联 众 科 技

HIS3 系统架构设计说明书

首次发布日期：	2006 年 8 月 10 日
文档编号：	MediCARE-TS-DES-TMP-STR-v0.1
读者：	项目经理，项目团队，高层经理，研发部经理，SQA
审阅者：	项目经理
保密等级：	绝 密

修订历史

A-增加 M- 修订 D- 删除

版本	日期	变更类型（A-M-D）	修订原因	修改人
1．0	2006-8-9	A	新增	张楚俊

目　录

1 文档介绍

1.1 目的

提示： 列出本文档的主要目的

1.2 范围

提示： 列出本文档的所阐述的范围边界

1.3 读者对象

项目经理 ， 项目团队 ， 高层经理 ， 研发部经理 ，SQA

1.4 参考文献

1、王子健 &张楚俊 《项目技术研究报告 [MediCARE-TS-DES-TMP-STU].doc》 MediInfo 2006-03-24

2、张楚俊 《项目技术方案 [MediCARE-TS-DES-TMP-CAS].doc 》 MediInfo 2006-8-5

3、张友生 & 王胜祥 & 殷建民《系统架构设计师教程》电子工业出版社 2006-1-1

4、王春森《系统设计师（高级程序员）教程》清华大学出版社 2001-5-1

1.5 术语与缩写

缩写、术语	解 释
实体	数据集合；它由两部分构成，一部分是实体的架构（定义实体的数据结构），另一部分是实体的数据（由数据格式组合而成）
Subprogram	Oracle PRO * C 的集合，它完成映射实体和 Oracle 数据库的工作
Program	一些 C 语句业务逻辑和对 Subprogram 调用的集合
XMLPath	提供了中间层实体数据结构及一些公共函数的 C 语言包
EntityManager	用于管理中间层实体的架构，实现实体和数据的映射

2 物理架构设计

描述本项目各子系统之间的物理分布并以图形式描述。

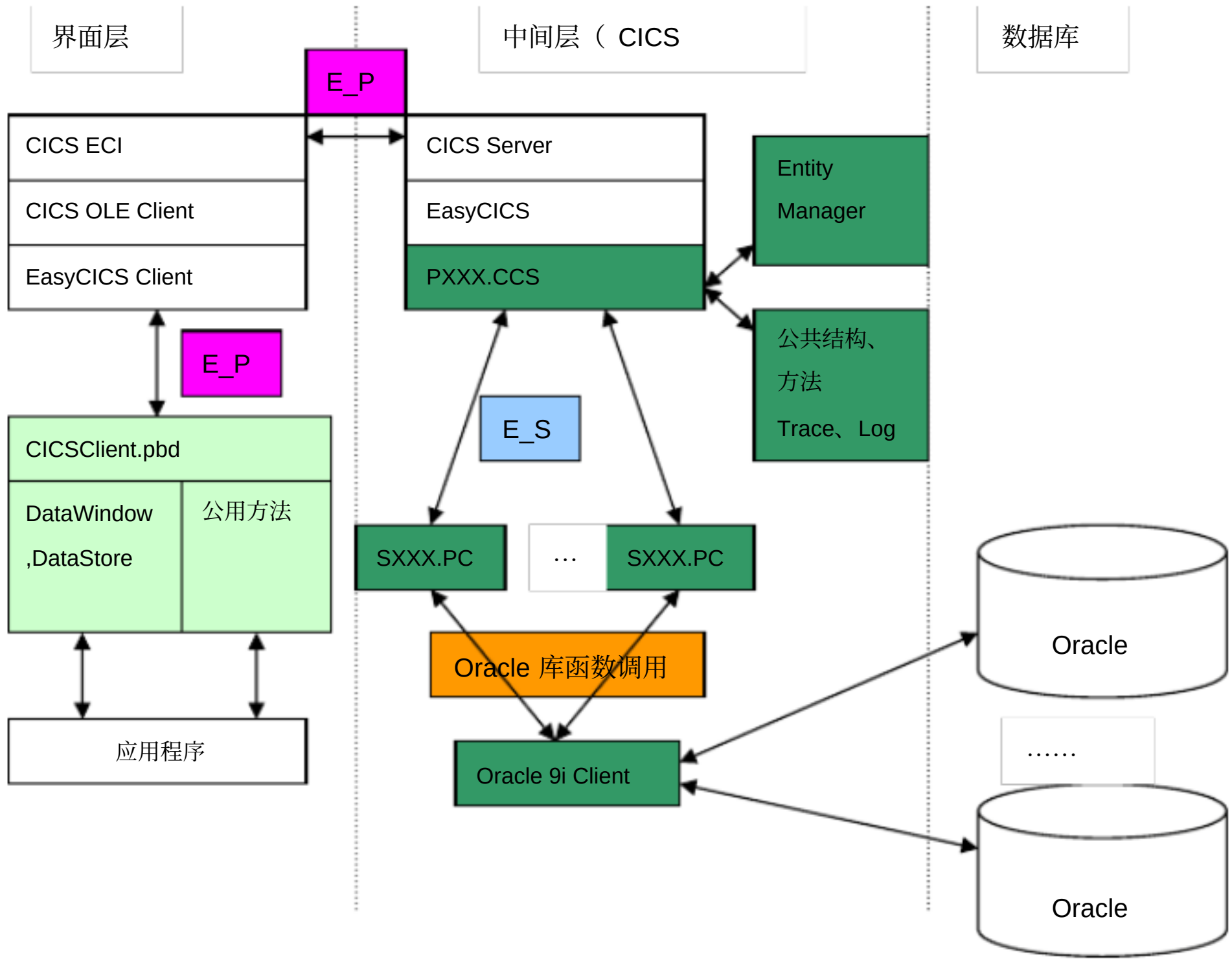
3 技术架构设计

3.1 总体架构

由于 CICS 对业务处理的支持能力不强，所以我们需要在 CICS 上封装一层，提供数据的打包和解包、公用结构、公用函数和一些对象的管理功能，并在此架构上提供一个开发工具包，以便开发人员能够快速、正确的开发业务组件和客户端。

3.1.1 系统架构分析

在分析了本项目的业务要求基础上，并结合以前的开发经验及 IBM 咨询师的建议后，我们设计了一个符合本项目的基于 CICS 的系统架构，它可以用下图来描述：



E_P：用于客户端和 CICS Server 之间的数据传输，客户端到 CICS Server 和 CICS Server 到客户端的传输结构是不一样的，具体的结构见本节附录。

E_S：用于 Subprogram 之间的数据传输，在本项目中一般是一个 EntityRow 结构或一组参数。

为了简化开发和提高组件的运行效率，我们应该尽量使 E_P 等价于 E_S。

3.1.2 功能描述

针对系统架构分析，我们将需要封装的内容划分为三个部分：客户端架构、中间层架构及开发工具包。

客户端架构需要实现以下功能：

封装 CICS 客户端、 EasyCICS 编程，让开发人员可以透明的使用它；

实现数据窗口和实体的映射，包括查询条件的映射和更新数据的映射；

提供调用 Program 的方法；

XXX

中间层架构需要实现以下功能：

实体数据结构，方便开发人员对客户端传送到中间层的数据进行操作；

实体架构管理，用于映射实体和数据库；

公共函数，方便开发人员的编程；

开发工具包需要实现以下功能：

实体生成和管理；

实体和数据库对象对应关系的生成和管理，即 Subprogram 生成和管理；

Porgram 生成和管理，主要是 Program 模板生成及管理 Program、Subprogram 之间的对应关系；

多平台的编译脚本文件生成；

XXX

3.2 业务流程

3.2.1 概述

结合本项目的实际情况，我们将系统中的业务分为三类，即查询、更新、存储过程，针对每种处

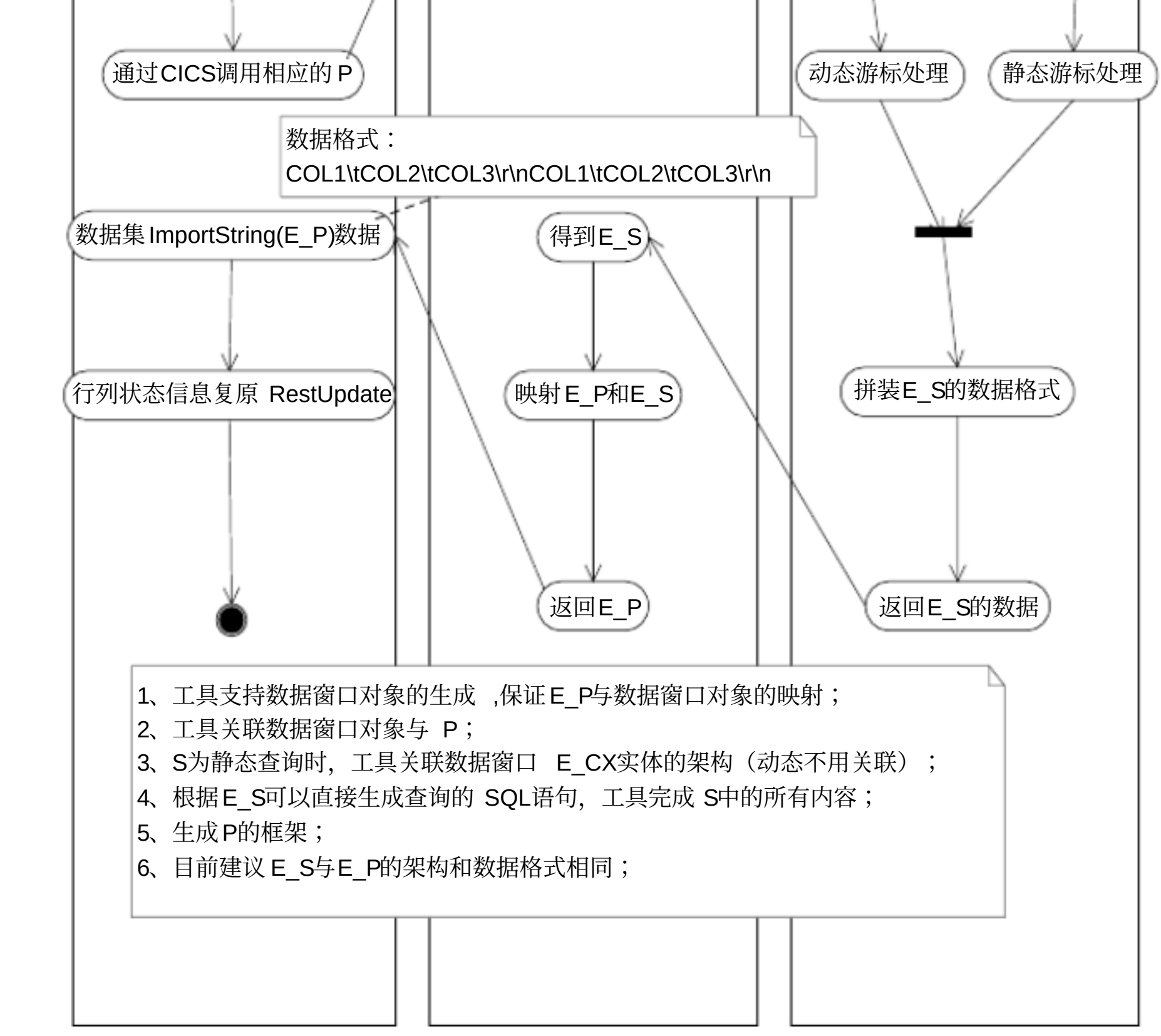
Comment [Z1]: 其他界面层需要完成的功能，如：共有缓存

Comment [Z2]: 开发助手完成的功能

理流程我们定制了不同的处理方式，如下：

3.2.2 查询

通过分析 PowerBuilder 、 CICS，并结合业务需要，我们得到查询的活动图如下：



通过上图的分析，我们将查询操作划分为客户端、中间层、工具三个方面，下面分别介绍下这三个方面如何实现：

客户端

客户端数据集 DataWindow 控件（ DataStore） 封装 RetrieveData(string E_CX)方法。其实现思路如下：

第一步：取数据窗口对象存放的基本信息，得到相应的 Program ；

第二步：通过 CICS 客户端和 EasyCICS 调用中间层相应的 Program，得到 E_P 数据；

第三步：利用 importstring 把 E_P 数据绑定到数据窗口上；

第四步：行列状态信息复原。

在此过程中，需要工具支撑如下内容：

1、工具生成数据窗口保证与 E_P 实体架构一致（否则，数据无法直接绑定） ；

2、工具关联数据窗口和 Program 的关系；

3、工具关联数据窗口 E_CX 实体的架构；（就象二层时数据窗口的 Retrieve 的参数）

服务器端

服务器端需要生成对应的 Program 和 Subprogram。

- 查询 Program 实现的思路：

  第一步：通过 CICS 服务器取得传入参数 E_CX 实体；

  第二步：解析传入的 E_CX 实体，做为 Subprogram 查询的入参；

  第三步：调用 Subprogram 查询；

  第四步：映射 E_P 和 E_S；［最好 E_P 和 E_S 的规格相同，这样映射就不用做了］ ；

- Subprogram 分为 动态和静态 两种处理方式， Subprogram 由工具生成，需要实现如下内容：

  动态：传入参数个数不确定；

  第一步：根据传入的 E_CX 实体，组装查询条件；

  第二步：动态 SQL、游标处理；

  第三步：拼装 E_S 实体的 数据格式 ；

静态：传入参数个数已确定。

第一步：静态　SQL、游标处理；

第二步：拼装　E_S 实体的数据格式；

为了实现这样的　Program、Subprogram，需要工具支撑如下内容：

1、　根据 E_S 实体的架构能够自动生成　　Subprogram.PC 文件；（ SQL 语句自动生成）

2、映射　E_S 到 E_P 上，并映射　E_P 到数据窗口对象上；

3、能够设计数据窗口的查询参数（需要与　　S_select 相关联）；

4、能够生成　Program.ccs 文件的框架。

附录一，　CICS Server 到客户端的传输结构：
状态 |错误信息 |||返回的数据
其中状态的取值：　-1 失败　1 成功
返回的数据：列　1\t 列 2\t …..列 N\r\n 列 1\t 列 2\t …..列 N\r\n …..

附录二，客户端到　CICS Server 的传输结构：

```
<EntitySet EN =　　”  ”  EC= ”  ” >
    <Entity EN=　　”E_BRXX”  ET= ”  ”  DS= ”  ”  RC= ”  ”  FT= ”{COL1}=:A1 and {COL2}=:A2　　”  V= ”@”
        <Row RS = ”-3 ”  CC= ”  ”  >
            < Col CN=　”COL3”  RV= ”  ”  CV= ”  ”  S= ”  ”  />
            < Col CN=　”COL4”  RV= ”  ”  CV= ”  ”  S= ”  ”  />
            < Col CN=　”COL5”  RV= ”  ”  CV= ”  ”  S= ”  ” >
            ……
        </Row>
    </ Entity >
    …… .
</EntitySet>
```

DS　　DynSQL ：动态 SQL 0、静态； 1、动态
ET　　EntityType ： 1、查询　2、更新
RS　　RowState：　－3、数据窗口所需的列　　－2、变量绑定
S　　State　　：-1、回传的列　-2、条件
FT　　Filter　　：查询条件
[用于拼装查询条件，　　CN 是逻辑列的名字，拼装时需要映射到物理列名上　　]

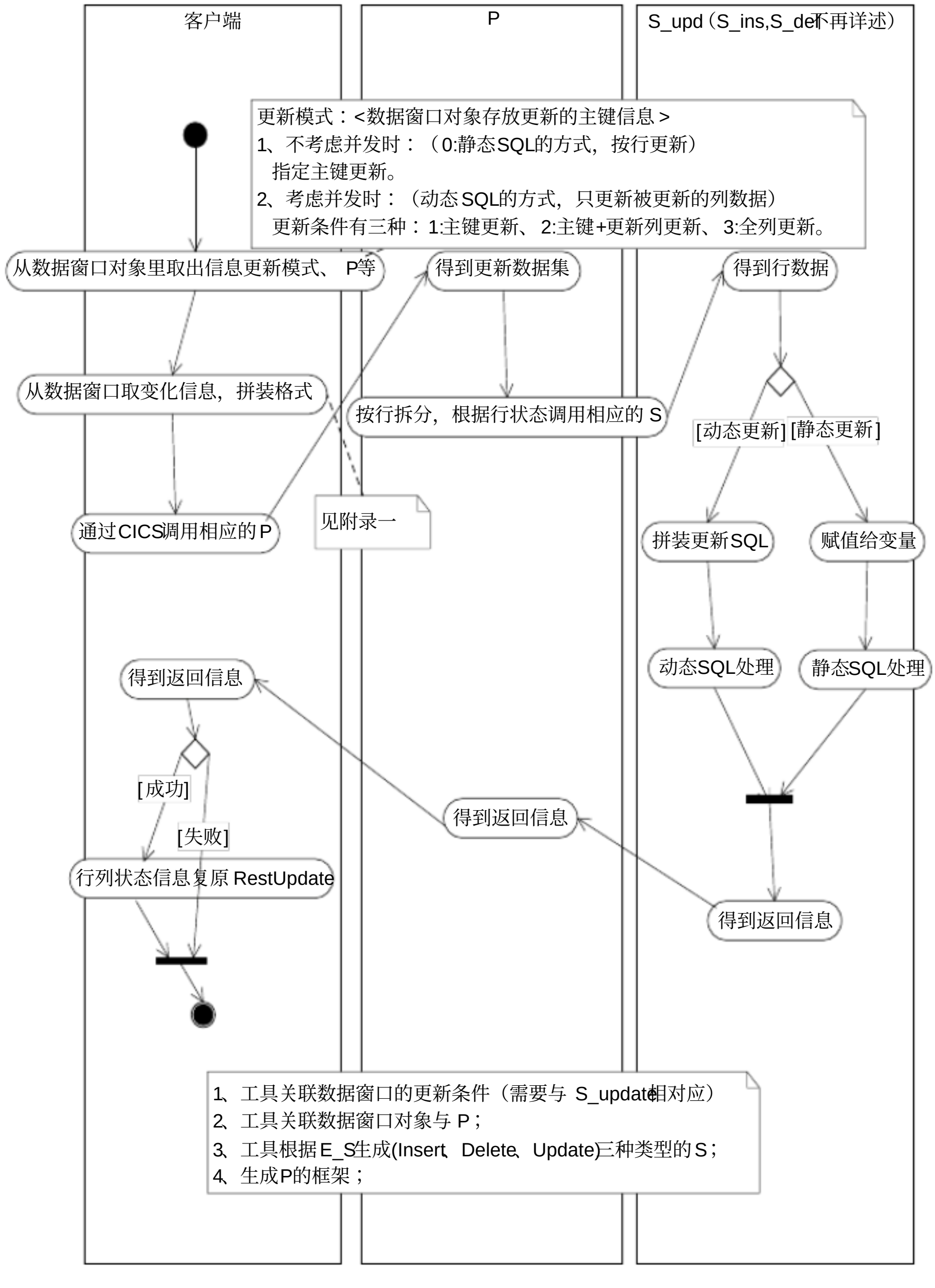
附录三，数据窗口的框架

```
release 10;
…… //省略数据窗口格式信息
```

```
//架构顺序与　E_P 一致
table(column=(type=char(100) updatewhereclause=no name=a dbname="a" )
  column=(type=date updatewhereclause=no name=b dbname="b" )
  ……
/* 最后一行为行状态信息　　*/
  column=(type=number updatewhereclause=no name=rowstate dbname="rowstate" )
  )
//列名的标题显示
text(band=header alignment="2" text="A" border="0" color="33554432" x="9" y="8" height="76" width="320" html.valueishtml="0" name=a_t visible="1" font.face="Tahoma" font.height="-12" font.weight="400" font.family="2" font.pitch="2" font.charset="0" background.mode="1" background.color="536870912" )
text(band=header alignment="2" text="B" border="0" color="33554432" x="338" y="8" height="76" width="320" html.valueishtml="0" name=b_t visible="1" font.face="Tahoma" font.height="-12" font.weight="400" font.family="2" font.pitch="2" font.charset="0" background.mode="1" background.color="536870912" )
//列的标题显示
column(band=detail id=1 alignment="0" tabsequence=10 border="0" color="33554432" x="9" y="8" height="88" width="320" format="[general]" html.valueishtml="0" name=a visible="1" edit.limit=0 edit.case=upper edit.focusrectangle=no edit.autoselect=yes edit.autohscroll=yes font.face="Tahoma" font.height="-12" font.weight="400" font.family="2" font.pitch="2" font.charset="0" background.mode="1" background.color="536870912" )
column(band=detail id=2 alignment="0" tabsequence=20 border="0" color="33554432" x="338" y="8" height="88" width="320" format="[general]" html.valueishtml="0" name=b visible="1" edit.limit=0 edit.case=any edit.focusrectangle=no edit.autoselect=yes edit.autohscroll=yes font.face="Tahoma" font.height="-12" font.weight="400" font.family="2" font.pitch="2" font.charset="0" background.mode="1" background.color="536870912" )
//实体附加信息,　name 为 entity__info ；tag 存放信息为：　实体名 ^查询用到的　P^查询的参数（用　|分格）^更新用到的　P^更新条件的模式（　0：不考虑并发；　2：主键更新；　3：主键 +更新列更新；　4：全列更新）　|主键（用　|分格）
text(band=footer alignment="0" text="text" border="0" color="33554432" x="338" y="288" height="76" width="320" html.valueishtml="0" name=entity__info tag="E_ABC^P_Select^A|B^P_Update^1|A|B" visible="1" font.face="Tahoma" font.height="-12" font.weight="400" font.family="2" font.pitch="2" font.charset="0" background.mode="2" background.color="1073741824" )
```

3.2.3　更新

通过分析　PowerBuilder 、 CICS，并结合业务需要，我们得到查询的活动图如下：



通过上图的分析，我们将更新操作划分为客户端、中间层、工具三个方面，下面分别介绍下这三个方面如何实现：

客户端

客户端数据集 DataWindow 控件（ DataStore） 封装 UpdateData() 方法，其实现思路如下：

第一步：取变化的数据，以某种数据格式进行组织成 ls_Data；

第二步：取数据窗口对象存放的基本信息，得到相应的 P；

第三步：通过 CICS 客户端调用中间层相应的 Program，传入 ls_Data；

第四步：如果更新成功，行列状态信息复原；失败，提示失败信息。

为了能够实现这样的功能，需要工具支撑如下内容：

1、工具关联数据窗口的更新条件（需要与 S_update 相对应）；

2、工具关联数据窗口和 Program 的关系；

服务器

服务器端需要生成对应的 Program 和 Subprogram。

- Program 实现的思路：

第一步：通过 CICS 服务器取得传入的数据；

第二步：按行解析入参；

第三步：根据行状态调用相应的 Subprogram（1：Insert：2：Delete；3：Update）；

第四步：返回结果是否成功；

- Subprogram 由工具生成：

主要有三种 Subprogram（Insert、Delete、Update），这三种 Subprogram 都是由 Subprogram 定制工具根据 E_S 自动生成的，其中 Insert、Delete 比较简单可以直接用静态 SQL 实现，这里不过多描述。

Update 的 Subprogram 的生成分成两种情况，一种是需要控制并发的更新；一种是不需要控制并发的更新。控制并发的更新需要用动态 SQL 的方式，比如表 ABC 只更新了 A 字段则 Update ABC set A=:1 where PK=:2 ，只更新了 B 字段则 Update ABC set B=:1 where PK=:2 ；不需要控制并发则用静态 SQL，比如 Update ABC set A=:1,B=:2,C=:3 where PK=:4; 每次传入 4 个参数，无论值是否发生变化。

Subprogram 的实现步骤：

第一步：对传入的单行实体数据进行解析；

第二步：类型映射；

第三步：动态 /静态 SQL 处理；

第四步：返回处理信息。

为了实现这样的 Program、Subprogram，需要工具支撑如下内容：

1、工具提供更新数据的 Program 的框架；

2、工具根据 E_S 生成 (Insert、Delete、Update)三种类型的 Subprogram；

3、根据选择 S_update 的是否并发控制，生成不同的 S_update(动态、静态)；

4、S_update 的更新条件生成（根据主键、主键 +更新列、全列 拼条件）。

附一：更新传入数据的数据格式：

```
<EntitySet EN = ”” EC= ”1”>
<Entity EN=”E_实体名字 ” ET= ”1” DS = ”1” RC=”DS1”://DynSQL , 0、静态； 1、动态 */
    <Row RS=”3” CC=2 >更新行
    /*RS ：RowState, -1、Select 0、Unchanged 1、Added 2、Deleted 3、Modified */
        <Col CN= ”A” RV= ”” CV= ”” S= ”1”>
        /* 列的逻辑名、原值、现值、状态： 1、更改, 0、未更改 */
        </Col>
        <Col CN= ”B” RV= ”” CV= ”” S= ”0”>
        </Col>
    </Row>
    <Row RS = ”2” CC=2 >删除行
        <Col CN= ”A” RV= ”1” CV= ”1” S= ”0”>
        /* 列的逻辑名、源值、现在值、更改状态： 1、更改, 0、未更改 */
        //只有主键
</Row>
    <Row RS = ”1” CC=2 >插入行
        <Col CN= ”A” RV= ”” CV= ”1” S= ”1”>
        /* 列的逻辑名、源值、现在值、更改状态： 1、更改, 0、未更改 */
        </Col>
            <Col CN= ”B” RV= ”” CV= ”1” S= ”1”>
            </Col>
    </Row>
    <Row>
    ……
    </Row>
</Entity>
</EntitySet>
```

说明：新增、删除行不传原值

每个 Tag 具体的含义如下：

EntitySet ： EN：EntitySetName 实体集名称

EC：EntityCount 实体数

Entity ： EN：EntityName 实体名称

ET：EntityType 实体类型 1、查询 2、更新

DS：DynSQL SQL 类型 1、静态； 2、动态

RC：RowCount 行数

Row ： RS：RowState 行状态 -2、Selected -1、Unchanged 1、 Added 2、 Deleted 3、 Modified

CC：ValueCount 值个数

Col ： CN：ColName 列名

RV：RegionValue 原值

CV：CurrValue 现值

S： State 状态, -2、条件 Filter -1 、回传的列 Value 1、未更改 UnChange 2、更改 Change

附录二：传出的数据的格式

状态 |错误信息 |||返回的数据

状态： -1 失败 1 成功

3.2.4 存储过程

存储过程的调用和查询的过程基本是一样的，但需要在工具中添加一个模块以自动生成调用存储过程的 Subprogram。

4 业务架构设计

5 系统专题设计

5.1 客户端架构

5.1.1 概述

5.1.2 界面方案

5.1.3 数据缓存

5.2 中间层架构

5.2.1 概述

根据中间层需要完成的功能，我们将中间层分为 Entity 数据结构、实体架构管理和公用方法三个部分的内容，下面分别介绍这三个部分所要完成的功能。

5.2.2 Entity 数据结构

由于客户端传输到中间层的数据是一份 XML，当需要对这份 XML 进行读写时比较麻烦，为了简化对 XML 数据的操作，我们将其解析为一个与 .NET Framework 1.1 中的 DataSet 类似的结构，它包含了下面这些结构：

EntitySet ：一组实体的集合，提供了对实体的管理功能。其属性和方法如下：

char	*EntitySetName ;	实体集名称
long	EntityCount ;	实体总数
Entity	**Entities ;	实体集中实体的集合
long	AcceptChanges(EntitySet *pEntitySet) ;	提交实体集更改
long	Clear(EntitySet *pEntitySet) ;	清空实体集数据

Comment [Z3]: 此处需要界面组补充，可能在其下还有一些子主题，请一并补充

EntitySet*　　Clone(EntitySet *pEntitySet) ;　　复制实体集架构

EntitySet*　　Copy(EntitySet *pEntitySet) ;　　拷贝实体集架构和数据

EntitySet*　　GetChanges(EntitySet *pEntitySet) ;　　获取实体集更改

Bool　　HasChanges(EntitySet *pEntitySet) ;　　判断实体集是否存在更改

long　　RejectChanges(EntitySet *pEntitySet) ;　撤销实体集更改

Entity*　　CreateEntity(EntitySet *pEntitySet) ;　　创建指定名称的实体

Entity*　　GetEntity(EntitySet *pEntitySet, const char *pEntityName) ;　　获取指定名称的实体

long　　AddEntity(EntitySet *pEntitySet, Entity *pEntity) ;　　向实体集中添加实体

long　　Remove(EntitySet *pEntitySet, const char *pEntityName) ;　　移除实体集中的实体

long　　SetEntitySetName(EntitySet *pEntitySet , const char *pEntitySetName) ;　　设置实体集的名称

Entity ：提供了实体数据和架构的管理功能。其属性和方法如下：

EntitySet　　*EntitySet ;　　实体所属的实体集

char　　*EntityName ;　实体名称

char　　*Caption ;　　实体标题，即中文名称

long　　CaseSensitive ;　　是否区分大小写

SQLType　　SQLType ;　　SQL 类型

EntityType　　EntityType ;　　实体类型

long　　RowCount ;　　行数

long　　ColumnCount ;　　列数

EntityColumn　　**Columns ;　　实体列的集合

EntityRow　　**Rows ;　　实体行的集合

long　　AcceptChanges(Entity *pEntity) ;　　提交实体更改

long　　Clear(Entity *pEntity) ;　　清空实体数据

Entity*　　Clone(Entity *pEntity) ;　　复制实体架构

Entity*　　Copy(Entity *pEntity) ;　　拷贝实体架构和数据

Entity*　　GetChanges(Entity *pEntity) ;　　获取实体更改

Bool　　HasChanges(Entity *pEntity) ;　　获取实体更改

long　　ImportRow(Entity *pEntity, EntityRow *Row) ;　　向实体中插入新行

long ImportData(Entity *pEntity, const char *pConstData) ; 向实体中插入新行

long ImportXMLData(Entity *pEntity, const char *pConstXMLData) ; 向实体中插入新行

long ImportDataWindow(Entity *pEntity,const char *pConstDataWindow) ;向实体中插入新行

char* ExportData(Entity *pEntity , const char *pConstEntityColumnsStr) ; 导出实体中所有行指定列的数据

char* ExportXMLData(Entity *pEntity , const char *pConstEntityColumnsStr) ; 导出实体中所有行指定列的数据

char* ExportDataWindow(Entity *pEntity , const char *pConstEntityColumnsStr) ; 导出实体中所有行指定列的数据

EntityRow* NewNullRow(Entity *pEntity) ; 在实体上新增行

EntityRow* NewRow(Entity *pEntity) ; 在实体上新增行

long AddRow(Entity *pEntity , EntityRow *pEntityRow) ; 向实体上添加新行

long RejectChanges(Entity *pEntity) ; 撤销实体更改

EntityRow* Select(Entity *pEntity, const char *pConstFilterExpression) ; 查找实体中满足条件的行

EntityRow* Find(Entity *pEntity , const char *pColumnName , const char *pValue) ; 查找实体中指定列满足条件的第一行

ArrayList* FindRows(Entity *pEntity , const char *pColumnName , const char *pValue) ; 查找实体中指定列满足条件的所有行

EntityColumn* NewColumn(Entity *pEntity) ; 在实体上新增列

long AddColumn(Entity *pEntity , EntityColumn *pEntityColumn) ; 在实体上添加列

long RemoveAt(Entity *pEntity , long rowIndex) ; 移除实体上的行

DataType GetDataType(Entity *pEntity , const char *pConstColumnName) ; 获取实体上指定列的数据类型

char* GetDBTableName(Entity *pEntity , const char *pConstColumnName) ; 获取实体上指定列对应的物理表名

char* GetDBColumnName(Entity *pEntity , const char *pConstColumnName) ; 获取实体上指定列对应的物理表列名

long GetPrimaryKeys(Entity *pEntity , long *pCount, char **ppPrimaryKeys) ; 获取实体上的主键

long AddEntitySchema(Entity *pEntity , Entity *pEntitySchema) ; 向实体上添加架构信息

long GetNewRowCount(Entity *pEntity) ; 获取实体中新增行的数量

long SetEntityName(Entity *pEntity , const char *pEntityName) ; 设置实体的名称

long SetCaption(Entity *pEntity , const char *pCaption) ; 设置实体的标题

Bool IsMatch(Entity *pEntity , const char *pConstColumnsStr) ; 判断传入的列串是否和实体的结构一致

EntityRow ：

Entity *Entity ; 行所属的实体

long ValuesCount ; 行上值的个数

RowState RowState ; 行状态

EntityValue **Values ; 行上值的集合

char* GetValue(EntityRow *pEntityRow, const char *pConstColumnName) ; 获取行上指定列的数据

long SetValue(EntityRow *pEntityRow, const char *pConstColumnName , const char* pConstValue) ; 设置行上指定列的数据

char* GetRegionValue(EntityRow *pEntityRow, const char *pConstColumnName) ; 获取行上指定列的数据

long SetRegionValue(EntityRow *pEntityRow, const char *pConstColumnName , const char* pConstRegionValue) ; 设置行上指定列的原值

ValueState GetValueState(EntityRow *pEntityRow, const char *pConstColumnName) ; 获取行上指定列的数据

long SetValueState(EntityRow *pEntityRow, const char *pConstColumnName , ValueState valueState) ; 设置行上指定列的值状态

EntityValue* NewValue(EntityRow *pEntityRow) ; 在行上新增值

long AddValue(EntityRow *pEntityRow , EntityValue *pEntityValue) ; 将值添加到行上

long ImportDataWindow(EntityRow *pEntityRow , char *pConstXMLValues) ; 向行上导入值集合

long ImportXMLData(EntityRow *pEntityRow , char *pConstDataWindowValues) ; 导入 XML 数据到实体中

char* ExportXMLData)(EntityRow *pEntityRow , char *pConstEntityColumnsStr) ; 导出实体中所有行指定列的数据

char* ExportDataWindow(EntityRow *pEntityRow , char **ppColumnNames , long columnCount) ; 导出实体中所有行指定列的数据

long AcceptChanges(EntityRow *pEntityRow) ; 提交更改

long RejectChanges(EntityRow *pEntityRow) ; 取消更改

long Delete(EntityRow *pEntityRow) ; 删除当前行

Bool IsNull(EntityRow *pEntityRow, const char *pConstColumnName) ; 判断当前行的指定列是否为空

EntityRow* Copy(EntityRow *pEntityRow) ; 拷贝当前行

EntityColumn ：

Entity *Entity ; 所属的实体

Bool AllowDBNull ; 是否允许空值

char *Caption ; 列标题，即中文名称

char *ColumnName ; 实体列名

DataType DataType ; 列数据类型

char *DefaultValue ; 缺省值

char *Expression ; 表达式

long MaxLength ; 最大长度

long Scale ; 精度，即小数点位数

Bool Unique ; 列值是否唯一

char *DBTableName ; 所属的数据库表的表名

char *DBColumnName ; 所属的数据库表列的列名

Bool IsPrimaryKey ; 是否为主键

EntityColumn* Clone(EntityColumn *pEntityColumn) ; 复制实体列的架构

long SetCaption(EntityColumn *pEntityColumn , const char *pCaption) ; 设置实体列的标题

long SetColumnName(EntityColumn *pEntityColumn , const char *pColumnName) ; 设置实体列的列名

long SetDefaultValue(EntityColumn *pEntityColumn , const char *pDefaultValue) ; 设置实体列的缺省值

long SetExpression(EntityColumn *pEntityColumn , const char *pExpression) ; 设置实体列的表达式

long SetDBTableName(EntityColumn *pEntityColumn , const char *pDBTableName) ; 设置实体列的数据库表名

long SetDBColumnName(EntityColumn *pEntityColumn,const char *pDBColumnName) ; 设置实体列的数据库列名

EntityValue ：

EntityRow	*EntityRow ;	所属的实体
char	*ColumnName ;	列名
char	*RegionValue ;	原值
char	*CurrValue ;	当前值
ValueState	ValueState ;	数据状态

EntityValue* Copy(EntityValue *pEntityValue) ; 拷贝实体值

long SetCurrValue(EntityValue *pEntityValue , const char *pCurrValue) ; 设置实体值的当前值

long SetRegionValue(EntityValue *pEntityValue , const char *pRegionValue) ; 设置实体值的原值

long SetColumnName(EntityValue *pEntityValue , const char *pColumnName) ; 设置实体值的列名

附录一，实体数据 XML 及其节点的定义：

```
<EntitySet EN = ”??” EC= ”1”>
  <Entity EN= ”???” ET= ”1” DS= ”1” RC= ”1”>
      <row RS = ”-1” CC=2更新行
            < Col CN= ”B” RV= ”” CV= ”” S= ”-2” />
            < Col CN= ”A” RV= ”” CV= ”” S= ”-2” />
            < Col CN= ”C” RV= ”” CV= ”” S= ”-1”>
            ……
      </row>
</ Entity >
```

</EntitySet>

每个 Tag 具体的含义如下：

EntitySet EN ：EntitySetName 实体集名称

EC：EntityCount 实体数

Entity EN：EntityName 实体名称

ET：EntityType 实体类型 1、查询 2、更新

DS：DynSQL SQL 类型 1、静态； 2、动态

RC：RowCount 行数

Row RS：RowState 行状态 -2、Selected -1、Unchanged 1、Added 2、Deleted 3、Modified

CC：ValueCount 值个数

Col CN：ColName 列名

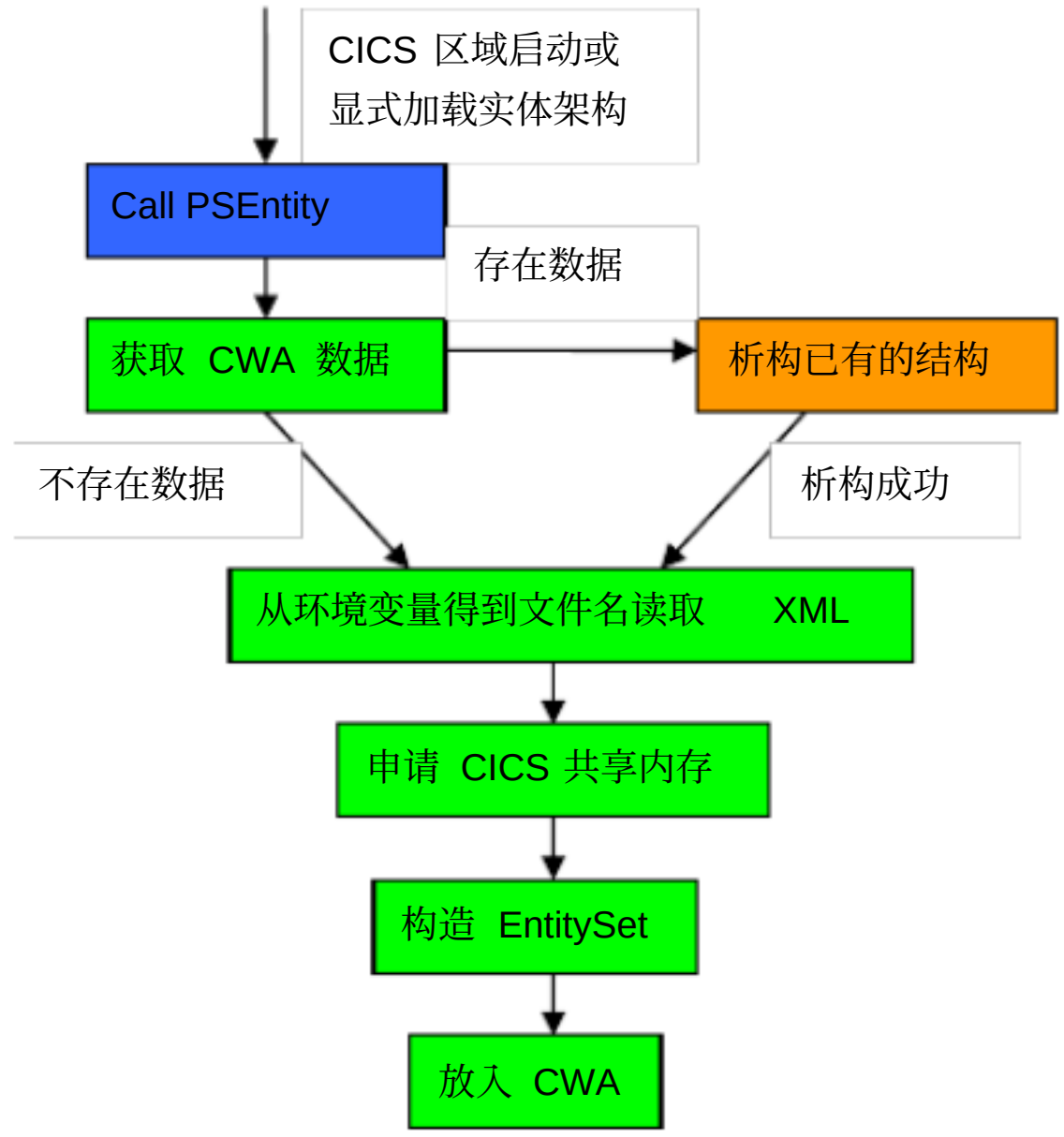
RV：RegionValue 原值

CV：CurrValue 现值

S： State 状态, -2、条件 Filter -1、回传的列 Value 1、未更改 UnChange 2、更改 Change

5.2.3 实体架构管理

实体架构管理主要是构建、管理用户定义的实体和数据库映射关系，以便用户能够随时的调用之，实体架构按照设计应该存放于数据库，在部署时能够生成为 XML 文档（ Entity 定制工具完成），以便部署到不同的平台上，其工作流程如下图所示：



结合上图，它需要完成以下功能：

XML 文件读取：

实现将 XML 文件读取到内存中，方法原型如下：

long ReadFile(const char *pFileName , char *pXMLData)

实体架构的构建：

需要实现解析 EntitySet、Entity 、Column 节点为 EntitySet 结构、 Entity 结构、 EntityColumn 结构的方法，主要的方法原型如下：

创建实体集： long CreateEntitySetSchema(const char *pDoc , EntitySet **ppEntitySet)

创建实体 ：long CreateEntitySchema(xmlDocPtr doc, xmlNodePtr currNode , Entity **ppEntity)

创建实体列： long CreateEntityColumnSchema(xmlNodePtr curNode,EntityColumn **ppEntityColumn)

实体架构的析构 ：

需要实现实体架构的析构，包括了实体集、实体、实体列的析构，主要的方法原型如下：

析构实体集： long DestroyEntitySetSchema(EntitySet *pEntitySet)

析构实体 ：long DestroyEntitySchema(Entity *pEntity)

析构实体列： long DestroyEntityColumnSchema(EntityColumn *pEntityColumn)

实体架构的维持：

实体架构将被创建在 CICS 共享内存中，同时将其地址放在全局的 CWA 中，以备用户随时取用。

用户获取实体架构的副本：

当用户需要建立实体和数据库对象之间的映射时，就需要取得实体架构的副本，实现此功能的方法原型如下：

Entity* GetEntityClone(const char* pEntityName)

此方法先从 CWA 中取得架构共享内存的地址，根据地址取得共享内存，并在系统内存中创建共享内存中指定实体名对应的实体的副本，返回给用户。

附录一、实体结构 XML 及其节点的含义：

```
<EntitySet EntitySetName= EntitySchema ” >
    <Entity Name=" " Caption=" " >
        <Columns>// 列信息
            <Column Name="KSDM" DBTableName="" DBColumnCode="" Caption=""
            DataType=" " MaxLength="" Scale="" AllowDBNull="" Expression="" IsPrimaryKey="" />
            <Column …… />
        </Columns>
    </Entity>
</EntitySet>
```

各个节点的含义如下：

EntitySet	EntitySetName	：EntitySet 名称；
Entity	Name	：Entity 名称，一般为英文名称或拼音名称；
	Caption	：Entity 标题，一般为中文名称；
Column	Name	：实体列名称，一般为英文名称或拼音名称；

DBTableName ：数据库表名；

DBColumnCode ：数据库列名；

Caption ：实体列标题，一般为中文名称；

DataType ：实体列数据类型；

MaxLength ：实体列最大长度；

Scale ：实体列精度；

AllowDBNull ：实体列是否允许为空；

Expression ：表达式，目前没有使用；

IsPrimaryKey ：此实体列是否为主键；

5.2.4 公用方法

为了方便开发人员的开发，我们需要提供一些公用的结构和函数，包括了以下内容：

字符串函数：

切分字符串： char** Split(char *pEntityString ,const char * spliter , long *pRowNum) ;

释放字符数组： long FreeStringArray(char **ppStringArray , long stringCount) ;

字符串转大写： char* strupper(char *s) ;

比较字符串： long strcasecmp(const char * src, const char * dst) ; （忽略大小写）

字符串补齐： char* lpad(char *pStr , char chr , long length) ;

日期函数： 提供将时间转换为字符串的功能

long to_char(char *pTimeStr , struct tm *pTime) ;

错误管理： 提供编译参数控制的错误消息输出方法、错误代码管理及错误代码转为错误消息的方法

错误消息输出： WriteError(pMessage1 , pMessage2) ；

错误消息转换： char* ErrorCodeToMsg(long errorCode) ;

错误消息转换： void ErrorCodeToMsgEx(char* pOutput,long errorCode,const char* pCustomMsg);

Trace ：提供 Trace 功能供开发人员调试时使用，需要通过编译参数控制

打开 Trace： OpenTrace(pGUID, pProgrameName)

写入 Trace： WriteTrace(pFile , pMessage)

关闭 Trace： CloseTrace(pFile)

清空 Trace： ClearTrace(pFile)

Log ：提供 Log 功能供开发人员在业务系统正式运行时收集系统运行状态，此功能应该由客户端控制

打开 Log： OpenLog(pFile , pProgrameName , writeLog)

写入 Log： WriteLog(pFile , pMessage)

关闭 Log： CloseLog(pFile)

清空 Log： ClearLog(pFile)

编码转换： 用于实现不同的编码方式之间的转换

long CodeConvert(const char *pFromCharset, const char *pToCharset,const char *pInBuf, long inlen,char *pOutBuf, long *outlen) ;

ArrayList ：提供链表功能，包括元素的添加、删除、插入、选取、排序等功能，其包含的属性和方法如下：

long	Count ;	链表中的元素总数
void	**Item ;	链表中的元素集合
long	Add(ArrayList *pArrayList , void *pValue) ;	向链表中添加元素
long	Clear(ArrayList *pArrayList) ;	清空链表中的元素
Bool	Contains(ArrayList *pArrayList , const void *pValue) ;	检查链表中是否存在指定的元素
long	IndexOf(ArrayList *pArrayList , const void *pValue) ;	确定链表中指定元素的索引
long	Insert(ArrayList *pArrayList , long index , void *pValue) ;	向链表中插入元素
long	Remove(ArrayList *pArrayList , const void *pValue) ;	移除链表中指定的元素
long	RemoveAt(ArrayList *pArrayList , long index) ;	移除链表中指定位置的元素
long	Sort(ArrayList *pArrayList) ;	对链表中的元素进行排序
long	Destroy(ArrayList *pArrayList) ;	销毁链表

long Equals(const void *pValue1 , const void *pValue2) ; 判断链表两个元素相等的方法

Hashtable ：提供哈稀表功能，包括添加、移除、获取键值对等操作

long Count ; 元素总数

void **Keys ; 键集合

void **Values ; 值集合

long Add(Hashtable *pHashtable , void *pKey , void *pValue) ; 向 hashtable 中添加键值对

long Clear(Hashtable *pHashtable) ; 清空 Hashtable 中的元素

Bool ContainsKey(Hashtable *pHashtable , const void *pKey) ; 确定 Hashtable 中是否包含指定的键

Bool ContainsValue(Hashtable *pHashtable , const void *pValue) ; 确定 Hashtable 中是否包含指定的值

long Remove(Hashtable *pHashtable , const void *pKey) ; 从 Hashtable 中移除指定的键值对

long Destroy(Hashtable *pHashtable) ; 销毁 Hashtable

void* GetValue(Hashtable *pHashtable , const void *pKey) ; 获取 Hashtable 中指定的键对应的值

long SetValue(Hashtable *pHashtable , const void *pKey , void *pValue) ; 设置 Hashtable 中指定的键对应的值

long Equals(const void *pValue1 , const void *pValue2) ; 判断两个元素是否相等

long KeyEquals(const void *pKey1 , const void *pKey2) ; 判断两个键是否相等

long ValueEquals(const void *pValue1 , const void *pValue2) ; 判断两个值是否相等

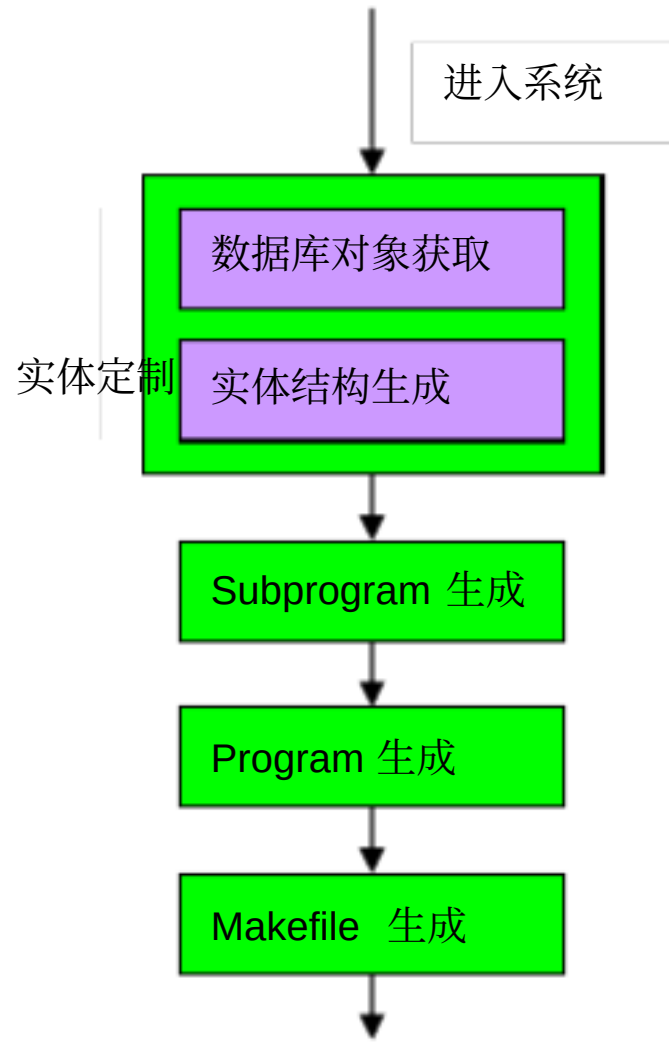
其他

比较浮点数： short floatcmp(float input , float standardValue , float Scale) ;

5.3 工具支持包

5.3.1 概述

根据工具包的功能需求，我们将工具包分为 Entity 定制、Subprogram 定制、Program 定制、Makefile 生成及开发助手五个部分，其流程如下：



下面具体介绍这五个部分的系统架构。

5.3.2 Entity 定制

实体定制是整个架构的基础，它定义了界面层和中间层之间及中间层的 Program 和 Subprogram 中间的数据传输结构，在我们的设计中它是根据数据库表结构来生成的，把一个业务处理所需要用到的信息封装在了一起，可以理解为一个视图，它主要需要完成以下几个方面的工作：

一、取得数据库中所有表的结构并展现给开发人员，需要注意的是：在每次数据库表结构发生更改时都要重新取数据库表的结构；

二、提供用户构造实体结构的功能，包括实体的新增，实体列的插入、删除、修改及实体列顺序的调整；

三、提供导出 XML 的功能（供中间层实体架构管理器使用）；

四、提供实体删除、更名、拷贝的功能，在删除和更名的过程中应该提示用户有那些 Subprogram 使用了这个实体，以方便开发人员同时更新对应的 Subprogram，避免造成实体和 Subprogram 步匹配的问题。

5.3.3 Subprogram 生成

在生成了实体之后我们需要将其和生成 Subprogram 将其和数据库对象建立映射关系，Subprogram 即为一组 Oracel PRO * C 和一组辅助 C 语句的集合，它完成对数据库表的查询、更新及调用数据库存储过程的功能，它需要完成的功能如下：

一、 SQL 语句生成：

1） 对不同的 Code 生成不同的 SQL（包括静态查询、动态查询、删除、更新、插入）；

2） 支持两表关联；

3） 支持排序；

4） 支持选择更新的列、更新条件、静态查询条件

二、存储过程生成：

1） 支持获取数据库中的存储过程；

2） 支持存储过程调用 Subpram 的自动生成。

三、 Subprogram 浏览：

1） 支持 Subprogram 的更名、删除、修改；

2） 支持 Subprogram 导出为 PC 文件；

3） 生成 Subprogram 的编译命令（可以放在 NMAKE 工具中）；

5.3.4 Program 管理

Program 即为业务组件，它包括了对 Subprogram 的调用来操作数据库、 CICS 系统调用以及用 C 语言书写的业务逻辑，它需要完成的功能如下：

一、管理 Program 和 Subprogram 的对应关系：

1） 提供查找 Subprogram 并将其加入到 Program 中的功能；

二、生成 Program 的模板：

1) 提供由 Subprogram 构造简单 Program 的功能（包括查询、更新、存储过程）；

2) 生成复杂 Program 的模板；

5.3.5 Makefile 生成

由于生成的 Subporgram 和 Program 中包含了 Oracle PRO * C 和 CICS PRO * C，所以通常通过编写 Makefile 文件的方式来编译它们，以生成可执行代码供 CICS 服务器调用；同时 CICS 服务器需要知道业务组件的位置、 XA 配置等信息，这些信息也需要以脚本的方式写在 Makefile 文件中。在本项目中我们需要支持多平台运行，所以需要生成不同平台的 Makefile 文件。其需要完成的功能如下：

一、生成 Windows 平台的 MakeFile 文件；

1） Debug 版本生成：

XA 配置；

Subprogram 编译命令；

Program 编译命令；

2） Release版本生成：

XA 配置；

Subprogram 编译命令；

Program 编译命令；

3） 安装、卸载、更新 Program 和事务的命令。

二、生成 AIX 版本的 MakeFile 文件；

1） Debug 版本生成：

XA 配置；

Subprogram 编译命令；

Program 编译命令；

2） Release版本生成：

XA 配置；

Subprogram 编译命令；

Program 编译命令；

3） 安装、卸载、更新 Program 和事务的命令。

5.3.6 开发助手

Comment [Z4]: 需要界面组补充