平台组件模块详细设计

# 概述

如平台组件模块需求分析与概要设计中所述，该模块的设计可划分为平台基础层、平台外的接口与组件、平台与平台外组件之间的交互协议以及平台外组件之间的交互协议。下文将对每一部分逐一介绍。

# 定义

在具体介绍设计之前，需要先明确一些设计中使用的具体名词。

## 接口ID

每一个接口的唯一标识，也是平台用来区分每一个接口的唯一方式。IComponent接口作为所有接口的共有接口，其ID为0，其它接口依次递增。在组件的编码中，组件会定义自己的外部接口，每个接口亦必须声明其接口ID，这通过C++语言中的enum方式实现。在定义外部接口时，必须依赖平台提供的宏BEGIN\_CLIENT\_CIID和END\_CLIENT\_CIID进行定义，以保证外部接口的ID紧接内部接口的ID。

## 接口名称

每一个接口名称的字符表示。

## 组件类型ID

我们使用组件类型ID这一概念以区分组件实例ID这一概念。每一个组件都有其类型，具体表现为一个C++的类，他通过继承来达到对某一接口的具体实现。组件类型ID即为每一个组件类的唯一标识，也是平台用来区分每一个组件类的唯一方式。在定义一个组件类时，必须声明其组件类型ID，这通过C++语言中的enum方式实现。在定义外部组件类时，必须依赖平台提供的宏BEGIN\_CLIENT\_CCID和END\_CLIENT\_CCID进行定义，以保证外部组件类的ID紧接内部组件类的ID。

## 组件类型名称

每一个组件类型名称的字符表示。

## 属性ID

属性ID是组件每一个属性的唯一标识，也是组件内用来区分每一个属性的唯一方式。因平台对于组件的操作并不区分其属性类型，故而必须通过属性ID的方式进行标识。由于属性被限定在组件空间内，故而其ID的定义只在组件内保持唯一性。每一个组件都有其ID从0开始的一组属性。

## 属性类型

由于平台对于组件的操作并不区分其属性类型，故而需要用另外一个域来定义这个类型。属性类型包括未知类型（非法类型）、整型、双精度浮点型以及字符串型。在组件内对属性进行操作时（获得与设置操作），会根据传入的属性类型对值进行对应的转换，获得所需类型。

# 平台基础层

平台基础层提供了平台对外的全部操作。外部的程序，包括客户端程序、服务器程序以及模型搭建程序，都通过平台基础层对平台功能进行调用。平台基础层提供的功能主要包括平台初始化、平台关闭、获得外部接口列表、获得组件类型列表、获得组件实例列表、获得内部算法列表等功能。

## 平台初始化

平台对象在创建之后，需要进行初始化调用。他主要完成如下功能：

### 读取平台配置

通过读取平台配置文件，获取平台配置信息。平台配置文件名不可变，随平台一同交付，文件名写死在平台代码中。配置信息内容可变，平台根据配置信息，获得组件配置文件名和算法配置文件名。

### 读取组件配置文件

在得到组件配置文件名之后，读取该组件配置文件。该配置文件中的内容为全部外部组件DLL文件名列表。

### 注册外部组件

根据组件配置文件内容，逐个读入外部组件DLL文件，调用DLL文件中导出的规定好名字的组件注册函数，完成对外部组件的注册。该组件注册函数将在后文中详述。

### 初始化算法运行环境

由于平台运行的算法格式为MatLab编译出的DLL文件，运行时需要初始化MatLab运行时环境，该步骤即完成这项工作。

### 读取算法配置文件

根据算法配置文件内容，逐个读入算法配置信息。算法配置信息包含算法ID、算法名称、算法DLL文件名、算法入口函数等信息。

### 注册内部算法

根据读入的算法配置信息，根据算法ID、算法名称、算法DLL文件名及算法入口函数名，逐个初始化算法实例。

## 平台关闭

在程序运行结束，需要对平台对象进行销毁以释放资源。它主要完成如下功能：

### 释放全部外部组件

释放所读入的全部组件DLL。

### 关闭算法运行环境

关闭MatLab运行时环境。

## 获得外部接口列表

对于平台外部的程序，若想获得接口信息，必须通过平台来进行。该功能即为向外部提供全部的外部接口列表。外部程序可根据该列表获得外部接口信息，以显示给用户。该功能返回平台内维护的外部接口列表。

## 获得组件类型列表

对于平台外部的程序，若想获得组件类型列表，必须通过平台来进行。该功能即为向外部提供全部的外部组件类型列表。外部程序可根据该列表获得外部组件类型的信息，以显示给用户。该功能返回平台内维护的外部组件类型列表。

## 获得组件实例列表

用户通过模型搭建程序，创建出若干组件实例，这些组件实例均被平台所保存并维护。外部程序若想获得组件实例信息，必须通过平台来进行。该功能即为向外部提供全部的组件实例列表。外部程序可根据该列表获得组件实例信息，以显示给用户。该功能返回平台内维护的组件实例列表。

## 获得内部算法列表

对于平台外部的程序，若想获得内部算法列表，必须通过平台来进行。该功能即为向外部提供全部的内部算法列表。外部程序可根据该列表获得内部算法类型的信息，以显示给用户。该功能返回平台内维护的内部算法列表。

# 平台与平台外组件之间的交互协议

平台设计的一个重要出发点即是实现平台与组件之间的分离。采用C++语言提供的面向对象特性，可以实现通过接口的编程，最终实现平台与组件的分离。平台与组件的分离可以在绝大多数情况下达到免去重新编译的程度。这就要求平台与平台外组件之间定义一套相互间的交互协议。这个交互协议首先定义了一组平台能够识别的基础接口，所有组件均要实现这样一组接口，以供平台组织、管理。之后平台还需要对平台外组件的编程定义一套规范，规定平台外组件必须导出的函数以供平台获得必要的组件信息。

## 基础接口

基础接口通过一个接口类（IComponent）来定义，它包含了平台对组件要进行的所有操作，这些操作均被定义为纯虚函数，不提供具体实现。所有组件必须符合该组接口的标准，具体即为必须继承这个接口类，并实现其全部虚函数。基础接口所定义的操作如下所述。

### 获得组件类型ID

平台需要知道一个组件实例所述的组件类型，该信息通过此接口获得，返回组件类型ID。

### 保存至二进制文件

组件需要将自己保存至一个二进制文件，通过此接口完成。

### 从二进制文件中读取

平台需要从二进制文件中获取组件实例，通过此接口完成。

### 销毁组件

平台需要能够将一个组件实例销毁，通过此接口完成。

### 得到该组件的某一接口

每个组件必须实现这一函数，以将自己实现的某一接口返回，以供外部操作。该功能通过传入一个欲查询的接口ID，内部进行检测，若该组件实现了这个接口，便可将该接口返回，否则返回一个空指针。

### 获得组件实例ID

每一个组件实例都有自己唯一的组件实例ID，通过此接口实现。

### 设置组件实例ID

通过此接口将一个组件实例的ID设置为某值。

### 获得组件实例名称

每一个组件实例都有自己的名称，该名称可有用户在搭建模型时指定，平台亦需要获得此名称，通过此接口完成。

### 设置组件实例名称

当用户对一个组件实例进行操作时，可以设置组件实例名称，通过此接口完成。

### 销毁组件

平台需要能够将一个组件实例销毁，通过此接口完成。

### 获得组件的属性列表

平台需要知道一个组件实例都有哪些属性，通过此接口完成。

### 获得组件的某一个属性值

平台需要知道一个组件实例的某一个属性的值，通过该接口，传入具体的属性ID，获得该值。

### 设置组件的某一个属性值

当用户对一个组件实例进行操作时，可以设置组件实例的某一个属性的值，通过该接口，传入具体的属性ID及其值进行设置。

### 将组件与另一组间相关联

该接口主要用于进行组件之间的交互，即将某一组件实例与另一组件实例相关联。该接口将欲关联的组件作为参数传入，组件内部进行类型检测判断关联是否合法，之后返回关联是否成功。

## 外部组件编程规范

平台为了实现获知外部组件的信息，需要外部组件编写符合规范的代码以供平台调用，平台规定了该编程规范。他具体通过要求外部组件定义两个导出函数来实现，分别为注册接口类型以及注册组件类型。平台会在初始化时，自动调用这两个函数，完成对接口类型和外部组件类型的注册。

### 注册接口类型

该函数原型为：

extern "C" \_\_declspec(dllexport) void RegisterInterfaceType(InterfaceTypeMap &interfaceTypeMap);

外部组件应当实现该函数以供平台获得定义的外部接口信息。该函数传入一个接口类型列表的引用，函数内部需要将外部接口逐个设置到该列表中。

### 注册外部组件类型

该函数原型为：

extern "C" \_\_declspec(dllexport) void RegisterComponentType(ComponentTypeMap &componentTypeMap);

外部组件应当实现该函数以供平台获得定义的外部组件类型信息。该函数传入一个组件类型列表的引用，函数内部需要将外部组件类型逐个设置到该列表中。

# 平台外组件组件之间的交互协议

由于模型搭建的需要，平台外组件之间需要进行交互，即一个组件需要能够调用另外一个组件的某些接口（可能是内部接口，也有可能是外部接口），而平台对于所有的组件并不区别对待，所以需要提供一套平台外组件之间的交互协议。这套交互协议主要包括平台外的接口式编程，以及具体的交互手段。

## 平台外组件的接口式编程

在设计一个平台外组件之前，需要按照具体项目要求，抽象出该项目中的各类事物，并为其定义完善的接口。当然，每个外部接口都需要继承自基础接口，以供平台识别和管理。每一个外部组件，只实现某一个具体的外部接口。而在组件进行交互时，只看得到目标组件的接口类型，进而对该接口进行操作，调用接口方法实现交互。

## 平台外组件之间的交互手段

平台外组件之间的具体交互手段需要遵循以下方式。首先组件对于要进行交互的另一组件，要对其进行类型检测。在确定类型合法后，再将该组件转化成对应类型的接口，之后通过该接口的操作，完成具体的交互行为。

### 类型检测

类型检测主要通过基础接口中的得到该组件的某一接口操作来完成。对于一个外部组件，他对他所能交互的接口类型必须有着清晰的认识。他在知道欲交互的接口后，通过将该接口ID传入基础接口中的“得到该组件的某一接口”操作，获得返回的接口。若目标组件没有实现这一接口，返回的接口将为空。组件则以此判断目标组件是否为合法类型。

### 调用接口操作

在组件获得目标组件的合法接口后，需要将该接口强制转换成欲操作的接口的实际类型（获得接口操作返回的指针类型为void\*）。强制转换后，该接口即具备了具体的操作能力。之后该组件通过对这一接口的操作的调用，实现具体的交互功能。