Call Redirection System
version 0.1
Implementation Description

Dominik Radziszowski
Bartosz Klimek
e-mail:{radzisz,bartekk}@student.uci.agh.edu.pl

University of Mining and Metallurgy
Cracow, Poland

January 1999

Contents

1 Introduction **3**

2 Audio Device Repository **4**
2.1 Functional Specification 4
2.1.1 Functionality Description 4
2.1.2 Entity 4
2.1.2.1 Interface Description 4
2.1.2.2 Equality of Entities 6
2.1.3 Repository 6
2.1.3.1 Interface Description 6
2.1.3.2 Relationships 7
2.1.4 Implementation Classes 7
2.1.4.1 Audio_Device_I 7
2.1.4.2 Audio_Device_Rep_I 7
2.1.4.3 Repository Server 8
2.1.5 Persistency 8
2.1.6 CORBA Services Usage 8
2.1.7 Life Cycle Management 8
2.2 Implementation Specification 9
2.2.1 Source Code Organization 9

3 Property Record Repository **10**
3.1 Functional Specification 10
3.1.1 Functionality Description 10
3.1.2 Entity 10
3.1.2.1 Interface Description 10
3.1.2.2 Equality of Entities 14
3.1.2.3 Masks 14
3.1.3 Repository 17
3.1.3.1 Interface Description 17
3.1.3.2 Relationships 18
3.1.4 Implementation Classes 18
3.1.4.1 Property_Record_I 18
3.1.4.2 Call_Mask_I 19
3.1.4.3 General_Mask_I 19
3.1.4.4 Caller_Mask_I 19
3.1.4.5 Location_Mask_I 19
3.1.4.6 Property_Record_Rep_I 19
3.1.4.7 Repository Server 20

3.1.5 Persistency 20
3.1.6 CORBA Services Usage 20
3.1.7 Life Cycle Management 20
3.2 Implementation Specification 20
3.2.1 Source Code Organization 20

4 Call Redirection Manager **22**
4.1 Functional Specification 22
4.1.1 Functionality Description 22
4.1.2 Server 22
4.1.2.1 Interface Description 22
4.1.2.2 Relationships 22
4.1.3 Implementation Class 23
4.1.4 Persistency 23
4.1.5 CORBA Services Usage 23
4.1.6 Life Cycle Management 23
4.2 Implementation Specification 23
4.2.1 Source Code Organization 23

Chapter 1

Introduction

This document decribes the implementation of Call Redirection System (CRS), version 0.1. The primary objective of this version is to set up the software and hardware independent part of the system, which is a base for future development. Currently we lay stress on building a simple but complete system so that we can test how the conception of computer-controlled call redirection works in practice, and find any problems that may occur.

These elements of CRS have been implemented:

- Audio Device Repository
- Property Record Repository
- Call Redirection Manager

The following chapters contain both interface specification and implementation details of those elements.

Chapter 2

Audio Device Repository

2.1 Functional Specification

2.1.1 Functionality Description

Audio Device is an abstract device which can be used by a human being to talk to another human being. An obvious example of such device is a telephone, but it can also be a properly equipped and configured work station.

This repository is a database of audio devices that can be used by Call Redirection System. Every audio device in the repository is related to a location, which means that if the receiver of an incoming call is found in that location and it is not his or her home location, the call is redirected to that audio device by default.

2.1.2 Entity

2.1.2.1 Interface Description

```
#ifndef __AUDIO_DEVICE_IDL__
#define __AUDIO_DEVICE_IDL__

interface Audio_Device : Domain_Repository::Static_Data_Observed_E {

   enum Audio_Device_Type {PHONE, COMPUTER};

   struct Audio_Device_Type_Pattern {
      boolean is_any;
      Audio_Device_Type value;
   };

   struct Description {
      Domain_Repository::Repository_Item_Id item_id;
      unsigned long adid;
      Domain_Repository::Location_Description location;
      Audio_Device_Type type;
      string audio_device_number;
   };
```

```
  typedef sequence<Description> Descriptions;

  struct Value_Description {
    Domain_Repository::Repository_Item_Id item_id;
    unsigned long adid;
    Domain_Repository::Location_Description::Value_Description location;
    Audio_Device_Type type;
    string audio_device_number;
  };

  typedef sequence<Value_Description> Value_Descriptions;

  struct Pattern {
    boolean is_any;
    Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
    Domain_Repository::Unsigned_Long_Pattern adid_pattern;
    Domain_Repository::Repository_Item_Id_Pattern location_pattern;
    Audio_Device_Type_Pattern type_pattern;
    Domain_Repository::String_Pattern audio_device_number_pattern;
  };

    readonly attribute Description descr;
    readonly attribute Value_Description value_descr;
    readonly attribute Domain_Repository::Repository_Item_Id item_id;
    readonly attribute unsigned long adid;
    readonly attribute Domain_Repository::Location_Description location;
    readonly attribute Audio_Device_Type type;
    readonly attribute string audio_device_number;
 };

#endif                                          /* __AUDIO_DEVICE_IDL__ */
```

The *Description* structure of the *Audio Device* entity contains:

- *item_id* — Repository Item Id of the entity
- *adid* — Audio Device Id
- *location* — a reference to a location description for which this device is default
- *type* — this determines the type of the audio device (telephone or computer)
- *audio_device_number* — a string that can be interpreted as a telephone number (it this is a telephone), or an IP address or symbolic name (if this is a computer)

Value_Description structure is similar to *Description*. The only difference is that the type of the *location* field is *Location_Description::Value_Discription*.

An *Audio Device* entity supports the following attributes:

- *descr* — get description of a given entity
- *value_descr* — get value description of a given entity

- *item_id* — get Repository Item Id of the entity
- *adid* — get Audio Device Id
- *location* — get the location for which this device is default
- *type* — get the type of the entity
- *audio_device_number* — get the number of the actual device

2.1.2.2 Equality of Entities

Two entities are considered equal if their Audio Device Ids are equal.

2.1.3 Repository

2.1.3.1 Interface Description

```
#ifndef __AUDIO_DEVICE_REP_IDL__
#define __AUDIO_DEVICE_REP_IDL__ 1

interface Audio_Device_Rep : Domain_Repository::Static_Data_Observed_R {

    Audio_Devices find(in Audio_Device::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    Audio_Device::Value_Descriptions find_values(
        in Audio_Device::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);

    Audio_Device::Query_Results
      complex_query( in Audio_Device::Entity_Set es,
                     in Domain_Repository::Query_Options qo,
                     in Domain_Repository::Static_Data_Observer obs)
      raises (Domain_Repository::Search_Not_Implemented,
              Domain_Repository::Destination_Unreachable);
};

#endif                                  /* __AUDIO_DEVICE_REP_IDL__ */
```

```
#ifndef __AUDIO_DEVICE_REP_MANAGER_IDL__
#define __AUDIO_DEVICE_REP_MANAGER_IDL__ 1

interface Audio_Device_Rep_Manager {

    exception Invalid_Data { };

    readonly attribute Domain_Repository::Audio_Device_Rep rep;
    readonly attribute Domain_Repository::Audio_Devices audio_device_list;
```

```
    Domain_Repository::Audio_Device add(
        in Domain_Repository::Audio_Device::Description data_record)
        raises(Domain_Repository::Duplicate_Data_Record,
               Invalid_Data);

    void remove(
        in Domain_Repository::Audio_Device object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);

    void update(
        in Domain_Repository::Audio_Device object_to_update,
        in Domain_Repository::Audio_Device::Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Duplicate_Data_Record,
               Invalid_Data);
};

#endif                           /* __AUDIO_DEVICE_REP_MANAGER_IDL__ */
```

2.1.3.2 Relationships

The *Audio Device Repository* is related to *Location Description Repository*.

2.1.4 Implementation Classes

2.1.4.1 Audio_Device_I

The class implements interfaces of *Audio_Device* and *System_Static_Data_Observed_E*.

2.1.4.2 Audio_Device_Rep_I

The class uses the building blocks of the repository template to implement functionality of *Audio_Device_Rep* and *System_Static_Data_Observed_R* interfaces. It inherits from the *Basic_Rep_I <Audio_Device_Types, AD_SDO_Types>* class, where the template parameter classes are defined as follows:

```
class AD_SDO_Types {

public:

  typedef Domain_Repository::Static_Data_Observed_R Observed;
  typedef Domain_Repository::Static_Data_Observer Observer;
  typedef Domain_Repository::Static_Data_Observer_ptr Observer_ptr;
  typedef Domain_Repository::Static_Data_Observed_E::
    Static_Data_Observers Observers;
};

class Audio_Device_Types {
```

```
public:

   typedef Domain_Repository::Audio_Device Item;
   typedef Domain_Repository::Audio_Device_ptr Item_ptr;
   typedef Domain_Repository::Audio_Devices Items;
   typedef Audio_Device_I Item_I;
   typedef Domain_Repository::Audio_Device::Description Initial_Data;
   typedef Domain_Repository::Static_Data_Observer::
      Static_Data_Object Data_Object;
};
```

The class overwrites the following template method:

- *private_find* — invokes the function matching entities with the specified pattern. Its implementation is not complex as audio device description is simple.
- *validate_add_data* — checks if the *location* field of the data record is not **nil**.
- *validate_update_data* — checks if the *location* field of the data record is not **nil**.

2.1.4.3 Repository Server

This is the repository server implementation class based on the template of the repository server.

- *create_objects* — creates instances of *Audio_Device_I* and calls *init* on the *Audio_Device_Rep_I* instance
- *delete_objects* — deletes the instance of the created objects.

2.1.5 Persistency

Currently the persistent state of the repository is stored in files. As *Audio Device* is related to *Location Description*, the implementation object of this interface remembers the repository item id of the appropriate location description. It is used when the object is being saved, and then, after a repository startup, when the location description is resolved.

2.1.6 CORBA Services Usage

Current implementation uses only naming service.

2.1.7 Life Cycle Management

The repository stores its reference in the naming service using ABng Server Manager. Currently it must be launched by hand.

2.2 Implementation Specification

2.2.1 Source Code Organization

- *audevrep/audio_device_i.hh*
  audevrep/audio_device_i.cc —
  implementation of the *Audio_Device_I* class

- *audevrep/audio_device_rep_i.hh*
  audevrep/audio_device_rep_i.cc —
  implementation of the *Audio_Device_Rep_I* class

- *audevrep/audio_device_rep_server_i.hh*
  audevrep/audio_device_rep_server_i.cc —
  implementation of the *Audio_Device_Rep_Server_I* class

- *audevrep/audio_device_rep_main.cc* —
  implementation of the main function.

Chapter 3

Property Record Repository

3.1 Functional Specification

3.1.1 Functionality Description

Property Record is an entity that keeps properties of the Call Redirection System, determining the way it replies to incoming calls for a given user. The record can be configured by the user and by the administrator. Each property record contains (currently) three objects called "masks". The role of them is to find the reply for the incoming call if they can. Each of them is different and is sensitive to different properties of the call.

3.1.2 Entity

3.1.2.1 Interface Description

```
#ifndef _PROPERTY_RECORD_IDL
#define _PROPERTY_RECORD_IDL 1

interface Property_Record : Domain_Repository::Static_Data_Observed_E {

  enum Mask_Type {
    GENERAL_M,
    CALLER_M,
    LOCATION_M
  };

  typedef Mask_Type Masks_Order[3];

  union Gen_Mask_Description switch(boolean)
    {
    case 0 :
      // If this field has been chosen,
      // initialize the mask with default values (while initializing)
      // or do not update mask (while updating).
      // Mask's methods can been used to modify it.
      long dummy;
    case 1 :
```

```
        // if this field has been chosen,
        // it contains a reference to a mask (should be valid!)
        // whose fields will be used to initialize or update the mask
        General_Mask ref;
    };

union Caller_Mask_Description switch(boolean)
    {
    case 0 :
        long dummy;
    case 1 :
        Caller_Mask ref;
    };

union Loc_Mask_Description switch(boolean)
    {
    case 0 :
        long dummy;
    case 1 :
        Location_Mask ref;
    };


struct Description {

    Domain_Repository::Repository_Item_Id item_id;
    unsigned long lid;
    string uid;
    Prop_Rec_Rep_Types::Reply_Type outside;
    // here are references to objects permanently
    // related to this object
    // updating Property_Record cannot change these references
    // if you update Property_Record with references of objects
    // taken from somewhere else, their properties will be just
    // copied to the original objects
    // if any of these references is null, it will be ignored
    // let's hope this will work
    Gen_Mask_Description gen_mask;
    Caller_Mask_Description caller_mask;
    Loc_Mask_Description loc_mask;
    // order in which masks are applied
    // masks_order[0] is applied first etc.
    Masks_Order order;

};

typedef sequence<Description> Descriptions;
```

```
struct Value_Description
{
   Domain_Repository::Repository_Item_Id item_id;
   unsigned long lid;
   string uid;
   Prop_Rec_Rep_Types::Reply_Type outside;
   General_Mask::Value_Description gen_mask;
   Caller_Mask::Value_Description caller_mask;
   Location_Mask::Value_Description loc_mask;
   Masks_Order order;
};

typedef sequence<Value_Description> Value_Descriptions;

struct Mask_Type_Pattern
{
   boolean is_any;
   Mask_Type value;
};

struct Masks_Order_Pattern
{
   boolean is_any;
   Mask_Type_Pattern pattern[3];
};

struct Pattern {
   boolean is_any;
   Domain_Repository::Repository_Item_Id_Pattern item_id_pattern;
   Domain_Repository::Unsigned_Long_Pattern lid_pattern;
   Domain_Repository::String_Pattern uid_pattern;
   Masks_Order_Pattern order_pattern;
};

readonly attribute Description descr;
readonly attribute Value_Description value_descr;
readonly attribute Domain_Repository::Repository_Item_Id item_id;
readonly attribute unsigned long lid;
readonly attribute string uid;
readonly attribute Prop_Rec_Rep_Types::Reply_Type outside;
readonly attribute General_Mask gen_msk;
readonly attribute Caller_Mask caller_msk;
readonly attribute Location_Mask loc_msk;
readonly attribute Masks_Order order;

exception Invalid_Priority { };

// priority should be 1..3
```

```
  Call_Mask get_mask_with_priority(in unsigned short priority)
    raises(Invalid_Priority);

  void set_mask_priority(in Mask_Type mask, in unsigned short priority)
    raises(Invalid_Priority);

  unsigned short get_mask_priority(in Mask_Type mask);

};

#endif
```

The *Description* structure of the *Property Record* entity contains:

- *item_id* — Repository Item Id of the entity
- *lid* — Logical Id which identifies a user in CRS; it is also his or her phone number (or something related to it nonambiguously)
- *uid* — User Id of the user
- *outside* — a reply that will be returned when the user is away
- *gen_mask* — a reference to an object implementing the *General_Mask* interface
- *caller_mask* — a reference to an object implementing the *Caller_Mask* interface
- *loc_mask* — a reference to an object implementing the *Location_Mask* interface
- *order* — an array which specifies the order in which masks are applied to incoming calls, order[0] is applied first etc.

Value_Description structure is similar to *Description*. The only difference is that the types of **_Mask* fields are **_Mask::Value_Description*.

An *Property Record* entity supports the following attributes and operations:

- *descr* — get description of a given entity; **_mask* fields are references to the masks contained within the record
- *value_descr* — get value description of a given entity
- *item_id* — get Repository Item Id of the entity
- *lid* — get Logical Id
- *uid* — get User Id
- *outside* — get the reply for the case of user being away
- *gen_mask* — get the general mask related to the record
- *caller_mask* — get the caller mask related to the record
- *loc_mask* — get the location mask related to the record

- *get_mask_with_priority* — get the mask with the given priority (e.g. mask with priority 1 is applied first to incoming calls)
- *set_mask_priority* — change the priority of the given mask
- *get_mask_priority* — get the priority of the given mask

3.1.2.2 Equality of Entities

Two entities are considered equal if their Logical Ids are equal.

3.1.2.3 Masks

Call_Mask is an interface for all masks.

```
interface Call_Mask {

  exception Invalid_Data { };

  // determines whether the mask should be taken into account
  // when finding replies
  attribute boolean enabled;

  // location is receiver's current location
  Prop_Rec_Rep_Types::Reply_Type
    find_reply_type(in Prop_Rec_Rep_Types::Call c,
                    in Domain_Repository::Location_Description
                    location)
    raises(Invalid_Data);

};
```

It has one attribute, *enabled*. If it is 0, the mask does not work for any call.

find_reply_type operation tries to find a reply for the given call. Its second parameter, *location*, describes the receiver's current location. It can be **nil** if the receiver is away.

General_Mask is a mask which replies to all calls the same way, determined by attribute *reply*.

```
interface General_Mask : Call_Mask {

  // reply type returned for any call
  attribute Prop_Rec_Rep_Types::Reply_Type reply;


  struct Value_Description {
```

```
        boolean enabled;
        Prop_Rec_Rep_Types::Reply_Type reply;

    };

};
```

Caller_Mask checks the caller's number and, if it is found in the list stored in the object implementing the mask, it returns the reply prepared for the number.

```
interface Caller_Mask : Call_Mask {

    exception Already_Exists { };
    exception No_Such_Number { };

    // adds the number to a set of numbers to which this masks has replies
    // reply_type will be returned for this number
    void add_number(in string number, in Prop_Rec_Rep_Types::Reply_Type reply)
        raises(Already_Exists, Invalid_Data);

    void remove_number(in string number)
        raises(Invalid_Data);

    void update_number(in string number, in Prop_Rec_Rep_Types::Reply_Type reply)
        raises(No_Such_Number, Invalid_Data);

    // an element of the mapping number -> reply_type
    struct Item {

        string number;

        Prop_Rec_Rep_Types::Reply_Type reply;

    };

    typedef sequence<Item> Items;

    Items get_replies();

    struct Value_Description {

        boolean enabled;
        Items item_list;

    };

};
```

This interface specifies three operations for manipulating the list of numbers:

- *add_number* — adds the given number to the list and associates the given reply with it; if the number has already been added, it throws an exception (*Already_Exists*)
- *remove_number* — removes the given number from the list (if it is found there)
- *update_number* — sets the reply for the given number if the number is found in the list; otherwise it throws *No_Such_Number*.

get_replies operation returns the list of the numbers and replies associated with them.

Location_Mask checks the receiver's current location and, if it is found in the list stored in the object implementing the mask, it returns the reply prepared for the location.

```
interface Location_Mask : Call_Mask {

  exception Already_Exists { };
  exception No_Such_Location { };

  // adds the location to a set of locations to which this masks has replies
  // reply_type will be returned for this number
  void add_location(in Domain_Repository::Location_Description location,
                    in Prop_Rec_Rep_Types::Reply_Type reply)
    raises(Already_Exists, Invalid_Data);

  void remove_location(in Domain_Repository::Location_Description location)
    raises(Invalid_Data);

  void update_location(in Domain_Repository::Location_Description location,
                       in Prop_Rec_Rep_Types::Reply_Type reply)
    raises(No_Such_Location, Invalid_Data);

  // an element of the mapping location -> reply_type
  struct Item {

    Domain_Repository::Location_Description location;

    Prop_Rec_Rep_Types::Reply_Type reply;

  };

  typedef sequence<Item> Items;

  Items get_replies();

  struct Value_Description {
```

```
    boolean enabled;
    Items item_list;

  };

};
```

This interface specifies three operations for manipulating the list of locations:

- *add_location* — adds the given location to the list and associates the given reply with it; if the location has already been added, it throws an exception (*Already_Exists*)
- *remove_location* — removes the given location from the list (if it is found there)
- *update_location* — sets the reply for the given location if the location is found in the list; otherwise it throws *No_Such_Location*.

get_replies operation returns the list of the locations and replies associated with them.

3.1.3 Repository

3.1.3.1 Interface Description

```
#ifndef __PROP_RECORD_REP_IDL__
#define __PROP_RECORD_REP_IDL__ 1

interface Property_Record_Rep : Domain_Repository::Static_Data_Observed_R {

    Property_Records find(in Property_Record::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);
    Property_Record::Value_Descriptions find_values(
        in Property_Record::Pattern data_record_pattern)
        raises(Domain_Repository::Search_Not_Implemented);

    Property_Record::Query_Results
      complex_query( in Property_Record::Entity_Set es,
                     in Domain_Repository::Query_Options qo,
                     in Domain_Repository::Static_Data_Observer obs)
      raises (Domain_Repository::Search_Not_Implemented,
              Domain_Repository::Destination_Unreachable);
};

#endif                                  /* __PROP_RECORD_REP_IDL__ */



#ifndef __PROP_RECORD_REP_MANAGER_IDL__
#define __PROP_RECORD_REP_MANAGER_IDL__ 1
```

```
interface Property_Record_Rep_Manager {

    readonly attribute Domain_Repository::Property_Record_Rep rep;
    readonly attribute Domain_Repository::Property_Records prop_record_list;

    exception Invalid_Order { };

    Domain_Repository::
      Property_Record add(in Domain_Repository::
                             Property_Record::Description data_record)
      raises(Domain_Repository::Duplicate_Data_Record,
             Invalid_Order);

    void remove(
        in Domain_Repository::Property_Record object_to_remove)
        raises(Domain_Repository::Unknown_Object_Ref);

    void update(
        in Domain_Repository::Property_Record object_to_update,
        in Domain_Repository::Property_Record::
        Description data_record_pattern)
        raises(Domain_Repository::Unknown_Object_Ref,
               Domain_Repository::Duplicate_Data_Record, Invalid_Order);
};

#endif                          /* __PROP_RECORD_REP_MANAGER_IDL__ */
```

Both *add* and *update* operations of the *Propery_Record_Rep_Manager* assume that each **_mask* field of the *data_record* and *data_record_pattern*, respectively, is one of the following:

- a reference to an existing object implementing the appropriate mask; if it is the mask contained within the record being updated, this field will be ignored; otherwise the mask will be deep-copied
- a value of type **long** — in this case the field will be ignored, which means that:
  - if a new record is being added, the mask will be created with some default values
  - if a record is being updated, the mask will not be changed.

3.1.3.2 Relationships

The *Property Record Repository* is related to *Location Description Repository* because some replies can contain references to the items from that repository.

3.1.4 Implementation Classes

3.1.4.1 Property_Record_I

The class implements interfaces of *Property_Record* and *System_Static_Data_Observed_E*.

3.1.4.2 Call_Mask_I

This is an abstract class that implements (but in sense of IDL only) interface of *Call_Mask*. It cannot be instantiated because it declares the *find_reply_type* method as pure virtual.

3.1.4.3 General_Mask_I

The class implements the interfaces of *Call_Mask* and *General_Mask*.

3.1.4.4 Caller_Mask_I

The class implements the interfaces of *Call_Mask* and *Caller_Mask*.

3.1.4.5 Location_Mask_I

The class implements the interfaces of *Call_Mask* and *Location_Mask*.

3.1.4.6 Property_Record_Rep_I

The class uses the building blocks of the repository template to implement functionality of *Property_Record_Rep* and *System_Static_Data_Observed_R* interfaces. It inherits from the *Basic_Rep_I <Property_Record_Types, PR_SDO_Types>* class, where the template parameter classes are defined as follows:

```
class PR_SDO_Types {

public:

   typedef Domain_Repository::Static_Data_Observed_R Observed;
   typedef Domain_Repository::Static_Data_Observer Observer;
   typedef Domain_Repository::Static_Data_Observer_ptr Observer_ptr;
   typedef Domain_Repository::Static_Data_Observed_E::
      Static_Data_Observers Observers;
};

class Property_Record_Types {

public:

   typedef Domain_Repository::Property_Record Item;
   typedef Domain_Repository::Property_Record_ptr Item_ptr;
   typedef Domain_Repository::Property_Records Items;
   typedef Property_Record_I Item_I;
   typedef Domain_Repository::Property_Record::Description Initial_Data;
   typedef Domain_Repository::Static_Data_Observer::
      Static_Data_Object Data_Object;
};
```

The class overwrites the following template method:

- *private_find* — invokes the function matching entities with the specified pattern.
- *validate_add_data* — checks if the order of masks in the data record is valid
- *validate_update_data* — checks if the order of masks in the data record is valid

3.1.4.7 Repository Server

This is the repository server implementation class based on the template of the repository server.

- *create_objects* — creates instances of *Property_Record_I* and calls *init* on the *Property_Record_Rep_I* instance
- *delete_objects* — deletes the instance of the created objects.

3.1.5 Persistency

Currently the persistent state of the repository is stored in files. As *Property Record* is related to *Location Description*, the implementation object of this interface remembers the repository item id of the appropriate location description. It is used when the object is being saved, and then, after a repository startup, when the location description is resolved.

3.1.6 CORBA Services Usage

Current implementation uses only naming service.

3.1.7 Life Cycle Management

The repository stores its reference in the naming service using ABng Server Manager. Currently it must be launched by hand.

3.2 Implementation Specification

3.2.1 Source Code Organization

- *prrecrep/prop_record_i.hh*
  prrecrep/prop_record_i.cc —
  implementation of the *Property_Record_I* class
- *prrecrep/call_mask_i.hh*
  prrecrep/call_mask_i.cc —
  implementation of the *Call_Mask_I*, *General_Mask_I*, *Caller_Mask_I* and *Location_Mask_I* classes
- *prrecrep/prop_record_rep_i.hh*
  prrecrep/prop_record_rep_i.cc —
  implementation of the *Property_Record_Rep_I* class
- *prrecrep/prop_record_rep_server_i.hh*
  prrecrep/prop_record_rep_server_i.cc —
  implementation of the *Property_Record_Rep_Server_I* class

- *prrecrep/prop_record_rep_main.cc* —
  implementation of the main function
- *prrecrep/crs_utils.hh*
  prrecrep/crs_utlls.cc —
  utilities

Chapter 4

Call Redirection Manager

4.1 Functional Specification

4.1.1 Functionality Description

Call Redirection Manager is a simple server supporting currently only one operation. Its function is to find replies for incoming calls. It implements the algorithm described in the specification of Call Redirection System. It will be used by the Telephone Exchange Interface.

4.1.2 Server

4.1.2.1 Interface Description

```
#ifndef __CALL_REDIRECTION_SYSTEM_IDL
#define __CALL_REDIRECTION_SYSTEM_IDL 1

  interface Call_Redirection_Manager
    {
      exception Invalid_Data { };
      exception Unknown_LID { };
      exception System_Failure { };

      Command find_reply(in string number, in unsigned long lid,
           in unsigned long call_id /* currently not used */)
        raises(Invalid_Data, System_Failure, Unknown_LID);

    };

};

#endif
```

4.1.2.2 Relationships

The *Call Redirection Manager* is related to *Property Record Repository*, *Audio Device Repository* and *View Managers*.

4.1.3 Implementation Class

The *Call_Redirection_Manager_I* class implements the interfaces of *Call_Redirection_Manager*.

4.1.4 Persistency

The server has no variables the state of which should be stored.

4.1.5 CORBA Services Usage

Current implementation uses only naming service.

4.1.6 Life Cycle Management

The server is currently available through Orbix daemon and its identification string is "CRMgr". In the future the naming service will be used. The server must be launched by hand.

4.2 Implementation Specification

4.2.1 Source Code Organization

- *crm/call_redir_mgr_i.hh*
  crm/call_redir_mgr_i.cc —
  implementation of the *Call_Redirection_Manager_I* class

- *crm/crm_main.cc* —
  implementation of the main function.