通信接口及协议配置组件技术方案

# 概述

通信接口及协议配置组件主要用于以项目为单位，对项目中的受控对象的TCP、UDP、串口、CAN和DI/DO等的接口及通信协议进行配置、配置文件存储、接口和协议开发的C#、C++代码自动生成。为上层应用提供一致的通信接口及协议配置和开发代码框架支持。

# 技术要求

## 功能要求

1. 提供独立的底层通信接口模块，包括TCP（服务器和客户端）、UDP、串口、CAN和DI/DO。其中CAN需提供对广州致远和研华两个厂家设备的支持，DI/DO需提供对研华厂家设备的支持；
2. 提供协议配置模块，可对TCP、UDP、串口、CAN和DI/DO这些通信方式进行协议配置。支持配置任意格式的通信协议。

数据内容部分，支持配置常用的数据格式，包括bool、byte、sbyte、ushort、short、uint、int、ulong、long、float、double、枚举类型以及数组类型。支持按位或任意位的组合（不大于8位）进行数据定义与解析。支持通过配置公式的方式进行数据自动计算。支持数据校验算法及其配置选取（单、双字节和校验、CRC校验以及其它自定义校验等）。

1. 协议配置数据以文件方式存储，格式需要采用Xml或JSON或其他自定义文本格式。文件以项目为单位，一个项目一组独立的文件，一个项目中包含多个受控对象，与受控对象之间的通信方式（TCP、UDP、串口等）可配置，每种通信方式的通信参数可配置，比如串口和CAN的波特率等；
2. 提供独立的协议配置软件，支持以面向对象的方式进行协议配置（即不是基于数据包本身来进行协议配置，而是基于受控对象进行协议配置，包括受控对象的属性、收发数据包、通信方式以及其属性等），从而实现对协议配置文件的新建、编辑和删除操作。协议配置软件支持可视化的配置方式，同时支持对协议配置文件以文本方式直接编辑（提供语法检查等功能）；
3. 提供针对协议配置的代码自动生成软件（可与协议配置软件合并）。选择某个项目协议配置文件后，可选择其中的一个或多个受控对象自动生成代码，生成的代码直接体现为受控对象本身（包括受控对象的属性、收发数据包、通信方式及其属性等）。生成代码的模板运行编辑。

生成C#代码时，根据项目ID和受控对象ID自动生成命名空间，每个受控对象的类文件以文件夹形式分组排列。所有的值类型变量，需要将其封装为类使用，类名为Parameter<T>，数值使用T Value属性访问。值类型变量，泛型传入的类型均使用Nullable类型。Parameter<T>提供一些基本的属性，比如string Name，填入中文描述。还需提供ValueChanged事件。

1. 应针对组件处理异常设计异常处理机制，并向上层提供异常代码传输接口。
2. 开放底层硬件驱动调用接口，提供接入新的接口通信方式和硬件驱动功能。

## 开发与运行环境要求

该软件组件在Windows操作系统下使用（Windows7及以上版本）。

需支持C#语言（运行库为.NET Framework4.5及以上版本），以及C++语言（包括Visual C++和Qt）。其中对C#语言的开发采用Visual Studio 2017。

## 质量要求

该软件组件开发应严格按照软件工程化要求进行，完成需求分析、软件设计、软件配置项测试等相关工作，形成软件需求规格说明、软件设计说明、软件测试说明、软件测试报告和软件用户手册，所有文件应经正式评审/会签。

该软件组件需进行第三方测评，并提供正式的第三方测评报告。

## 交付要求

应提供如下交付物：

1. 通信接口及协议配置组件源代码及可执行程序；
2. 用户手册；
3. 需求规格说明；
4. 软件设计说明；
5. 软件测试说明；
6. 软件测试报告；
7. 软件第三方测评报告。

注：文档均提供电子版和纸质两种交付物。

# 软件设计说明

## 软件架构

通信接口及协议配置组件软件采用分层的软件开发框架，软件分层架构如图 3‑1所示，共分为3层，分别为驱动层、传输层和应用层，通过这3层结构，实现了软件的构件化与模块化，确保能够适应快速变化的测试需求。



图 3‑1软件架构图

驱动层为各个接口板卡提供的驱动，运行于Windows操作系统之上，包括以太网卡驱动、串口板卡驱动、CAN板卡驱动和DI/DO板卡驱动，其中以太网口和串口板卡驱动为开发主机所带板卡的驱动，CAN板卡为广州致远和研华两个厂家板卡设备的驱动，DI/DO为研华厂家设备板卡的驱动。

传输层用于在驱动层的基础上，封装成程序编程接口，以屏蔽不同的硬件设备通信编程接口的不同，通信服务层包括TCP服务器通信、TCP客户端通信、UDP通信、串口通信、CAN通信和DI/DO通信等，通过传输层可以确保应用层进行调用通信时能够统一实现。

应用层提供满足用户需求的应用功能，由项目管理、协议配置、协议数据存储、C#代码生成和C++代码生成等5个模块组成。

## 软件使用流程

软件使用流程如图 3‑2所示。

首先建立项目，形成受控对象及其接口与协议管理的统一框架，接下来对受控对象进行建模描述，把项目中包含的受控对象添加到项目中，之后逐一对受控对象的接口进行配置，配置接口的参数，再对接口上的通信协议进行配置，按照通信协议格式要求，建立通信协议的结构和行为，在进行上述操作的过程中均可保存项目及接口和协议配置的数据。协议配置完成后，进行语法检查后可按需生成C#代码和C++代码。



图 3‑2软件使用流程图

## 应用层设计

### 项目管理

测试项目是用户组织通信接口及协议配置的基本单位，将受控对象、受控对象的接口及接口协议等内容组织成有机整体，统一管理接口及协议配置的资源数据结构。每个测试项目默认包括了受控对象、接口及协议等主要部分。用户创建测试项目成功后，自动生成一个空白的项目框架，用户可以在该框架下增加受控对象、在受控对象下可增加接口、在接口下可增加协议。项目管理还包括打开项目、关闭项目和项目属性设置等功能

### 协议配置

提供可视化的接口和协议配置功能。

首先对受控对象进行建模，按需配置受控对象的属性；

之后进行受控对象接口配置，提供TCP服务器、TCP客户端、UDP、串口、CAN和DI/DO等接口类型供用户选择，提供适合各类接口的参数供用户可视化配置；

再后对接口的通信协议进行配置，提供可视化的协议配置方式，支持配置常用的数据格式，包括bool、byte、sbyte、ushort、short、uint、int、ulong、long、float、double、枚举类型以及数组类型。支持按位或任意位的组合（不大于8位）进行数据定义与解析。支持通过配置公式的方式进行数据自动计算。支持数据校验算法及其配置选取（单、双字节和校验、CRC校验以及其它自定义校验等）。

提供文本和图形化2种配置方式。

### 协议配置数据存储格式存储

采用XML文件格式存储协议配置数据，一个项目一个XML文件，XML文件根节点为项目，项目的下一级节点为受控对象，受控对象下一级为接口，接口的下一级为协议。XML文件格式如图 3‑3所示。

|  |
| --- |
| <Project name=”项目名称”，……>  < ControlledObject name=”受控对象名称1”，……>  <Interface name=”接口名称”，type=”接口类型”，parameter1=”参数值”,……>  <Protocol name=”协议名称”，其他属性>  协议字段  </Protocol>  ……  <Protocol>  ……  </Protocol>  </ Interface >  ……  <Interface name=”接口名称”，type=”接口类型”，parameter1=”参数值”,……>  ……  </ Interface >  </ ControlledObject >  ……  < ControlledObject name=”受控对象名称n”，……>  ……  </ ControlledObject >  </Project> |

图 3‑3 XML文件格式示例

### C#代码生成

按照C#语法要求将保存的协议配置生成C#代码，并把需要的库文件一起生成为一个可供开发使用的框架。

生成代码的范围可以按受控对象、接口及协议等层级选择。

生成C#代码时，根据项目ID和受控对象ID自动生成命名空间，每个受控对象的类文件以文件夹形式分组排列。所有的值类型变量，需要将其封装为类使用，类名为Parameter<T>，数值使用T Value属性访问。值类型变量，泛型传入的类型均使用Nullable类型。Parameter<T>提供一些基本的属性，比如string Name，填入中文描述。提供ValueChanged事件。

提供的库及API具有异常处理功能，可向上层提供异常代码传输接口。

### C++代码生成

按照C++语法要求将保存的协议配置生成C++代码，并把需要的库文件一起生成为一个可供开发使用的框架。

生成代码的范围可以按受控对象、接口及协议等层级选择。

根据项目ID和受控对象ID自动生成命名空间，每个受控对象的类文件以文件夹形式分组排列。所有的值类型变量，需要将其封装为类使用，类名为Parameter<T>，数值使用T Value属性访问。值类型变量，泛型传入的类型均使用Nullable类型。Parameter<T>提供一些基本的属性，比如string Name，填入中文描述。提供ValueChanged事件。

提供的库及API具有异常处理功能，可向上层提供异常代码传输接口。

## 传输层设计

传输层主要按需要对TCP服务器、TCP客户端、UDP、串口、CAN和DI/DO等板卡驱动进行封装，提供一致的对应用层使用的API，封装分为设计时封装和运行时封装2部分。

设计时封装。主要完成设备板卡参数的配置，包括板卡级设备封装和通道级设备封装；封装方式：动态链接库；加载动态链接库后在测试设计时新增板卡可供用户使用。板卡级设备封装按照设备集成标准，将新增的板卡的具体参数按照平台的接口标准，以具体参数属性化的方式封装到平台上。通道级接口封装主要是按照通道集成标准，将每个板卡的通道参数具体化、可配置化，包括通道数的可视化表格配置及通道运行时参数的可视化配置。以属性表格的界面形式展现给用户进行应用配置。

运行时封装。将每一种板卡驱动以两层结构封装，上层是板卡，下层是通道，生成C++文件（.h和.cpp）供上层应用程序使用。

板卡类的成员函数主要包括4个，分别是获取板卡标识函数className()、获取当前板卡对象函数get(int id)、获取板卡的通道对象函数getChannel(const Setup\_T& setup)和检查板卡是否有效函数isValid()。

通道类的成员函数包括主要包括7个，分别是初始化设置通道函数setup(const Setup\_T& s)、检查通道是否有效函数isValid()、获取通道标识函数path()，同步读函数read()、同步写函数write()、异步中断读函数intRead()、异步写函数asyncWrite()。

# 开发运行环境

## 开发环境

操作系统：Windows10；

开发工具：Visual Studio2017；

开发语言：C#、C++。

## 运行环境

操作系统：Windows操作系统（Windows7及以上版本）；

运行库：.NET Framework4.5及以上版本；

开发环境：（1）C#语言，Visual Studio 2017；

（2）C++语言，Visual C++和Qt。

# 质量保证

本项目严格按照系统工程化标准进行开发，遵循GJB2786A-2009军用软件开发通过要求开发规程规范，产品开发经过以下几个阶段：

（1）软件需求分析与定义阶段；

（2）软件架构设计阶段；

（3）软件模块详细设计阶段；

（4）软件编码阶段；

（5）软件单元测试阶段；

（6）软件配置项测试阶段。

形成的软件产品包括软件源代码、软件安装包、 软件需求规格说明、软件设计说明、软件测试说明、软件测试报告和件用户手册，所有文件应经正式评审/会签。

该软件组件需进行第三方测评，并提供正式的第三方测评报告。

# 交付清单

交付清单详见表 6‑1。

表 6‑1交付清单

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 名称 | 数量 | 备注 |
|  | 通信接口及协议配置组件源代码 | 1 |  |
|  | 通信接口及协议配置组件可执行程序 | 1 |  |
|  | 用户手册 | 2 | 纸质和电子版各1份 |
|  | 需求规格说明 | 2 | 纸质和电子版各1份 |
|  | 软件设计说明 | 2 | 纸质和电子版各1份 |
|  | 软件测试说明 | 2 | 纸质和电子版各1份 |
|  | 软件测试报告 | 2 | 纸质和电子版各1份 |
|  | 软件第三方测评报告 | 2 | 纸质和电子版各1份 |