在含共享变量控件的图形化程序中，每次该控件运行时，会创建一个临时的 MQTT 客户端，用于发布/接收最新的共享变量值。通过在 MQTT 服务上订阅某个共享变量，通过该变量的 32 位 VariantUuid 在项目管理器中进行查找，当该共享变量的值发生变化，就会收到 MQTT 服务发送的消息信号。

因此，多程序之间的数据交互通过共享变量远程传输机制实现，数据进入具体程序内部后，再由单一程序内部数据及逻辑关系规则管理，即实现从不同程序之间到单一程序内部的完美接入。

2　解释性加载机制原理设计

2.1　解释性加载原理及规则

在图形化程序数据及逻辑关系建立及运行规则的基础上，需要特定的解释性规则来执行图形化程序的运行解释，

其解释性规则主要表现在图形化程序的保存功能和恢复功能上，两者都是解释性规则的体现。由于各编程控件功能代码相对固定，程序编制主要是配置程序包含哪些控件以及控件之间的逻辑关系，因此，本文所研究的解释性加载机制，采用模块化思想，将编程控件固定的功能代码与程序非固定的配置关系相互分离，针对图形化测控程序的发布、运行，提出了一种新的设计思路：以终端程序模板（编译后的终端程序模板）对程序文件（配置信息文件）进行解释性加载。

终端程序模板是一个通用的程序配置解析器，提前针对编程平台所需兼容的目标环境，将各功能控件的内部实现代码编译完成并集成于图形化编程平台中，由于在实际程序部署过程中没有编译步骤，因此，此方法可避免实时编译带来的时间成本较大的问题；同时，同一目标环境下，该环境对应的终端程序模板对部署至该目标环境的所有测控程序具有通用性。

程序文件（配置信息文件）是解释性加载机制中，程序配置信息的输入。图形化程序的执行逻辑为：程序启动时，编程平台的程序运行管理模块首先会遍历查找图形化程序中的所有运行起点，判断运行起点的条件是：①该控件没有输入端口；②该控件没有父节点。遍历成功后，按照平级关系之间的执行逻辑和父子关系之间的执行逻辑，按照具体的程序设计逻辑完成运行。因此，程序文件（配置信息文件）其主要包含三部分内容：控件配置信息、连线配置信息和共享变量配置信息，前两项实现单程序内部的数据及逻辑关系的配置，第三项进行多程序之间数据及逻辑关系的配置。

程序发布过程中，图形化编程平台会将适应目标环境的程序模板和用户编程产生的程序文件一起发布至运行终端。程序本地/远程部署运行主要由以下两个模块支撑：1）程序运行管理模块；2）场景与配置文件保存恢复模块。当用户启动程序或程序自启动时，程序模板通过加载程序文件来恢复用户的编程逻辑，将其加载到场景中，再配合程序运行管理模块，实现程序的运行。解释性加载机制基本原理如图 3 所示。

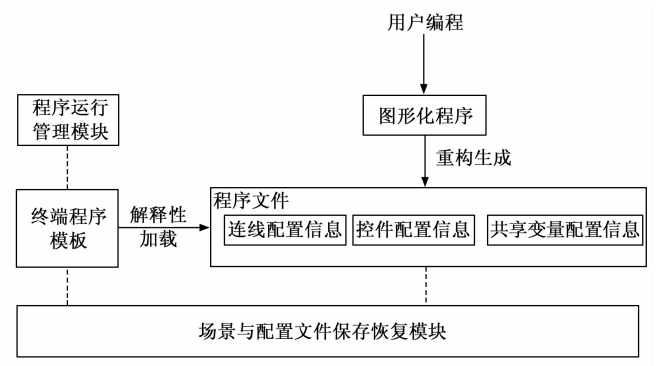

图 3 解释性加载机制基本原理

2.2 程序保存及恢复逻辑

基于解释性加载原理及规则的设计基础，对图形化程序的保存和恢复机制进行了具体设计，确定图形化程序文件的生成保存、加载恢复逻辑，分别如图 4～5 所示。

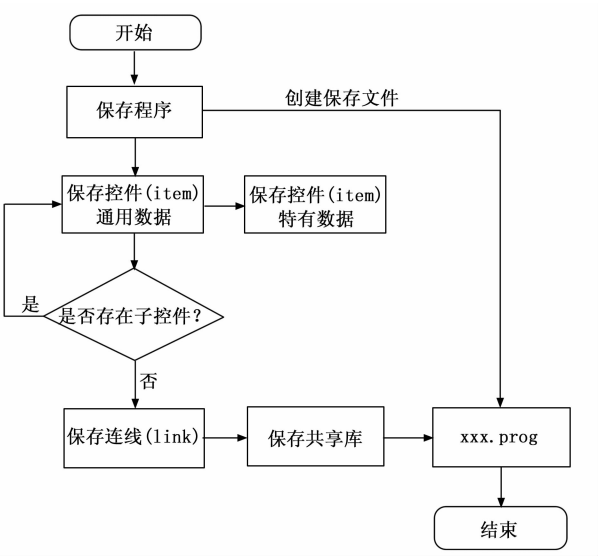

图 4 图形化程序保存逻辑设计

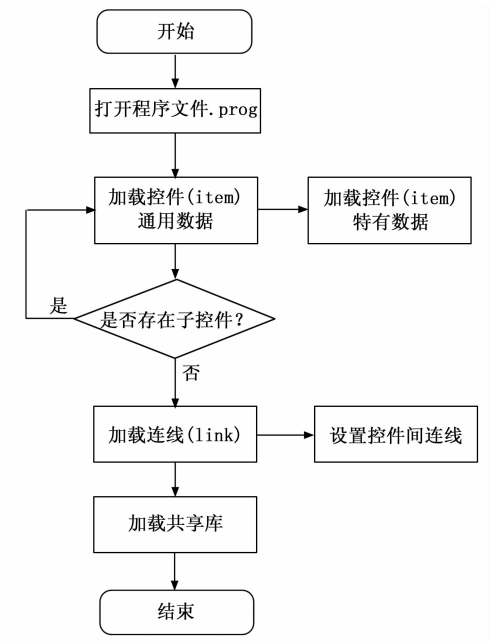

图 5 图形化程序恢复逻辑设计

3 解释性加载机制应用设计

3.1 程序文件设计

程序文件作为配置型文件，包含用户所编写程序的控件配置信息、连线配置信息和共享变量配置信息，因此，根据配置项信息，将程序文件按结构化数据模式进行设计。

在常规结构化数据中，一般采用 XML 和 JSON 作为具体文件形式，相较于 XML，JSON 具有更加简洁性、序列化和反序列化时的速度高和处理占用 CPU 资源更少等特点[15-16]。在满足图形化程序解释性加载程序文件设计的基础上，采用了 JSON 格式作为程序文件的具体格式，保证

了程序文件轻量级、易读性、运行效率等方面均要具有较好的特性。

程序文件的内容设计中，按 JSON 格式对具体的配置项信息进行了设计。其中，“控件配置信息（Items）”包含 BackObjectName、BackPosotionX/Y、Title、InputCount、OutputCount 等通用参数和不同控件的特有参数，在解释性加载时复原用户对于控件的编程信息；“连线配置信息（Lines）”包含 RranchUUID、SourceObject、TargetObject、SourceLinkedPointID、TargetLinkedPointID 等参数，在解释性加载时复原用户对于连线的编程信息，其中通过 SourceObject（源控件身份标识）、TargetObject（目标控件身份标识）、SourceLinkedPointID（源控件连线端口 ID）、TargetLinkedPointID（目标控件连线端口 ID）4 个参数，实现了图形化编程单条连线关系的唯一性和方向性；“共享变量配置信息（Progarm）”包含 SharedVariant、MqttServerIp、LibSharedVariant 等参数，在解释性加载时复原用户对于共享变量的编程信息。

典型程序文件的结构示意如下：

```
{
"Items": [
{
控件 1 配置信息 //此处为 For、While 等
"ChildItems":[ //此处控件 1 嵌套控件 2
{
控件 2 配置信息
}
},
{
控件 n 配置信息
},
],
"Lines": [
{
连线 1 配置信息
},
{
连线 2 配置信息
},
{
连线 n 配置信息
},
],
"program": [
{
"LibSharedVariant"配置信息
},
{
"SharedVariant":[
{
共享变量 1 配置信息
},
{
共享变量 2 配置信息
},
{
共享变量 n 配置信息
},
]
},
{
"MqttServerIp"配置信息
}
]
}
```

3.2　终端程序模板设计

终端程序模板为一种编译后的产物类型，针对不同终端目标环境，其对应有不同的终端程序模板的具象化实参。不同的终端程序模板可通过交叉编译器或者在各自目标环境中通过本地编译获得，集成到图形化编程平台的安装文件中，即可实现多目标平台终端程序模板的同步兼容。在程序部署时，由编程平台中的识别模块对目标环境进行自适应识别，并进行决策，最终执行程序部署操作。

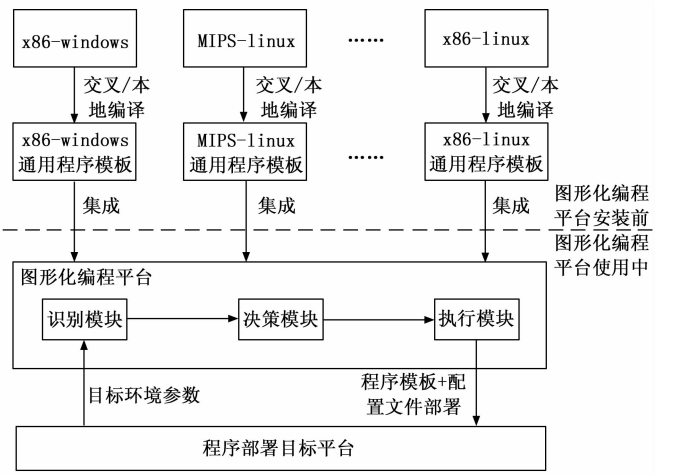

图 6　多程序模板兼容的实现机制

4　解释性加载机制验证

4.1　典型程序运行验证

以验证解释性加载机制的有效性为目的，利用图形化编程平台编制实现“循环加法”功能的应用程序，该应用程序的具体实现逻辑为：通过 For 循环，执行循环 5 次，首次循环时，接受循环外“初始值”和循环内“加数”执行加法操作；后续四次循环，通过 For 循环移位寄存器模式，将上一次循环“加”运算的输出结果和“加数”执行加法操作，最终 For 循环结束后通过“和”控件显示输出。

该程序“和”控件运行显示结果为 5，与预期结果一致，表明解释性加载机制，可实现图形化程序的加载恢复并正确运行。其中，测试程序文件片段如图 7 所示。

```
{
    "Items": [
        {
            "BackHeight": 36,
            "BackObjectName": "NumberConst_{53acae67-1e92-4489-90af-39b3cb1a3176}",
            "BackPositionX": 1746,
            "BackPositionY": 1865,
            "BackVisible": true,
            "BackWidth": 75,
            "ConstValue": "5",
            "ConstValueType": 2,
            "ExactType": 82,
            "InputCount": 0,
            "IsLogicControl": true,
            "ItemType": "NumberConstItemControl",
            "OutputCount": 1,
            "Title": "数值常量"
        },
        {
            "BackHeight": 232,
            "BackObjectName": "ForCircle_{c6b8ad7c-25cd-4424-8fbc-3f87e05b2653}",
            "BackPositionX": 1842,
            "BackPositionY": 1858,
            "BackVisible": true,
            "BackWidth": 180,
            "ChildItems": [
                {
                    "BackHeight": 40,
                    "BackObjectName": "Plus_{8c56ba7a-730e-4a2c-86da-2aabad16c2ba}",
                    "BackPositionX": 1944,
                    "BackPositionY": 1952,
                    "BackVisible": true,
                    "BackWidth": 40,
                    "ExactType": 5,
                    "InputCount": 2,
                    "IsLogicControl": true,
                    "ItemType": "PlusItemControl",
                    "LogicParent": "ForCircle_{c6b8ad7c-25cd-4424-8fbc-3f87e05b2653}",
                    "OutputCount": 1,
                    "Title": "加"
                },
                {
                    "BackHeight": 40,
```


图 7 For 循环加法测试程序文件片段

4.2 热工水力试验台架验证试验

4.2.1 回路运行试验

为了进一步验证解释性加载机制对于热工水力测控系统的分布式适应性，利用图形化编程平台中的各类型图形编程控件，对热工水力试验台架测控系统进行测控程序开发，具体涉及热工回路监控、测点曲线显示、预热器控制、本体加热等试验业务功能。

以自主研发的多总线兼容可编程终端控制器和现场总线控制系统[17-18]为依托，该控制器 CPU 采用龙芯 3A3000 高性能 4 核通用处理器[19]，基于 linux3.0 内核的国产支持 Loongnix 操作系统，通过背板高速通信总线和总线插槽板卡实现多总线信号的板卡兼容[20]。本热工水力试验测控系统中，包含 CAN、Profibus PA、MODBUS RTU 三种常用现场通信总线以及 AIO、DIO 信号。

本次测试中，测控程序部署目标环境包含上位机（x86-windows）、下位机（MIPS-linux）两种环境，进行了热工水力试验台架的本体升温、测点监控等功能的回路运行试验，其中，采集、存储、显示、数据处理等程序均正常运行。

4.2.2 测量精度试验

通过标准信号源设备给现场仪表输入标准信号，记录 CAN 通信、Profibus PA 通信、MODBUS RTU 通信、AIO 的现场信号值和上位机程序的显示信号值，并计算出每一组试验数据所对应的误差，试验数据如表 1～5 所示。

表 1 CAN 试验数据及误差

序号	输入值/mV	采集值/mV	引用误差/%
1	20.00	19.99	−0.005
2	40.00	40.00	0.000
3	60.00	59.99	−0.005
4	80.00	80.00	0.000
5	100.00	99.99	−0.005

表 2 Profibus PA 试验数据及误差

序号	输入值/kPa	采集值/kPa	引用误差/%
1	10	9.985	−0.015
2	20	19.962	−0.038
3	30	29.986	−0.014
4	40	39.978	−0.022
5	50	50.058	0.058

表 3 MODBUS RTU 试验数据及误差

序号	输入值/℃	采集值/℃	绝对误差/℃
1	0	0.540	0.540
2	10	10.542	0.542
3	20	20.530	0.530
4	30	30.534	0.534
5	40	40.529	0.529

表 4 AI 试验数据及误差

序号	输入值/V	采集值/V	引用误差/%
1	1.000	0.998	−0.040
2	2.000	1.997	−0.060
3	3.000	2.997	−0.060
4	4.000	3.996	−0.080
5	5.000	4.996	−0.080

表 5 AO 试验数据及误差

序号	设定值/mA	输出值/mA	引用误差/%
1	6.000	5.997	−0.019
2	8.000	7.997	−0.019
3	10.000	9.998	−0.013
4	12.000	11.998	−0.013
5	14.000	13.999	−0.006

通过现场短接 DI 通道，记录并查看 DI 的现场输入状态和上位机程序的显示状态是否一致；通过上位机设定 DO 输出状态，测量并查看 DO 的现场输出状态和上位机程序的设定状态是否一致，具体试验结果如表 6 所示。

表 6 DIO 状态试验数据

序号	DI 输入状态	DI 采集状态	DO 设定状态	DO 输出状态
1	1	1	1	1
2	0	0	0	0
3	1	1	1	1
4	0	0	0	0
5	1	1	1	1

试验结果进一步证明，该解释性加载机制具有多目标运行终端环境兼容性的特点，同时可实现热工水力试验测控系统图形化程序的动态加载与运行，满足热工水力试验测控系统的开发要求。

（下转第 207 页）