ATS 软件详细设计说明书

Prepared by 拟制	刘志坤	Date 日期	2013-9-13
Reviewed by 评审人		Date 日期	
Approved by 批准		Date 日期	

修订记录

Date 日期	Revision Version 修订 版本	Sec No. 修改 章节	Change Description 修改描述	Author 作者
2013-9-13	0.0.1		初始版本	刘志坤
2013-9-14			1. 添加数据类型和 API 接口部分。 2. 添加测试点详解章节 3. 添加配置部分	刘志坤

目录

1. 引言

1.1. 编写目的

- 为编码及后期维护提供依据。
- 详细解释系统的架构及组成。
- 本文档的预期读者为系统设计人员、软件开发人员、软件测试人员和项目评审人员。

1.2. 参考资料

《概要设计文档》

1.3. 术语说明

序号	术语	说明
1	ATS	自动测试软件
2	SDK	软件开发包
3	CAT	相机自动测试

2. 概述

2.1. 任务和目标

- 现有的大部分或者全部的人工测试部分转为自动测试，配合生产部门提高生产效率，降低生产成本。
- 探索总结代码重用方法。
- 积累数字图像方面技术，为后期做储备。

3. 系统详细需求分析

3.1. 详细需求分析

3.1.1. 模拟球机自动测试功能

- 能够测试球机水平限位检测，水平速度。
- 能够测试球机垂直限位检测功能。
- 能够测试球机机芯通讯，能够获取测试机芯的型号等参数。
- 能够测试球机 PTZ 功能。
- 能够测试球机红外灯功能。
- 能够测试球机预置位功能。
- 能够自动给球机分配序列号。
- 具有测试报告导出能力。

3.1.2. 高清球机自动测试功能

- 能够测试球机水平限位检测，水平速度。
- 能够测试球机垂直限位检测功能。
- 能够测试球机机芯通讯，能够获取测试机芯的型号等参数。
- 能够测试球机 PTZ 功能。
- 能够测试球机预置位功能。
- 能够测试球机丢包率。
- 能够按照要求给球机设定 IP 地址。
- 能够测试图像延迟时间。

3.1.3. ERP 管理功能

- 能够获取当前生产订单，并让用户选择进行哪个订单的生产。
- 能够根据用户选择的订单，获取该订单的定制单 xml 及相关程序文件。
- 能够在完成升级和测试后，将测试数据及时间、操作员等录入服务器数据库。
- ERP 中能够查询订单生产状态及订单相关的球机序列号及其测试记录。

3.2. 系统环境分析

3.2.1. 硬件环境分析

如下图所示，在整个测试系统中，测试主机需要能够测试模拟球，IP 球机等多种监控设备，并且，需要把测试后的结果归入 ERP 系统进行管理。

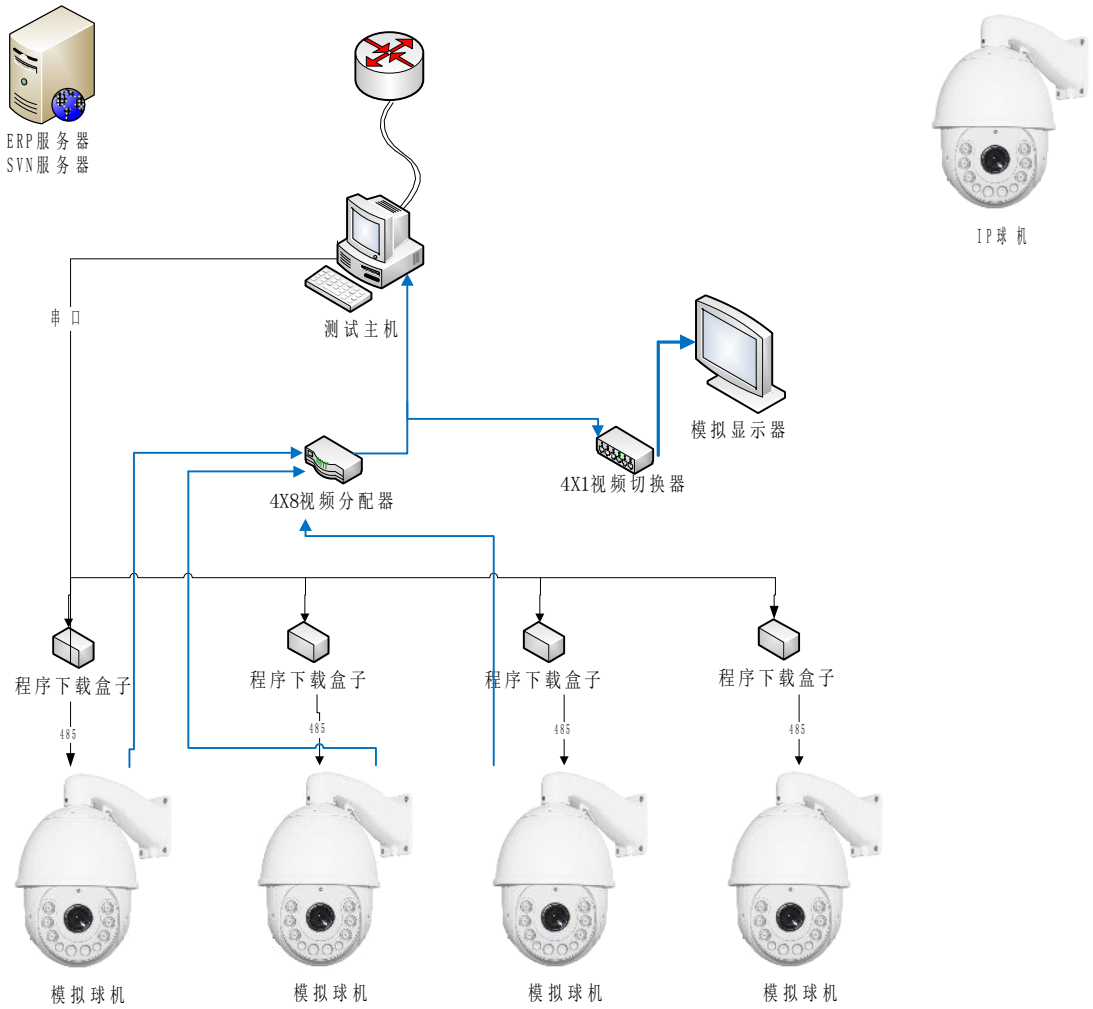

图 1. ATS 硬件环境

3.2.2. 软件环境分析

- 限于大部分厂家目前都只提供了 windows 平台的 SDK，且如视频采集卡等设备只有 windows 的驱动，所以，目前，暂时以支持 windows 为主。
- 在实际开发过程中，做好程序移植性评估工作，减少对特定平台的依赖，减少后期移植和维护的投入。

4. 系统简要设计

4.1. 系统总体架构设计

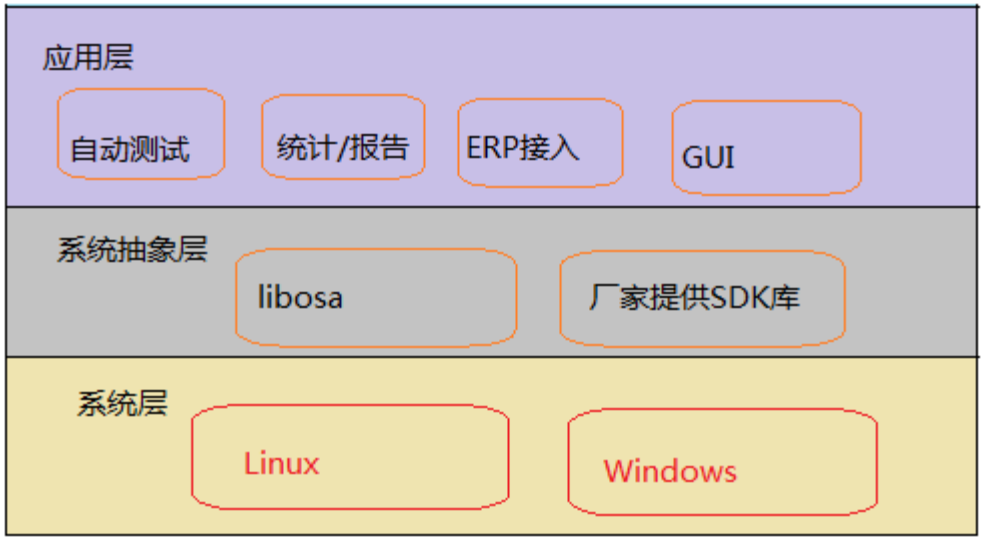

图 2. ATS 系统总体架构图

- 如图 2 所示，ATS 系统主要划分为三层。分别为系统层，抽象层，应用层。
- 系统层为 ATS 运行环境，目前以 Linux 和 Windows 为主.
- 抽象层主要分为两部分，第一部分为操作系统抽象层 libosa（Operation System Abstract Layer），主要提供了对下层系统 Linux 和 Windows 访问的统一接口，屏蔽系统的差异。Libosa 作为一个单独的部分，可以独立出来使用。第二部分为厂家提供的 SDK 库或者公司自己完成的库，主要用来和前端摄像设备进行通信，采集数据，供应用层使用。
- 应用层为 ATS 的主要部分，主要包括如下模块：
  - ✓ 自动测试：包括自动测试框架和自动测试用例。利用用户配置的 XML 文件来进行具体的自动测试。
  - ✓ 统计/报告模块：主要完成对测试结果的统计和报告输出。
  - ✓ ERP 接入：对统计结果，测试结果存入 ERP 归档。
  - ✓ GUI：图形操作界面，用于显示实施视频，测试结果等，GUI 部分需要厂家提供的 SDK 来完成。

4.2. 系统简要流程

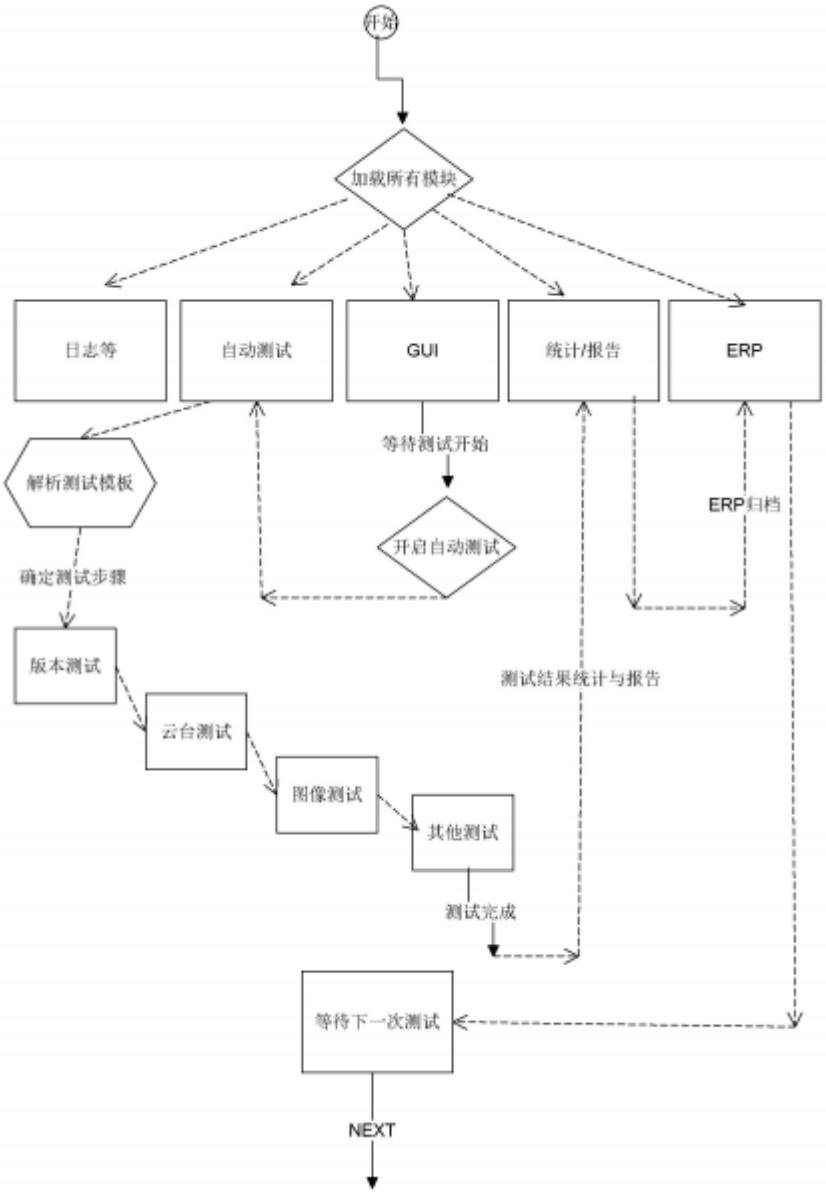

图 3. ATS 系统流程图

如图 3 所示，ATS 系统启动后，会先读取系统配置文件，根据系统配置文件确定需要加载的模块，然后加载相应的模块，初始化系统的信息，接着等待用户的操作。当用户执行开始测试后，自动测试模块开始读取设备相应的测试模板文件，然后解析此款设备支持的测试功能点有哪些，最后根据测试的优先级开始逐步测试，直到测试完成。每一次测试完成后，可以输出报告结果，统计结果等到统计/报告模块。当一台设备测试完毕后，则开始继续测试下一台设备或者等待用户处理。

4.3. 系统界面 GUI 详细设计

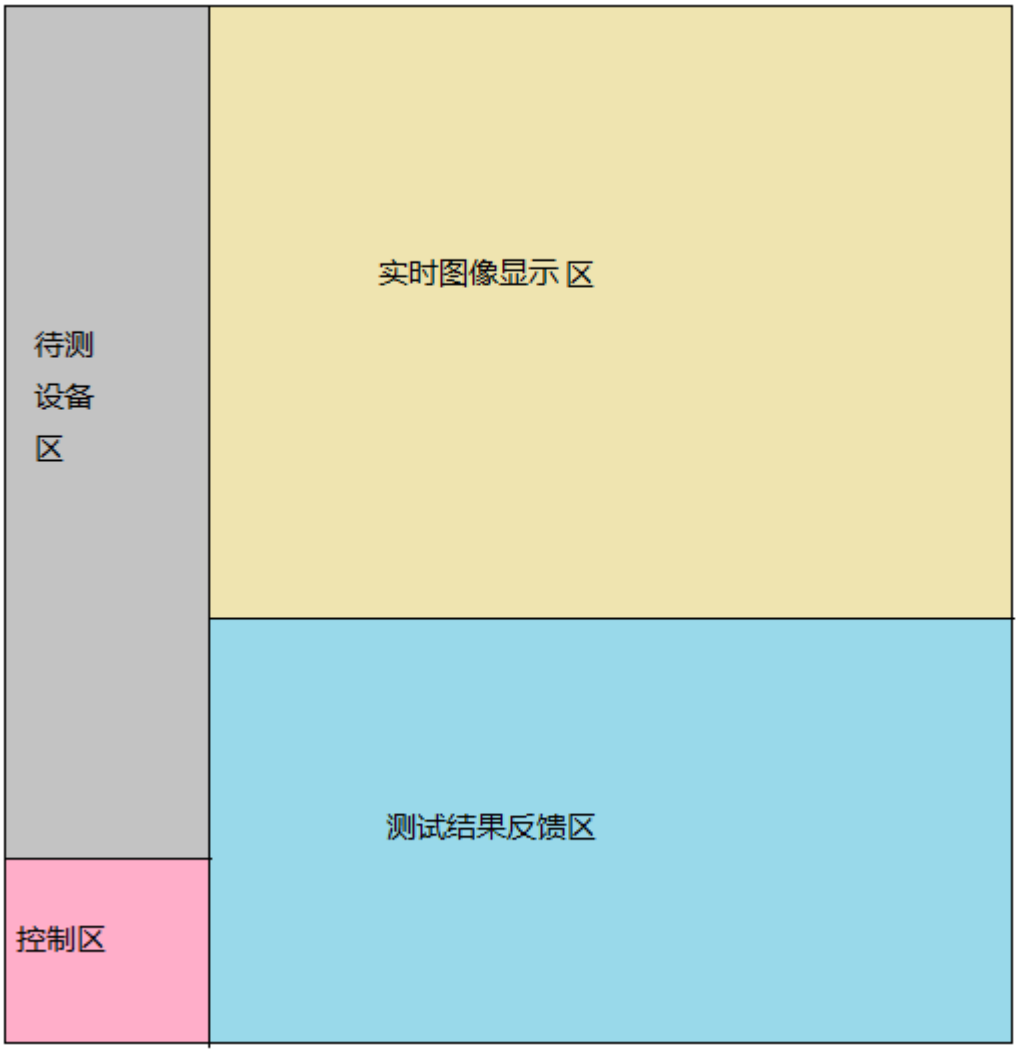

图 4. 系统 GUI 简要设计

如图 4，ATS 系统中，GUI 界面主要可以分为四个区块，分别为待测设备区，控制区，实时图像显示区，测试结果反馈区。

待测设备区用于显示目前等待测试的设备；控制区用于提供用户控制设置操作；实时图像显示区利用相应厂家提供的 SDK 包来显示实时视频流；测试结果反馈区用于反馈测试的结果，统计信息等。

5. 系统模块设计

5.1. libosa 模块

5.1.1. 概述

libosa 模块为操作系统抽象模块，本模块的目的是为了屏蔽操作系统之间的不同而设计的。可以作为一个独立的功能模块使用，而不必关心具体使用的操作系统环境，达到代码重用的特点。

5.1.2. 模块设计

对于 Linux 和 Windows 来说，标准 C 库和 C++库是可移植的，但是对于操作系统的系统调用接口，Windows 不支持 POSIX 系统调用，所以，libosa 的目的是为 Linux 和 Windows 封装一层接口层，屏蔽底层的系统调用接口。

Linux 和 Windows 的系统调用主要区别有：

- 进程管理
- 时间操作
- 系统信息获取
- 底层 I/O 操作
- Socket 通信
- IPC
- 线程

libosa 根据平台的不同，编译成具体的静态库，供其它模块使用。

5.1.3. 数据类型

序号	数据类型	描述
1	osa_list_t	
2	osa_file_t	
3	osa_socket_t	
4		

5.1.4. API 描述

序号	API	描述

5.2. 厂家提供 SDK/公司库模块

5.2.1. 概述

本模块主要的目的是为了统一各个厂家的 SDK 不同而设计的。一般来说，厂家提供的 SDK 中都包括图像处理部分、前端设备访问部分等操作。

5.2.2. 模块设计

5.2.3. 数据类型

序号	数据类型	描述
1		
2		

5.2.4. API 描述

序号	数据类型	描述
1		
2		

5.3. 自动测试模块

5.3.1. 概述

本模块主要用于管理所有测试，包括自动解析确定测试步骤，自动测试，测试后期处理等功能。

对于 ATS 系统来说，因为要涉及到多款设备产品的测试，所以，必须能够适应多款设备的自动测试，所以，自动测试模块根据一个测试模板来匹配测试。

一个测试模板中记录了一款设备的测试过程及测试数据，是对一款设备具体测试的文本化描述。比如，假设有一款名叫 XXX_IPC 的待测设备，此款设备有 100 个测试点，每个测试点提供 2 组测试用例，则系统会总共测试 100*2=200 次。一个典型的测试模板如下：

测试点 1：

名称：VersionTest（版本测试）

优先级：1

测试用例集：

用例 1：

输入数据：无

期望输出：1.1.1（目前 SVN 服务器上的代码版本）

用例 2：无

测试点 2：

名称：SharpnessTest（清晰度测试）

优先级：20

测试用例集：

测试用例 1：

输入数据：一帧 RGB RAW 图像数据

期待输出：OK（清晰达标）

测试用例 2：

输入数据：一帧 RGB RAW 图像数据

期待输出：OK（清晰度达标）

测试点 3：

...

...

5.3.2. 模块设计

对于一次测试来说，有如下特点，输入数据-->实际输出，和期待输出，实际输出表示把输入数据传入进行测试后，输出的实际测试结果，当实际输出和期待输出以某种方式比较后达到某一标准，则判定测试为成功，如果不达标准，则判定本次测试失败。

每一个测试叫做一个测试点，典型的测试点对象如下：

```
typedef struct _ATS_TEST_POINT
{
    // 测试点名字
    char                *name;
    // 测试优先级，数字越小优先级越大
    osa_uint32_t        priority;
    // 测试用例容器
    ATS_TestCaseBox     testCaseBox;
    // 测试结果
    ATS_TestResult      result;
```

```
    // 报告
    ATS_Report          *report;
    // 结果统计
    ATS_TestStatistic   statistic;

    // 链表
    osa_list_t          list;

    // 开始，每个测试点调用一次
    osa_err_t           (*begin)(ATS_TestPoint *self);
    // 启动测试,每个测试用例调用一次，每个测试点可能调用多次
    ATS_TestResult      (*startTest)(void *testCase);
    // 停止测试, 现在没有使用
    void                (*stopTest)();
    // 测试成功后调用,每个测试用例调用一次，每个测试点可能调用多次
    void                (*successFunc)(ATS_TestPoint *self);
    // 测试失败后调用,每个测试用例调用一次，每个测试点可能调用多次
    void                (*failedFunc)(ATS_TestPoint *self);
    // 结束,每个测试点调用一次
    void                (*end)(ATS_TestPoint *self);
}ATS_TestPoint;
```

其中，list 链表用来管理所有的测试点，startTest 接口函数用来开始测试，startTest 函数会返回一个测试结果，成功（ATS_TEST_SUCCESS)或者失败（ATS_TEST_FAILED)，当返回成功时，由测试管理器自动调用 successFunc 函数接口，同理，当测试失败时，则会自动调用 failedFunc 函数接口。report 参数是一个指向报告对象的指针。

一个典型的测试点工作过程如下：

① 把自己注册到测试点链表中。

② 测试管理器遍历测试点链表，调用 ATS_TestPoint 的 startTest 接口，开始测试。

③ 测试结束，测试管理器根据 startTest 接口的返回值，判断是否测试成功，成功则调用 ATS_TestPoint 的 successFunc，失败则调用 ATS_TestPoint 的 failedFunc。

④ 设置 ATS_TestPoint 的 result 参数，表明本次测试的测试结果。

一个典型的测试点需要实现的接口如下：

- startFunc：开始测试函数，比如版本测试就是获取设备版本，并和测试用例中的期望值比较。根据比较结果返回结果。
- successFunc：成功后的回掉函数：对于版本测试无。
- failedFunc：失败后的回掉函数：对于版本测试而言，失败后意味着设备的版本比 SVN 上的版本低，则失败函数中要做的事情就是去 SVN 上获取最新升级文件，调用板子升级接口进行升级。然后调用 ATS_TestPoint 下的 report 报告接口来报告升级结果。

5.3.3. 数据类型

序号	数据类型	描述
1	ATS_TestPointBox	测试点容器
2	ATS_TestPoint	测试点类
3	ATS_TestStatistic	测试统计

5.3.4. API 描述

序号	API	描述
1	ATS_TestModuleInit	测试模块初始化
2	ATS_TestModuleExit	测试模块退出
3	ATS_TestPointRegister	注册一个测试点
4	ATS_TestPointUnregister	卸载一个测试点
5	ATS_TestParseTemplete	解析测试点模板文件
6	ATS_TestStartAll	开始所有测试

5.4. 统计/报告模块【未完成】

5.4.1. 概述

本模块主要是用于统计/报告测试信息，具体要输出什么信息完全由用户确定。ATS_TestPoint 结构体中的 report 指针就是指向统计模块对象，report 模块只负责读写报告，不负责具体的报告格式。

5.4.2. 模块设计

一个典型的报告模块如下：

```
typedef struct _ATS_REPORT_INTERFACE
{
    // 打开一个报告文件，可以是文件，可以是STDOUT等
    void        *(*open)(const char *reportFile);
    // 关闭一个报告文件
    void        (*close)(void *reportFd);
    // 写报告到报告文件中
    void        (*write)(void *reportFd, const char *buf, osa_size_t size);
    // 从报告文件中读取文件
    void        (*read)(void *reportFd, char *outBuf, osa_size_t size);
    // 控制报告文件
    void        (*ctrl)(void *reportFd, osa_uint32_t cmd, void *arg);
}ATS_Report;
```

Open 用于打开报告文件，close 用于关闭报告，write 用于写报告，read 用于读取报告，ctrl 用于控制报告文件，从上面可以看出，报告模块只负责写报告和读报告，不负责报告的格式。因为报告的具体格式不是报告模块的责任。

5.4.3. 数据类型

序号	数据	描述
1	ATS_Report	报告接口

5.4.4. API 描述

序号	API	描述
1	ATS_ReportModuleInit	报告模块初始化
2	ATS_ReportModuleExit	报告模块退出

5.5. ERP 模块【未完成】

5.5.1. 概述

本模块是为了和 ERP 相连接而设计的，测试完成后，需要对测试的一些结果归入 ERP，需要和 ERP 打交道，所以，本模块是 ERP 的一个接口层，不涉及具体的 ERP 系统。

5.5.2. 模块设计

对于一般的 ERP 系统来说，可以通过 ERP 系统对外提供的 API 接口来访问 ERP 系统的数据，但是有的 ERP 系统有可能没有提供 API 接口，这时候也可以

直接访问 ERP 的数据库来获取数据，但这样操作的风险性较大。

对 ERP 系统，封装如下操作接口，可以方便上层应用层对 ERP 的访问。

```
typedef struct _ATS_ERP_INTERFACE
{
    // 初始化ERP
    osa_err_t       (*init)();
    // 退出ERP
    void            (*exit)();
    // 连接ERP系统
    void           *(*connect)(const char *erpFile, osa_uint32_t timeout);
    // 断开ERP系统
    osa_uint8_t     (*disconnect)(void *erpFd);
    // 向erp发送数据
    osa_err_t       (*send)(void *erpFd, void *data, osa_size_t size);

}ATS_Erp;
```

5.5.3. 数据类型

序号	数据	描述
1	ATS_Erp	ERP 接口
2		

5.5.4. API 描述

序号	API	描述
1	ATS_ErpModuleInit	初始化 ERP 模块
2	ATS_ErpModuleExit	ERP 模块退出
3		

5.6. GUI 模块【未完成】

5.6.1. 概述

GUI 模块是为了方便一般用户操作而设计的，所以，其工作就是方便用户的操作，不涉及具体的测试过程。

5.6.2. 模块设计

对于不同系统来说，GUI 是程序可移植部分中一个工作量比较大的地方，根据责任原则，GUI 只是负责和用户进行交互，不负责具体的处理。所以，可以把 GUI 模块单独独立出来实现。把 GUI 定为前端，其他模块定为后端，前端负责

接收用户命令，然后传递到后端，后端进行具体的处理，这样可以使前端和后端分离，减少移植成本。

GUI 前端和处理后端的通信可以采用本地 Socket 或者其他 IPC 方式通信。可以简化设计，分离模块。

5.6.3. 数据类型

序号	数据	描述
1		

5.6.4. API 描述

序号	API	描述
1	ATS_GuiStart	启动 GUI
2		

6. 系统其他设计

6.1. 模块管理设计

6.1.1. 概述

ATS 系统中，把整个系统划分为多个模块，对于系统中的不同模块，当模块数量比较少时，可以很好的管理模块，但是随着后期模块的增多，所以，有必要统一管理所有的模块。

6.1.2. 详细设计

模块管理器相当于管理者角色，用于管理系统中存在的模块，模块管理者提供固定的模块注册接口供具体的模块使用，为了让模块管理器知道特定模块的存在，必须由模块自己把自己注册到模块管理器中。当不再需要模块的时候，卸载相应模块即可。

对于系统中的每一个模块来说，都是可以配置的，可以单独打开或者关闭某些模块。模块读取配置文件，解析后决定启动哪些模块，所以，定义了一个 ATS_ModuleConf 接口类来实现模块自己对配置的解析操作。这种方式同样适用于测试模板部分。

```
struct _ATS_MODULE
{
    // 模块名字
    char            name[OSA_NAME_MAX];
    // 模块状态，ON或者OFF
    osa_uint8_t     state;
    // 模块配置文件
    ATS_ModuleConf  cf;
    // 模块入口
    osa_err_t       (*entry)(ATS_Conf *conf, int argc, char **argv);
    // 模块出口
    void            (*exit)(ATS_Conf *conf);
};
```

在 ATS_Module 中，name 表示模块的名字，state 表示模块的状态，可以为打开或关闭状态。cf 为模块的配置处理接口，用于从配置文件中解析出模块自己配置信息。entry 接口为模块的入口函数，exit 为模块退出函数接口。

6.1.3. 数据类型

序号	数据类型	描述
1	ATS_ModuleConf	模块配置接口
2	ATS_Module	模块对象
3		

6.1.4. API 描述

序号	API	描述
1	ATS_ModuleInitAll	初始化系统中的所有模块
2	ATS_ModuleFind	根据名字查找模块
3	ATS_ModuleRegister	注册模块
4	ATS_ModuleUnregister	卸载模块

6.2. 图像存储格式设计

6.2.1. 概述

在本测试系统中，因为要涉及到一些图像方面的处理，所以，有必要统一化图像存储的格式，方便处理。

6.2.2. 详细设计

开源库 Opencv 中实现了 IplImage 图像格式，其存储方式为顺序存储，比如单通道图像直接按字节存储，三通道 RGB 按 BGR,BGR,BGR 顺序存储。可以参考此格式来设计，方便和 Opencv 进行转换。

具体的图像存储方式如下：

```
typedef struct _ATS_IMAGE
{
    osa_uint32_t    width;      // 图像宽度
    osa_uint32_t    height;     // 图像高度
    ATS_ImageDepth  depth;      // 每个像素的深度
    osa_uint32_t    channels;    // 通道數
    osa_uint32_t    width_step; // 每行宽度步进
    osa_uint32_t    size;       // 图像大小
    ATS_ImageRoi    roi;        // ROI区域
    osa_uint32_t    roi_flags;  // ROI標誌
    osa_uint8_t     *scan0;      // 图像数据
}ATS_Image;
```

ATS_Image 结构体表示了一幅图像，其中 scan0 是图像数据的起始地址，存

储格式和 IplImage 方式一样，width_step 表示每一行的字节数，这里一般会以 4 字节对齐，所以，width_step 并不是总等于 width * depth/8 * channels。

6.2.3. 数据结构

序号	数据类型	描述
1	ATS_Image	图像对象

6.2.4. API 描述

序号	API	描述
1		

6.3. 配置文件设计

6.3.1. 概述

ATS 系统中，因为要满足多种设备的测试要求，所以，必须添加配置文件来让系统自动选择测试过程。

常用的配置文件格式有 Windows 的 INI 文件，普通文本文件，XML 文件，JSON 配置文件等。通过对 ATS 的分析，最后选定使用 XML 作为配置文件，使用 tinyxml 库作为 XML 的解析库，tinyxml 是 C++开发的库，所以，在和 C 语言集成时需要排除 C++特有的属性，使用和 C 兼容的方式来完成 C 和 C++之间的相互调用。

通常，C 和 C++相互调用要通过指针来完成，特别是 void * 指针，灵活运用 void *指针，可以简化步骤。

7. 测试点设计【未完成】

7.1. 版本测试

7.1.1. 测试用例

名称：VersionTest

优先级：1

测试用例集：

用例 1：

输入：SVN 服务器地址

期望输出：OK

测试结果：无

7.1.2. 实现方式

测试系统从待测设备上获取到待测设备的版本号，和 SVN 上的最新软件版本对比，如果发现待测设备的版本号低于 SVN 上软件的版本号，则调用相关接口升级。

7.2. PTZ 测试

7.2.1. 测试用例

名称：PTZTest

优先级：默认

测试用例集：

用例 1：

输入：

期望输出：

用例 2：

输入：

期望输出：

用例 3：

输入：

期望输出：

测试结果：

7.2.2. 实现方式

7.3. 预置位测试

7.3.1. 测试用例

1. 模拟球

名称：PresetTest

优先级：默认

测试用例集：

用例 1：

输入：

期望输出：

用例 2：

输入：

期望输出：

测试结果：

2. 网络球

名称：PresetTest

优先级：默认

测试用例集：

用例 1：

输入：

期望输出：

用例 2：

输入：

期望输出：

测试结果：

7.3.2. 实现方式

1. 模拟球：通过 PELCO-D 绝对位置定位指令，将球机定位到某个位置，进行其它测试后，调用该预置位，通过采集到的图像判别预置位的准确性。

2. 网络球：

7.4. 图像延时测试

7.4.1. 测试用例

7.4.2. 实现方式

7.5. 丢包率测试

7.5.1. 测试用例

7.5.2. 实现方式

7.6. 视频质量测试

7.6.1. 测试用例

7.6.2. 实现方式

7.7. 红外灯测试

7.7.1. 测试用例

7.7.2. 实现方式

7.8. CDS 测试

7.8.1. 测试用例

7.8.2. 实现方式

8. 附录

9.