## 背景

工艺信息系统一般包含产品、零部件、工艺、工序、工步等核心对象类，并由它们关联出如材料、设备、工装等对象类，各对象类有自己的属性与方法，由此形成面向对象的工艺信息建模体系。而随着工业的发展，各种新型工艺内容不断的增加，我们需要形成一套通用的解决方案，即依据用户给定的输入产品信息、计算规则等，设计一个满足工艺要求的工艺方案，输出加工过程的要求，最大化资源利用率。

随着新工艺的出现和现有工艺不断的完善，目前最好的方式是采用xml文件来保存工艺内容，这种存储方式可以更好的解释、更新、扩展工艺和根据规则计算出工艺要求。

所有的工艺内容都由三部分构成：产品的输入参数、产品工艺计算规则和加工过程的输出。基于输入的定义我们可以以一种指定格式来存取，而因为工艺规则的多样性，我们不可能以一种固定的方式来定义规则和对象的输出，需要定义一个规则引擎来解释用户定义的不同的规则。

## 方案

基于以上所述背景，我们可以实现一个自定义的规则引擎：即用户依据规则引擎的格式来定义规则，规则引擎再把用户定义的规则解析出来，参与工艺的计算。如果使用这种方式，在新的规则格式出现或计算方式更改之后，可能会面临频繁的改动，对扩展和用户熟悉难度来说，会相对困难。

虽然目前主流编程语言在系统开发和大型程序设计方面的地位仍不可替代，但由于脚本语言的小巧、容易上手、不用编译和安全性等特点，越来越多的大型程序都选择内嵌脚本语言来进行功能扩充和二次开发。基于通用性和扩展的要求，我们考虑的方向是偏向于使用第三方的脚本语言：即把用户定义的所有内容转换成脚本，用第三方的脚本来解释。

### 语言

脚本语言的选择有：JavaScript、VBScript、Python和Clips，为此，我们根据各种脚本语言在工艺软件系统中的适用性和优劣程度做了一个对比，如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 跨平台性 | 扩展能力 | 优点 | 缺点 |
| JavaScript | 是，适用于任何平台，但需要解释器来运行 | 支持COM组件和外部函数的调用 | 1.C风格的语法简单易用，基于对象  2.支持COM组件 | 1.主要内嵌在html中，提供的标准函数库非常小，很多功能不具备  2.与C/C++的交互操作繁琐 |
| VBScript | 不是，只能在Windows系列的系统环境中运行 | 支持COM组件的扩展 | 1.语法简单易用，基于对象  2.支持COM组件调用和扩展 | 1.不支持跨平台运行  2.标准函数库小 |
| Python | 是，适用于任何平台，和JavaScript一样都需要解释器来运行 | 不支持COM组件，支持C/C++函数和类的扩展 | 1.语法简单易用，基于对象  2.内置函数标准库强大，支持绝大多数的数学运算  3.与C/C++交互操作简单，易于调试  4.支持C/C++函数的扩展  5.适合做数据的处理 | 1.不支持COM组件调用  2.没有明显的语句控制结构，依靠缩进来控制语句块  3.需安装解释器 |
| Clips | 是，适用于任何平台，和JavaScript一样都需要解释器来运行 | 不支持COM组件，支持C的扩展 | 1.函数式语言，函数也是元数据  2.细节先掩盖在函数中，从高层次抽象算法/流程，再逐步细化，对复杂事物的处理显得相对容易  3.适合做抽象计算，以推理的方式来解决问题 | 1.因为计算结果的不确定性，不适合处理固定结果的流程  2赋值语句就是公式定义，不能对变量直接赋值，不便理解  3.没有顺序和循环，只有递归和其他的东西，不适合处理简单情况的事物流程  4.无法结构化的返回处理结果 |

从以上表格中可以看出， JavaScript和Python都可以作为工艺存储的脚本引擎，Clips相对来说，更适用抽象层次很高的程序推理，在给定规则以后，用户可以给出一些事实，然后这个系统就可以像一个专家一样，判定你这个事实并给出相应的结论，但是因为程序相对来说语言不易理解、没有明显的顺序和循环控制以及结果的不确定性，并不适用于工艺的处理；python没有类似JavaScript的 ‘{}’、VBScript的 ‘if elif endif’来显式指定语句块的起始结构，它需要依靠按制表符缩进来控制语句结果，所以在定义完所有的变量、规则等之后，往脚本里写时，需要完成额外的工作来控制语句的构成，以确保符合脚本语言规范。

综合考虑语言的优缺点以及它在这个软件的适用性来看，因为python的面向对象性、可扩展性以及函数库的强大能力等，我们更倾向于使用python作为规则解析器来进行功能扩充。

### 输入输出

输入输出的定义可以分为以下三部分：

1. 输入参数的定义

包括参数名称、说明、参数类型、参数范围和参数值等；

1. 计算参数和规则的定义

包括参数名称或对象属性名称、说明和计算规则语句等；

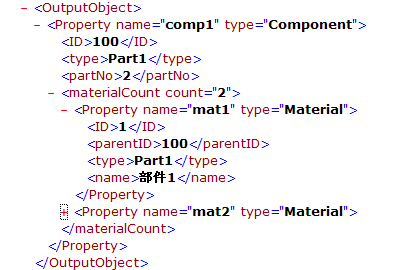
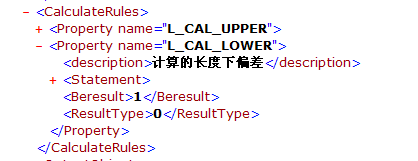
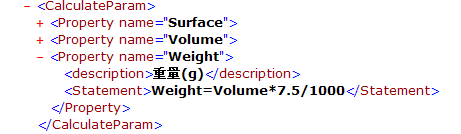
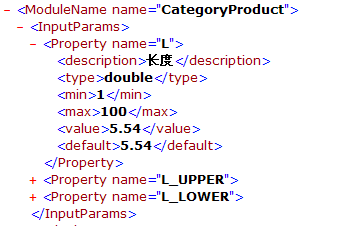
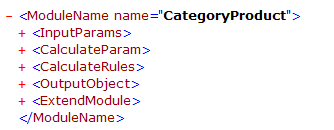
1. 输出对象和对象关系之间的定义

包括类名、对象名、对象属性值、与其他对象的关系和子对象类型与个数等；

如:一个零件可能包含若干种材料，而材料可能又包含若干属性和工艺。

各部分的格式如图表1所示：

输入参数对应InputParams，计算参数对应CalculateParams，计算规则对应CalculateRules，输出对象对应OutputObject，扩展模块对应ExtendModule。



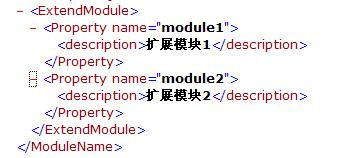
图表 1

### 模块扩展

在实际使用过程中，我们经常会需要调用某些已经定义好的函数和类来扩展工艺处理能力，这时就需要添加脚本对已有模块的扩展能力。

1. 扩展模块的定义

定义格式如下图表2所示，包括模块名称和说明。在定义好要导入的扩展模块后，用户就可以使用扩展模块里的函数或类。用C/C++语言扩展python很简单，只须按照python指定的方式生成动态库，即把需要引用定义好的外部函数，按照这种方式改写，然后就可以在import这个模块后，调用模块里的C/C++函数或类。



图表 2

1. 模块扩展示例

添加计算两个整数的和,把文本保存为add.c

#include <Python.h>

static PyObject \* ex\_add(PyObject \*self, PyObject \*args)

{

int a = 0;

int b = 0;

if (!PyArg\_ParseTuple(args, "ii", &a, &b)) {  
return NULL;

}

return Py\_BuildValue("i", a + b);;

}

static PyMethodDef add\_methods[] = {

   {"add", ex\_add, METH\_VARARGS, "integer add doc string"},

   {NULL, NULL}

};

PyMODINIT\_FUNC initadd(void)

{

   Py\_InitModule("add", add\_methods);

}

编译这个文件生成名为add.dll的动态库, 修改后缀名为add.pyd， 在脚本中可以用如下方式调用该方法：

import add  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 firstParam = 100   
    secondParam = 200   
    result = add.ex\_add(firstParam, secondParam)

if result:

print (result)

### 用户使用

1. 参数的定义

在添加参数时，指定参数的名称、说明、类型、取值范围和初始值。

1. 输出对象的定义

选择输出对象的类型，指定对象之间的关系，然后给各个对象的属性设置相应的值。

1. 规则的定义

规则的定义，采用文本编辑框来供用户设定，在设定规则时，可以检查对象属性、输入参数属性是否引用正确、规则设定是否合理，但是在定义阶段尚无法判断规则是否能够执行出正确结果。

1. 扩展模块的定义

添加需要引用的扩展模块，以便调用模块内部的函数。

### 部署说明

1. 需安装解释器

在需要使用这个功能时，我们会把生成的库和对应平台的解释器安装文件都打包在一起，会在安装库的同时把解释器一起安装，以避免无法使用的问题。

## 结语

这种采用xml格式存储工艺内容与使用第三方脚本语言来作为规则解析的方式，一方面为用户提供了完全可视化的解决方案，系统的可扩展性好，支持多种接口的扩展，另一方面因为输出类型和输出对象以及对象与对象关系的多样性，可以更清晰、一目了然的把工艺内容展现给用户，用户就可以得到一个按结构组织的工艺加工流程。